Personal Communications
AS/400 and 3270 for Windows** 95

IBM

Client/Server Communications Programming

Version 4.1

SC31-8208-00

Personal Communications
AS/400 and 3270 for Windows** 95

IBM

Client/Server Communications Programming

Version 4.1

SC31-8208-00

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page ix.

First Edition (March 1996)

This level applies to Version 4.1 Personal Communications and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication, or you may address your comments to the following address:

International Business Machines Corporation
Department CGM
P.O. Box 12195
Research Triangle Park, North Carolina
27709-2195

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

© **Copyright International Business Machines Corporation 1994, 1996. 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

Notices ix
Trademarks x

About This Book xi
Who Should Read This Book xi
How to Use This Book xi
Double-Byte Character Set Support xiii
Where to Find More Information xiii

Part 1. APPC API 1

Chapter 1. Introducing APPC 3
SNA Communications Support 3
SNA LU Type 6.2 Support 4

Chapter 2. Fundamental APPC Concepts 5
What Is a Transaction Program? 5
APPC Transaction Programs 5
CPI Communications Transaction Programs 6
What Is a Logical Unit? 6
LU Types 6
Dependent and Independent LUs 6
What Is a Session? 7
What Is a Conversation? 7

Chapter 3. Using the Attach Manager 11
Identifying the Transaction Program Name on Both Machines 12
DEFINE_TP 13
Starting the Attach Manager 15
Starting Programs with the Attach Manager 16
Differentiating between an Application and a Transaction Program 16
Matching Incoming Allocation Requests with RECEIVE_ALLOCATE Verbs . . 17
Nonqueued Programs 17
Queued Programs 17
Transaction Program Definitions 20
Defining Conversation Attributes 20
Sync Level 20
Conversation Type and Style 21
Conversation Security for an Incoming Allocation Request 21
Conversation Security for an Outgoing Allocation Request 22
TPs and Local LUs 22

Chapter 4. Implementing APPC Transaction Programs 23
Writing Transaction Programs 23
Option Sets Supported 23
Queue-level Nonblocking 24
Default Local LU Pool 26

Chapter 5. Entry Points 27

© Copyright IBM Corp. 1994, 1996

APPC . . . 27
WinAsyncAPPC() . . . 28
WinAsyncAPPCEx() . . . 30
WinAPPCCancelAsyncRequest() . . . 32
WinAPPCCancelBlockingCall() . . . 33
WinAPPCCleanup() . . . 34
WinAPPCIsBlocking() . . . 35
WinAPPCStartup() . . . 36
WinAPPCSetBlockingHook() . . . 37
WinAPPCUnhookBlockingHook() . . . 38
GetAppcConfig() . . . 39
GetAppcReturnCode() . . . 40

Chapter 6. APPC Verbs . . . 43
Verb Control Blocks . . . 43
Common Fields . . . 43
APPC API Support . . . 43
Verbs Supported . . . 43
Indication Verb Control Blocks . . . 45
INITIALIZE_SESSION_LIMIT . . . 45
RESET_SESSION_LIMIT . . . 48
Type Independent Verb Control Blocks . . . 51
GET_TP_PROPERTIES . . . 51
GET_TYPE . . . 54
RECEIVE_ALLOCATE . . . 56
TP_ENDED . . . 58
TP_STARTED . . . 60
Basic Conversation Verb Control Blocks . . . 62
ALLOCATE . . . 62
CONFIRM . . . 66
CONFIRMED . . . 68
DEALLOCATE . . . 70
FLUSH . . . 73
GET_ATTRIBUTES . . . 75
PREPARE_TO_RECEIVE . . . 78
RECEIVE_AND_POST . . . 80
RECEIVE_AND_WAIT . . . 84
RECEIVE_IMMEDIATE . . . 87
REQUEST_TO_SEND . . . 90
SEND_CONVERSATION . . . 92
SEND_DATA . . . 95
SEND_ERROR . . . 98
TEST_RTS . . . 101
TEST_RTS_AND_POST . . . 103
Mapped Conversation Verb Control Blocks . . . 105
MC_ALLOCATE . . . 105
MC_CONFIRM . . . 109
MC_CONFIRMED . . . 111
MC_DEALLOCATE . . . 113
MC_FLUSH . . . 115
MC_GET_ATTRIBUTES . . . 117
MC_PREPARE_TO_RECEIVE . . . 120
MC_RECEIVE_AND_POST . . . 122
MC_RECEIVE_AND_WAIT . . . 125

MC_RECEIVE_IMMEDIATE 128
MC_REQUEST_TO_SEND 131
MC_SEND_DATA 136
MC_SEND_ERROR 139
MC_TEST_RTS 141
MC_TEST_RTS_AND_POST 143

Part 2. Common Services API 145

Chapter 7. Procedure Entry Points and Common Services Verbs (CSV) 147
Procedure Entry Points 147
ACSSVC 147
WinCSV() 147
WinCSVCleanup() 148
WinAsyncCSV() 149
WinCSVStartup() 150
GetCsvReturnCode() 151
TrnsDt 152
Common Services Verbs 156
GET_CP_CONVERT_TABLE 157
CONVERT 160

Part 3. LUA API 163

Chapter 8. Introducing the IBM Conventional LU Application 165
Understanding LUA and SNA 165
Connection Capabilities 165
LUA Application Programs 165
LUA Verbs 165
LUs, Local LUs, and Partner LUs 166
SNA Layers 166
Data Link Control Layer 166
Path Control Layer 166
Transmission Control Layer 167
Data Flow Control Layer 167
Presentation Services Layer 167
Using SNA Sessions 167
Prerequisites to an SNA Session 168
Starting Sessions 168
Transferring Data on an LU-LU Session 169
Stopping Sessions 169
Disconnecting the Host Link 170
Message Numbers 170
Restarting and Resynchronizing a Session 170
Using Protocols to Control Requests and Responses 171
Using the Pacing Protocol 171
Using the Half-Duplex Contention/Flip-Flop Protocol 172
Using the Bracket Protocol 172
Using the Data-Chaining Protocol 173
Configuration 173
LUA Verbs 174
Verb Summary 174

Asynchronous Verb Completion 174
Sample LUA Communication Sequence 175
LUA Compatibility 176
BIND Checking 176
Negative Responses and SNA Sense Codes 177
Pacing 178
Segmentation 178
Courtesy Acknowledgments 178
Purging Data to End of Chain 178
Configuration Information 179
LUA LU Pool (optional) 179

Chapter 9. Understanding LUA 181
RUI Sessions 181
Issuing RUI Verbs 181

Chapter 10. Writing LUA Application Programs 183
Calling the Dynamic Link Libraries 183
Procedure Entry Points 183
WinRUI 183
WinRUICleanup() 185
WinRUIGetLastInitStatus() 186
WinRUIStartup() 189
RUI() 190
GetLuaReturnCode() 191
Using the DLL Interface 193
Load-Time Linking 193
Run-Time Linking 193
Multiple Processes 193
Multiple Threads 193
LUA Verb Postings 193
Converting to EBCDIC from ASCII 194

Chapter 11. RUI Verbs 195
LUA Verb Control Block Format 195
Common Verb Header 196
RUI_BID Data Structure 200
RUI_BID 201
RUI_INIT 206
RUI_PURGE 210
RUI_INIT_STATUS 213
RUI_READ 214
RUI_TERM 220
RUI_WRITE 223

Appendix A. APPC Common Return Codes 229

Appendix B. APPC Conversation State Transitions 233

Glossary 237

Index 273

Figures

1. Personal Communications APPC Implementation 3
2. A Session between Two LUs 7
3. A Conversation between Two Transaction Programs 8
4. Parallel Sessions between LUs 9
5. Attach Manager Function in APPC 11
6. Example of a Language Statement 246
7. Example of an NCP Definition Statement 246
8. Example of a VTAM Definition Statement 246

Tables

1. Verb Processing and Transaction Program Name Configuration 19
2. Permissible LU-LU FMPs and TSPs for the RUI 181
3. RUI Verb Conditions 182
4. APPC Conversation State Transitions 234

© Copyright IBM Corp. 1994, 1996

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program or service is not intended to state or imply that only IBM's product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter 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, New York 10594
United States of America

For license inquires regarding double-byte character set (DBCS) information, contact:

IBM World Trade
Asia Corporation Commercial Relations
IBM Kamiya-cho Building,
4-3-9 Toranomon, Minato-ku
Tokyo, Japan

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:

Site Counsel
IBM Corporation
P.O. Box 12195
3039 Cornwallis Road
Research Triangle Park, NC
27709-2195
USA

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 document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement.

This document is not intended for production use and is furnished as is without any warranty of any kind, and all warranties are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

© Copyright IBM Corp. 1994, 1996

Trademarks

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

Application System/400
APPN
AS/400
CICS
IBM
Operating System/2
OS/2
System/370
S/370
Systems Application Architecture
SAA

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

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

C-bus is a trademark of Corollary, Inc.

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

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

About This Book

This book is divided into three parts.

Part 1, “APPC API,” describes how to develop programs that use the Personal Communications advanced program-to-program communications (APPC) interface. APPC refers to an implementation of Systems Network Architecture (SNA) for logical unit (LU) type 6.2. Throughout this book, unless otherwised noted, APPC represents the Personal Communications implementation of APPC.

APPC provides a distributed transaction processing capability in which two or more programs cooperate to carry out some processing function. This capability involves communication between the programs so they can share resources, such as processor cycles, databases, work queues, and physical interfaces such as keyboards and displays.

Part 2, “Common Services API,” includes the verbs that make up the Common Services API.

Part 3, “LUA API,” describes how to develop programs that use the IBM conventional logical unit application (LUA) interface (in this book LUA also refers to request unit interface {RUI), that gives access to SNA LU types 0, 1, 2, and 3.

This book is for users of both Personal Communications Application System/400 (AS/400) V4.1 and Personal Communications AS/400 and 3270 V4.1. In this book, *Windows* refers to Windows 95.

Who Should Read This Book

This book is intended for programmers and developers who are writing either APPC or LUA applications.

This book assumes the reader is familiar with *SNA Transaction Programmer's Reference Manual for LU Type 6.2*.

How to Use This Book

Chapter 1, “Introducing APPC,” describes advanced program-to-program communications (APPC).

Chapter 2, “Fundamental APPC Concepts,” describes APPC transaction programs.

Chapter 3, “Using the Attach Manager,” describes how to use the attach manager.

Chapter 4, “Implementing APPC Transaction Programs,” describes the APPC extensions.

Chapter 5, “Entry Points,” describes the procedure entry points for the APPC API.

© Copyright IBM Corp. 1994, 1996

Chapter 6, “APPC Verbs,” describes the syntax of each verb. A copy of the structure that holds the information for each verb is included and each entry is described, followed by a list of possible return codes.

Chapter 7, “Procedure Entry Points and Common Services Verbs (CSV),” describes the common services extensions (CSVs).

Chapter 8, “Introducing the IBM Conventional LU Application,” describes the fundamental LUA programming concepts in this book.

Chapter 9, “Understanding LUA,” describes additional information about the structure and verbs for LUA.

Chapter 10, “Writing LUA Application Programs,” describes information about writing programs.

Chapter 11, “RUI Verbs,” describes details for each LUA verb.

Appendix A, “APPC Common Return Codes,” contains descriptions of the common return codes.

Appendix B, “APPC Conversation State Transitions,” describes the conversation states in which each APPC verb can be issued, and the state change that occurs on completion of the verb.

Conventions Used in This Book

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

Text Conventions

Bold	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.
Italics	Italic type indicates the following things: A variable that you supply a value for. 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. Book titles. A letter is being used as a letter or a word is being used as a word. **Example:** When you see an *a*, make sure it is not supposed to be an *an*.
Bold italics	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'.")

Double-Byte Character Set Support

Personal Communications supports double-byte character sets (DBCS), in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, the typing, displaying, and printing of DBCS characters require hardware and programs that support DBCS.

Where information applies specifically to DBCS, it is noted in this information unit.

ASCII refers to PC single-byte code in this book. ASCII should be considered as JISCII in Japan.

Where to Find More Information

Related Publications

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

IBM Systems Network Architecture: LU6.2 Reference: Peer Protocols, SC31-6808

IBM Systems Network Architecture: APPN Architecture Reference, SC30-3422

IBM Systems Network Architecture: Management Services, SC30-3346

IBM Systems Network Architecture: Formats, GA27-3136

IBM APPN Architecture and Product Implementations Tutorial, GG24-3669

IBM AS/400 Advanced Peer-to-Peer Networking, GG24-3287

IBM Communications Manager/2 System Management Programming Reference, SC31-6173

IBM Communications Manager/2 APPC Programming Guide and Reference, SC31-6160

IBM System/370 Principles of Operation, GA22-7000

IBM Systems Network Architecture: Technical Overview, GC30-3073

IBM Systems Network Architecture: VTAM Programming for LU Type 6.2, SC30-3400

IBM Systems Network Architecture Concepts and Products, GC30-3072

IBM Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2, SC30-3269

IBM Systems Network Architecture: Introduction to APPC, GG24-1584

IBM Systems Network Architecture: Transaction Programmer's Reference Manual for LU Type 6.2, GC30-3084

IBM Systems Network Architecture: Introduction to Sessions between Logical Units, GC20-1869

IBM Systems Network Architecture Format and Protocol Reference Manual: Architectural Logic, SC30-3112

Part 1. APPC API

Chapter 1. Introducing APPC . . . 3
SNA Communications Support . . . 3
SNA LU Type 6.2 Support . . . 4

Chapter 2. Fundamental APPC Concepts . . . 5
What Is a Transaction Program? . . . 5
APPC Transaction Programs . . . 5
CPI Communications Transaction Programs . . . 6
What Is a Logical Unit? . . . 6
LU Types . . . 6
Dependent and Independent LUs . . . 6
What Is a Session? . . . 7
What Is a Conversation? . . . 7

Chapter 3. Using the Attach Manager . . . 11
Identifying the Transaction Program Name on Both Machines . . . 12
DEFINE_TP . . . 13
Starting the Attach Manager . . . 15
Starting Programs with the Attach Manager . . . 16
Differentiating between an Application and a Transaction Program . . . 16
Matching Incoming Allocation Requests with RECEIVE_ALLOCATE Verbs . . . 17
Nonqueued Programs . . . 17
Queued Programs . . . 17
Transaction Program Definitions . . . 20
Defining Conversation Attributes . . . 20
Sync Level . . . 20
Conversation Type and Style . . . 21
Conversation Security for an Incoming Allocation Request . . . 21
Conversation Security for an Outgoing Allocation Request . . . 22
TPs and Local LUs . . . 22

Chapter 4. Implementing APPC Transaction Programs . . . 23
Writing Transaction Programs . . . 23
Option Sets Supported . . . 23
Queue-level Nonblocking . . . 24
Default Local LU Pool . . . 26

Chapter 5. Entry Points . . . 27
APPC . . . 27
WinAsyncAPPC() . . . 28
WinAsyncAPPCEx() . . . 30
WinAPPCCancelAsyncRequest() . . . 32
WinAPPCCancelBlockingCall() . . . 33
WinAPPCCleanup() . . . 34
WinAPPCIsBlocking() . . . 35
WinAPPCStartup() . . . 36
WinAPPCSetBlockingHook() . . . 37
WinAPPCUnhookBlockingHook() . . . 38
GetAppcConfig() . . . 39
GetAppcReturnCode() . . . 40

© Copyright IBM Corp. 1994, 1996

Chapter 6. APPC Verbs 43
Verb Control Blocks 43
Common Fields 43
APPC API Support 43
Verbs Supported 43
Indication Verb Control Blocks 45
INITIALIZE_SESSION_LIMIT 45
RESET_SESSION_LIMIT 48
Type Independent Verb Control Blocks 51
GET_TP_PROPERTIES 51
GET_TYPE 54
RECEIVE_ALLOCATE 56
TP_ENDED 58
TP_STARTED 60
Basic Conversation Verb Control Blocks 62
ALLOCATE 62
CONFIRM 66
CONFIRMED 68
DEALLOCATE 70
FLUSH 73
GET_ATTRIBUTES 75
PREPARE_TO_RECEIVE 78
RECEIVE_AND_POST 80
RECEIVE_AND_WAIT 84
RECEIVE_IMMEDIATE 87
REQUEST_TO_SEND 90
SEND_CONVERSATION 92
SEND_DATA 95
SEND_ERROR 98
TEST_RTS 101
TEST_RTS_AND_POST 103
Mapped Conversation Verb Control Blocks 105
MC_ALLOCATE 105
MC_CONFIRM 109
MC_CONFIRMED 111
MC_DEALLOCATE 113
MC_FLUSH 115
MC_GET_ATTRIBUTES 117
MC_PREPARE_TO_RECEIVE 120
MC_RECEIVE_AND_POST 122
MC_RECEIVE_AND_WAIT 125
MC_RECEIVE_IMMEDIATE 128
MC_REQUEST_TO_SEND 131
MC_SEND_DATA 136
MC_SEND_ERROR 139
MC_TEST_RTS 141
MC_TEST_RTS_AND_POST 143

Chapter 1. Introducing APPC

Personal Communications provides Advanced Peer-to-Peer Networking (APPN) end node support for workstations, enabling them to communicate more flexibly with other systems in the network.

Personal Communications provides advanced program-to-program communications (APPC) to support communications between distributed processing programs, called *transaction programs* (TPs). APPN extends this capability to a networking environment. The transaction programs can be located at any node in the network that provides APPC.

Personal Communications improves APPC throughput in local area network (LAN) environments. Personal Communications supports the following protocols for APPC:

IBM Token-Ring Network
Synchronous Data Link Control (SDLC)
Twinaxial
Ethernet

Figure 1 illustrates the functional structure of the Personal Communications implementation of APPC.

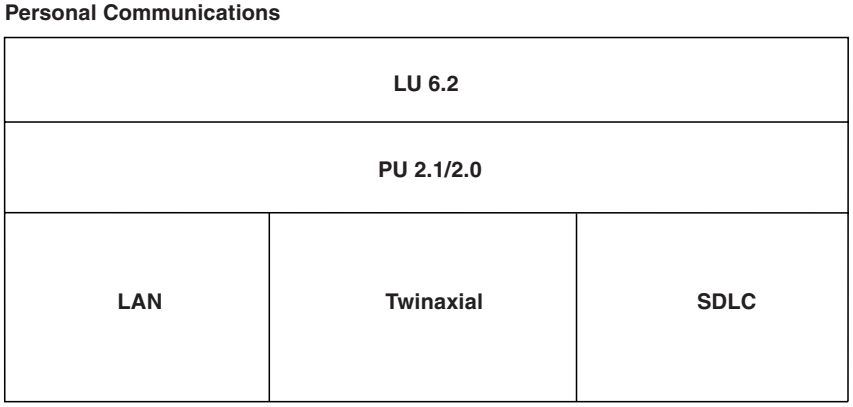

Figure 1. Personal Communications APPC Implementation

SNA Communications Support

Personal Communications supports Systems Network Architecture (SNA) type 2.1 nodes (including SNA type 2.0 and SNA type 2.1 support for logical units [LUs] other than SNA LU 6.2). This support lets you write programs to communicate with many other IBM SNA products.

You can write programs without knowing the details of the underlying network. All you need to know is the name of the partner LU; you do not need to know its location. SNA determines the partner LU location and the best path for routing data. A change to the underlying network, the addition of a new adapter, or the relocation of a machine, does not affect APPC programs. A program might, however, need to establish link connections over switched SDLC connections.

© Copyright IBM Corp. 1994, 1996

When Personal Communications starts, it establishes local LU and logical link definitions, which are stored in a configuration file. The system management application programming interface (API) provides functions that control configuration definition and adapter and link activation. Refer to *Personal Communications System Management Programming* for information about these functions. Users can use the configuration and node operations functions while Personal Communications runs. Refer to *Personal Communications Up and Running* and *Personal Communications System Management Programming* for more information about configuration and node operations.

SNA LU Type 6.2 Support

LU 6.2 is an architecture for program-to-program communications. Personal Communications supports all base LU 6.2 functions. Some of the optional SNA LU 6.2 functions are:

- Basic and mapped conversations
- Half-duplex conversation styles
- Synchronization level of confirm
- Security support at session and conversation levels
- Multiple LUs
- Parallel sessions, including the ability to use a remote system to change the number of sessions
- Concurrent multiple links, each of which can be connected to a subarea network (one per PU definition)

Chapter 2. Fundamental APPC Concepts

This book describes the APPC API supported by Personal Communications. Its purpose is to provide:

A brief overview of the structure of the APPC API
Reference information about the specific syntax of the verbs that flow across the interface

What Is a Transaction Program?

A transaction program is a block of code, or part of an application program, that uses APPC communications functions. Application programs use these functions to communicate with application programs on other systems that support APPC. A transaction program has a 64-byte name (**tp_name**).

Personal Communications provides the APPC API and supports the IBM Systems Application Architecture* (SAA*) Common Programming Interface for Communications (CPI Communications) calls for transaction programs.

Transaction programs issue APPC verbs to invoke APPC functions. See Chapter 4, “Implementing APPC Transaction Programs,” for details about how transaction programs issue APPC verbs. Transaction programs can issue CPI Communications calls to invoke CPI Communications functions. The CPI Communications calls let application programs take advantage of the consistency that SAA provides. See “CPI Communications Transaction Programs” on page 6 for information about the CPI Communications calls.

APPC Transaction Programs

An APPC transaction program is not an application; it is a section of an application. A single application can contain many transaction programs. Every transaction program has a unique 8-byte identification number (**tp_id**).

APPC supplies verbs that start and stop transaction programs within applications. APPC also supplies a full set of conversation verbs that you can use to implement the function of your transaction program.

A transaction program issues a request to APPC, in the form of a verb, to perform some action for an application program. A verb is a formatted request that a transaction program issues and APPC executes. A program uses APPC verb sequences to communicate with another program. Two programs that communicate with each other can be located at different systems or on the same system. The APPC API is the same in both cases.

You can consider the set of APPC verbs described in this book as a programming language in which you write conversations. APPC verbs are coded as control blocks; each verb's syntax is precisely defined. Your programs pass verb control blocks to the APPC API to gain access to APPC facilities.

When a transaction program exchanges data with another transaction program, they are called *partner* transaction programs.

© Copyright IBM Corp. 1994, 1996

CPI Communications Transaction Programs

A CPI Communications transaction program is similar to an APPC transaction program; both types of transaction programs use APPC support. Rather than issuing verbs, however, a CPI Communications transaction program invokes each CPI Communications function with a call to the function that passes the appropriate parameters on the call.

Most CPI Communications calls correspond to APPC verbs. For example, the calls that allocate outbound conversations and accept (receive) conversations, and the calls that send and receive data on the conversation, provide functions that are similar to those of the corresponding APPC verbs. The exceptions are the calls that initialize a conversation before allocating the conversation and the calls that set and extract individual conversation characteristics.

Refer to *CPI Communications Reference* for details about the support that Communications Server provides for CPI Communications programs.

What Is a Logical Unit?

Every transaction program gains access to an SNA network through a *logical unit* (LU). An LU is SNA software that accepts verbs from your programs and acts on those verbs. A transaction program issues APPC verbs to its LU. These verbs cause commands and data to flow across the network to a partner LU. An LU also acts as an intermediary between the transaction programs and the network to manage the exchange of data between transaction programs. A single LU can provide services for multiple transaction programs. Multiple LUs can be active simultaneously.

LU Types

Communications Manager supports LU types 0, 1, 2, 3, and 6.2. LU types 0, 1, 2, and 3 support communication between host application programs and different kinds of devices, such as terminals and printers. Refer to Part 3, “LUA API,” for details on writing these programs.

LU 6.2 supports communications between two programs located at type 5 subarea nodes, type 2.1 peripheral nodes, or both, and between programs and devices. APPC is an implementation of the LU 6.2 architecture, which is described in this part of the book.

Communication occurs only between LUs of the same LU type. For example, an LU 2 communicates with another LU 2; it does not communicate with an LU 3.

When Personal Communications is configured, your local and partner LUs are defined once for each machine. When you design programs, you must understand the capabilities supported by the LU in the particular machines you plan to use.

Dependent and Independent LUs

A *dependent LU* depends on a system services control point (SSCP) to activate a session. A dependent LU needs an active SSCP-LU session, which the dependent LU uses to start an LU-LU session with an LU in a subarea node. A dependent LU can have only one session at a time with the subarea LU. For communications with a transaction program at a subarea node, each dependent LU can have only

one conversation at a time, and each dependent LU can support communications for only one transaction program at a time.

An *independent LU* does not depend on an SSCP to activate a session. An independent LU supports multiple concurrent sessions with other LUs in a subarea node, so you can have multiple conversations and support multiple transaction programs for communications with subarea transaction programs. LUs between peripheral nodes also use this support.

The distinction between a dependent LU and an independent LU is meaningful only when discussing a session between an LU in a peripheral node and an LU in a subarea node. Otherwise, dependent and independent LUs both support multiple concurrent sessions and conversations when communicating between type 2.1 peripheral nodes (for example, between two workstations). A Personal Communications LU can support a single session with a dependent LU or multiple sessions with an independent LU.

What Is a Session?

Before transaction programs can communicate with each other, their LUs must be connected in a mutual relationship called a *session*. A session connects two LUs, so it is called an *LU-LU* session. Figure 2 illustrates this communication relationship. Multiple concurrent sessions between the same two LUs are called *parallel* LU-LU sessions.

Sessions act as conduits that manage the movement of data between a pair of LUs in an SNA network. Specifically, sessions deal with things such as the quantity of data transmitted, data security, network routing, and traffic congestion.

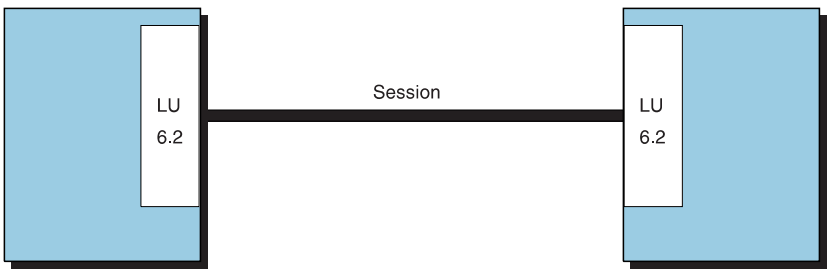

Figure 2. A Session between Two LUs

Sessions are maintained by their LUs. Normally, your transaction programs do not deal with session characteristics. You define session characteristics when you:

Configure your system
Use the management verbs

What Is a Conversation?

The communication between transaction programs is called a *conversation*. Conversations occur across LU-LU sessions. A conversation starts when a transaction program issues an APPC verb or CPI Communications call that allocates a conversation. The conversation style associated with the conversation indicates the style of data transfer to be used, two-way alternate or two-way

simultaneous. A conversation that specifies a two-way alternate style of data transfer is also known as a *half-duplex* conversation.

When a half-duplex conversation is allocated to a session, a send-receive relationship is established between the transaction programs connected to the conversation, and a two-way alternate data transfer occurs where information is transferred in both directions, one direction at a time. Like a telephone conversation, one transaction program calls the other, and they "converse," one transaction program talking at a time, until a transaction program ends the conversation. One transaction program issues verbs to send data, and the other transaction program issues verbs to receive data. When it finishes sending data, the sending transaction program can transfer send control of the conversation to the receiving transaction program. One transaction program decides when to end the conversation and informs the other when it has ended.

Conversations can exchange control information and data. The transaction program should select the conversation style best-suited for its application.

Figure 3 shows a conversation between two transaction programs as it occurs over a session.

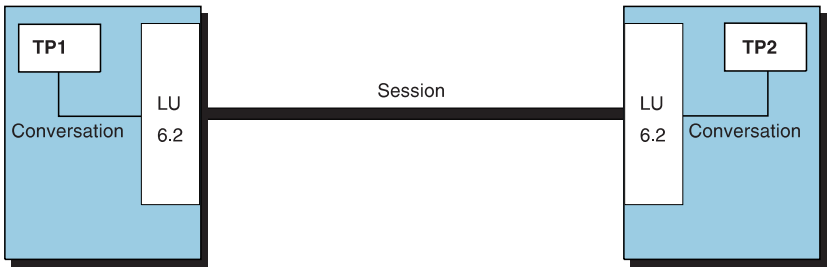

Figure 3. A Conversation between Two Transaction Programs

A session can support only one conversation at a time, but one session can support many conversations in sequence. Because multiple conversations can reuse sessions, a session is a long-lived connection compared to a conversation. When a program allocates a conversation and all applicable sessions are in use, the LU puts the incoming Attach (allocation request) on a queue. It completes the allocation when a session becomes available. See Chapter 3, "Using the Attach Manager," for more information about the attach manager.

Two LUs can also establish parallel sessions with each other to support multiple concurrent conversations. A parallel session occurs when either transaction program allocates a conversation, and a session exists but is being used by a conversation. The LU can request a new session to satisfy the allocation.

Figure 4 on page 9 shows three parallel sessions between two LUs; each session carries a conversation.

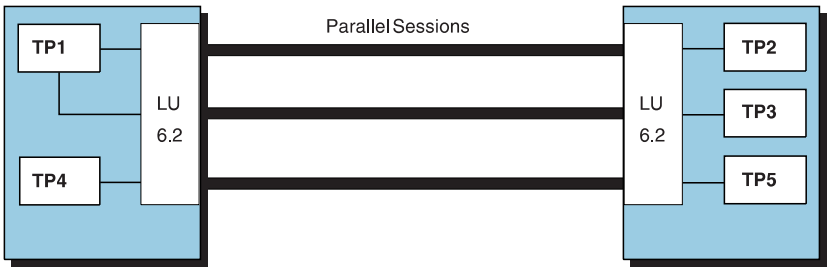

Figure 4. Parallel Sessions between LUs

Chapter 3. Using the Attach Manager

An important LU 6.2 feature is the ability of a program in one node to start corresponding programs in other nodes. The *attach manager* handles incoming requests to start programs.

This chapter considers programs in your (local) workstation that start at the request of partner programs. The local program is referred to as *remotely started*. Workstation users and administrators want to control which programs can be remotely started for security and resource control. Users at remote nodes should not start programs that destroy data or use the local workstation's memory at critical times. The attach manager acts as a gatekeeper, handling incoming requests to start programs on the local workstation.

The attach manager takes its name from an SNA message, called an *Attach*, that flows between a pair of LUs. An Attach flows when a program that uses the partner LU initiates a conversation. The LU 6.2 component in the local workstation passes any Attach it receives to its attach manager for handling. A received Attach is called an *incoming allocation request* or *incoming Attach*. In this chapter, the phrase *incoming allocation request* means that the SNA Attach is generated by a partner LU.

The attach manager does the following things:

- Enables remote nodes to start applications in the local workstation. Multiple instances of a program can be started in series (queued) or in parallel (nonqueued).
- Passes parameters to remotely started programs.
- Starts programs in windows or in the background.
- Uses security guidelines to verify incoming allocation requests.
- Specifies timeout values for holding incoming allocation requests and locally issued APPC **RECEIVE_ALLOCATE** verbs or CPI Communications Accept_Conversation or Accept_Incoming (CMACCP, CMACCI) calls.
- Checks the conversation type (that is, basic or mapped) and synchronization level of incoming allocation requests.

Figure 5 illustrates the attach manager function.

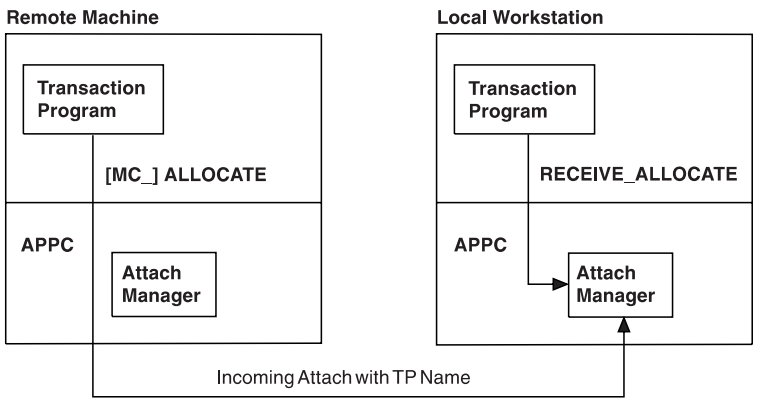

Figure 5. Attach Manager Function in APPC

© Copyright IBM Corp. 1994, 1996

In a communicating pair of transaction programs, only the node that receives allocation requests needs the attach manager. The attach manager manages three kinds of input:

Incoming allocation requests (Attaches) from partner transaction programs

APPC **RECEIVE_ALLOCATE** verbs or CPI Communications CMACCP and CMACCI calls from local programs

Configuration definitions for transaction programs, user IDs, and passwords

The *TP name* is a key piece of information in an incoming allocation request. The attach manager uses the transaction program name to decide which program to start in the local workstation. Programmers and administrators at both nodes need to agree on each transaction program name. A program that issues an allocation request supplies a transaction program name as a parameter to the APPC **[MC_]ALLOCATE** or **[MC_]SEND_CONVERSATION** verb.

When an Attach is received, the transaction program name in the Attach is matched against transaction program names from the **DEFINE_TPs**. If a match is found, the executable name from that definition is started. If a match is not found, then the name of the executable is assumed to be the same as that which is specified in the Attach appended with **.EXE**.

If a transaction program issues a **RECEIVE_ALLOCATE** call and specifies a transaction program name that has not previously been defined, the system performs an implicit definition of the transaction program and assigns default values to the parameters.

The defaults used are:

Attach timeout	= 0	(no timeout is applied)
Receive Allocate timeout	= 0	(no timeout is applied)
Attach Manager dynamically-loaded	= Yes	(the transaction program can be loaded by the attach manager)

These defaults mean that if you issue a call to **RECEIVE_ALLOCATE** as described above, it will not complete until an attempt is made to attach to the named transaction program, or you can cancel the call.

Identifying the Transaction Program Name on Both Machines

If the program identified by the attach manager cannot be started, the attach manager rejects the allocation request; the program that issued an allocation request is notified that the attach manager could not start the program.

Users or administrators define transaction programs during Personal Communications configuration to build the list of defined transaction program names. Each unique transaction program name to be accepted from a partner requires a transaction program definition in the local (accepting) workstation unless you are willing to accept the default. The transaction program definition contains information about the transaction program. Similarly, during configuration, a list of security information (allowable passwords and user IDs) is built from the LU 6.2 conversation security information. Refer to *Personal Communications Up and Running* for Personal Communications configuration information. The following

describes the configuration data that must be specified to define a transaction program.

DEFINE_TP

VCB Structure

```
typedef struct define_tp
{
 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    tp_name[64];           /  TP name                             /
 TP_CHARS         tp_chars;              /  TP characteristics                  /
} DEFINE_TP;

typedef struct tp_chars
{
 unsigned char    description[RD_LEN]
                                         /  resource description               /
 unsigned char    conv_type;             /  conversation type                   /
 unsigned char    security_rqd;          /  security support                    /
 unsigned char    sync_level;            /  synchronization level support      /
 unsigned char    dynamic_load;          /  dynamic load                        /
 unsigned char    enabled;               /  is the TP enabled?                  /
 unsigned char    pip_allowed;           /  program initialization              /
                                         /  parameters supported               /
 unsigned char    reserv3[1 ];           /  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 reserved[6];              /  max size is 12  bytes              /
} TP_SPEC_DATA;
```

Supplied Parameters

The application supplies the following parameters:

opcode AP_DEFINE_TP

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 TP being defined. Note that Personal Communications 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 type(s) of conversation supported by this TP.

AP_BASIC
AP_MAPPED
AP_EITHER

tp_chars.security_rqd
Specifies whether conversation security information is required to start the TP (AP_NONE, AP_SAME or AP_PGM).

tp_chars.sync_level
Specifies the synchronization levels supported by the TP.

AP_NONE
The TP supports a synchronization level of None.

AP_CONFIRM_SYNC_LEVEL
The TP supports a synchronization level of Confirm.

AP_EITHER
The TP supports a synchronization level of None or Confirm.

tp_chars.dynamic_load
Specifies whether the TP can be dynamically loaded (AP_YES or AP_NO).

tp_chars.enabled
Specifies whether the TP can be attached successfully (AP_YES or AP_NO).

tp_chars.pip_allowed
Specifies whether the TP can receive program initialization (PIP) parameters (AP_YES or AP_NO).

tp_chars.tp_instance_limit
Limit on the number of concurrently active TP 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 TP data.

tp_spec_data
Provides information to attach manager on how to load the program.

tp_chars.tp_data.pathname
Specifies the path and TP name.

tp_chars.tp_data.parameters
Specifies the parameters for the TP.

tp_chars.tp_data.queued
Specifies whether a TP is queued.

tp_chars.tp_data.load_type
Specifies how the program should be loaded.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

Returned Parameters: Node Not Started

If the verb does not execute because the node has not yet been started, Personal Communications returns the following parameters:

primary_rc AP_NODE_NOT_STARTED

Returned Parameters: Node Stopping

If the verb does not execute because a STOP_NODE verb has been issued, Personal Communications returns the following parameters:

primary_rc AP_NODE_STOPPING

Returned Parameters: System Error

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

primary_rc AP_UNEXPECTED_SYSTEM_ERROR

Authorized users can, at any time, use the node configuration definition utility to update the list of transaction program, user ID, or password definitions. Applications can also issue their own **DEFINE_TP** and **DEFINE_USER_ID_PASSWORD** verbs while LU 6.2 is active.

Starting the Attach Manager

Users can start and stop the attach manager while the SNA node is active. Each time the attach manager starts, it begins to process incoming Attaches. When the attach manager stops, it purges any queued Attaches. Refer to the *Personal Communications System Management Programming* for the applicable verbs.

The attach manager needs to be started only in nodes that run remotely started transaction programs. The attach manager does not need to be started in a node if all transaction programs in the node initiate conversations (that is, they all issue APPC **[MC_]ALLOCATE** or **[MC_]SEND_CONVERSATION** verbs). Personal Communications node operations facility enables authorized users to start or stop the attach manager at any time. Authorized programs issue the Enable Attach Manager and Disable Attach Manager node operations verbs to start or stop the attach manager.

Starting Programs with the Attach Manager

When the attach manager starts a program on a workstation, it uses the **load_type** field in the defined transaction program list to decide how to run the program. A remotely started program can be configured to start in one of the following ways:

Console An application that displays a user-visible window or runs as a full DOS application.

Background
The program starts in a background (detached) process. A background process should not issue any input or output calls to the keyboard, the mouse, or the display. If your program is completely debugged and requires no interactive user input, this option provides the fastest performance.

If the attach manager cannot start the program (for example, Personal Communications cannot provide sufficient memory), the attach manager rejects the incoming allocation request.

Differentiating between an Application and a Transaction Program

The term transaction program has a special meaning in APPC. A transaction program is not an application; it is a section of an application.

A transaction program starts either when an application successfully issues an APPC **RECEIVE_ALLOCATE** or **TP_STARTED** verb. Both of these methods identify the transaction program as a new transaction program that APPC needs to know about. APPC reserves a group of memory blocks for the transaction program and creates a unique transaction program identifier, **tp_id**, which it returns to the calling program.

An application must supply a **tp_id** on all conversation verbs that it issues while it is a part of that transaction program. When an application issues a **TP_ENDED** verb, APPC clears its buffers for that transaction program and marks the **tp_id** as not valid. When an application terminates, APPC ends any active transaction programs associated with that process. When the attach manager receives an allocation request and ensures it is valid, and if a **RECEIVE_ALLOCATE** is not pending, it starts the application that corresponds to the incoming transaction program name. Notice that it starts a program, not a transaction program. Generally, the application then issues a verb that establishes it as a transaction program. Given mutual consent between the sending node and the local workstation, you can configure the attach manager to start *any* application in the local workstation.

Similarly, conversations have their own identifiers, and a transaction program must be established before a conversation can be allocated. Many conversations can use a single **tp_id** concurrently (such as in multiple threads) or sequentially (where one conversation follows another). When a transaction program ends, APPC deallocates any active conversations.

Matching Incoming Allocation Requests with RECEIVE_ALLOCATE Verbs

A remotely started program in a local workstation normally issues an APPC **RECEIVE_ALLOCATE** verb to start both a transaction program and a conversation. The APPC **RECEIVE_ALLOCATE** verb specifies the same transaction program name that the remote transaction program specified in its APPC **[MC_]ALLOCATE** or **[MC_]SEND_CONVERSATION** verb. APPC passes the **RECEIVE_ALLOCATE** verb to the attach manager for processing. When the attach manager sees a **RECEIVE_ALLOCATE** verb that matches a received Attach (and the attach manager performs several cross-checks), it signals APPC that a conversation can begin. At this point, the attach manager ends its involvement in the conversation.

During transaction program configuration, you have two choices for handling multiple incoming allocation requests for the same program. You can run multiple instances of the same program concurrently in the local workstation (*nonqueued* operation), or you can run one instance of the same program at a time (*queued* operation). These values are configured in the **queued** and **dynamic load** parameters, that have the following options:

Nonqueued—attach manager started
Queued—attach manager started
Operator started

Nonqueued Programs

When a program is configured as nonqueued, each incoming allocation request causes the attach manager to load and execute another instance of the program associated with the incoming transaction program name.

The attach manager holds valid incoming allocation requests indefinitely, waiting for a matching **RECEIVE_ALLOCATE** verb from the program it started. If that program fails to issue a **RECEIVE_ALLOCATE** verb (for example, it loops in the code that precedes the **RECEIVE_ALLOCATE** verb), the attach manager holds the allocation request until the process terminates.

Queued Programs

Queued programs can start in one of two ways:

Attach manager started
The program is started by the attach manager.

Operator started
The program is to be started by another program in the workstation or by an operator.

The attach manager maintains two queues for each queued transaction program name in the defined transaction program list. One queue is for incoming allocation requests; the other is for **RECEIVE_ALLOCATE** verbs. For example, when an incoming allocation request arrives, the attach manager starts the corresponding local program or sends a message to the operator. The node holds the incoming allocation request until the program that the attach manager started issues a matching **RECEIVE_ALLOCATE** verb or until a timeout occurs. The node uses the value configured for the **incoming_alloc_timeout** parameter to determine when time-outs occur. Other allocation requests can arrive for that transaction program

or for another transaction program. The other programs wait in their respective queues until a matching **RECEIVE_ALLOCATE** verb is issued, or until they time out.

Local programs can issue **RECEIVE_ALLOCATE** verbs before any matching allocation request arrives. The attach manager holds the **RECEIVE_ALLOCATE** verb on its respective queue and waits for an allocation request to arrive from a partner LU. Each queue has a timeout value; the **rcv_alloc_timeout** parameter specifies how long a **RECEIVE_ALLOCATE** verb can wait on a queue before the verb times out. The attach manager returns queued **RECEIVE_ALLOCATE** verbs to the associated programs with an ALLOCATE_NOT_PENDING return code. The timeout value for **RECEIVE_ALLOCATE** verbs can be 0 to enable programs to check whether any allocation requests are queued, and, if not, to continue other processing.

The **RECEIVE_ALLOCATE** verb can be issued as a nonblocking verb. This enables the transaction program to service multiple conversations from a single thread in a single process.

When **RECEIVE_ALLOCATE** is issued as a nonblocking verb, the attach manager returns control to the transaction program immediately; the transaction program need not remain in a wait condition while waiting for the matching incoming allocation request to arrive. Instead, the transaction program can perform other work, and choose when to wait for the matching incoming allocation request.

The transaction program can queue multiple nonblocking **RECEIVE_ALLOCATE** verbs for different conversations. The maximum number of verbs that can be queued is limited only by resource constraints. A nonblocking **RECEIVE_ALLOCATE** verb will remain on the attach manager's **RECEIVE_ALLOCATE** verb queue until either the matching allocation request arrives or the verb times out, that is, the **rcv_alloc_timeout** value has been reached.

The attach manager saves the information that identifies the transaction program when a queued program issues a valid **RECEIVE_ALLOCATE** verb call for a transaction program. When the queued program ends, the attach manager examines the queue of allocation requests for that transaction program. If the queue is not empty, the attach manager starts a new instance of the program, or sends a message that directs the operator to start the program.

You should configure the maximum size of the incoming allocation request queue for each transaction program. Resource constraints limit the number of queued **RECEIVE_ALLOCATE** verbs.

The following two cases summarize queued operations:

Case 1: One or more incoming allocation requests arrive before a **RECEIVE_ALLOCATE** verb or CPI Communications CMACCP call is issued for a given transaction program. The attach manager queues the incoming allocation requests (for a time specified by a configured timeout value) until a **RECEIVE_ALLOCATE** verb is issued. The first incoming allocation request satisfies the **RECEIVE_ALLOCATE** verb.

Case 2: A **RECEIVE_ALLOCATE** verb is issued before an incoming allocation request arrives for a given transaction program. The attach manager queues the **RECEIVE_ALLOCATE** verb (for a time specified by a configured timeout value) until an incoming allocation request arrives. In certain cases, more than one **RECEIVE_ALLOCATE** verb might be issued and queued before an incoming allocation request arrives. Each new incoming allocation request satisfies the next **RECEIVE_ALLOCATE** verb in the queue.

Table 1 on page 19 provides a summary of the verbs and incoming allocation requests associated with **queued** and **dynamic load** parameter values.

Table 1. Verb Processing and Transaction Program Name Configuration

	Transaction Program Operation		
Verb Processing	**Nonqueued—attach manager started**	**Operator started**	**Queued—attach manager started**
Incoming allocation request with pending **RECEIVE_ALLOCATE** verb.	Cannot occur; no queue of pending **RECEIVE_ALLOCATE** verbs.	OK **RECEIVE_ALLOCATE** verb.	OK **RECEIVE_ALLOCATE** verb.
Incoming allocation request without pending **RECEIVE_ALLOCATE** verb.	Load and execute another program instance. Hold incoming allocation request. Wait for **RECEIVE_ALLOCATE** verb.	Put incoming allocation request on queue unless queue is full. Wait for **RECEIVE_ALLOCATE** verb or for allotted time to expire.	If program is not started, load and execute it. Put incoming allocation request on queue unless queue is full. Wait for **RECEIVE_ALLOCATE** verb or for allotted time to expire.
RECEIVE_ALLOCATE verb with incoming allocation request pending.	OK **RECEIVE_ALLOCATE** verb.	OK **RECEIVE_ALLOCATE** verb.	OK **RECEIVE_ALLOCATE** verb.
RECEIVE_ALLOCATE verb with no pending incoming allocation request.	Cannot occur; pending allocation requests for nonqueued operations cannot run out of time.	Hold **RECEIVE_ALLOCATE** verb. Wait for incoming allocation request or for allotted time to expire.	Hold **RECEIVE_ALLOCATE** verb. Wait for incoming allocation request or for the allotted time to expire.
Transaction program operation ends.	Nothing happens.	Nothing happens.	If there is a pending allocation request, reload the program; otherwise, reload on the next incoming allocation request.

Transaction Program Definitions

Personal Communications configuration uses two naming levels to identify the remotely started program:

The 64-character name of the local program known by the partner transaction program (**tp_name**)

The file specification of the local program to be started (filespec)

The use of two names enables flexible reconfiguration that can increase the portability of your APPC programs among workstations.

TP name The name that a partner transaction program sends in the allocation request to the attach manager in the local workstation.

The partner transaction program and the local program must both know the transaction program name. The transaction program name is a supplied parameter for **RECEIVE_ALLOCATE** verbs in programs on local LUs. Partner transaction programs supply a transaction program name with APPC **[MC_]ALLOCATE** or **[MC_]SEND_CONVERSATION** verbs.

Path name

The transaction program file specification (path name) names the program to start locally. The transaction program file specification contains the executable file's drive, path, file name, and extension.

Multiple transaction program definitions can specify the same transaction program file specification. The attach manager must determine whether to run one or multiple instances of a program, so a given transaction program file specification must be configured as either queued or nonqueued in *all* definitions that name it. For example, if a definition that specifies **MYTP.EXE** is configured as "queued—attach manager started," **MYTP.EXE** cannot be configured as nonqueued in another transaction program definition. However, the transaction program filespec is case sensitive.

Defining Conversation Attributes

The conversation parameters **sync_level**, **conv_type**, and **security_rqd** do not directly influence how the attach manager starts a program. However, the attach manager uses the parameters to determine whether to reject an incoming allocation request before queuing it, or checking for corresponding **RECEIVE_ALLOCATE** verbs.

Sync Level

Specify whether your transaction program will support the verbs and parameters for confirmation processing when you define **sync_level**. These APPC verbs are **[MC_]CONFIRM** and **[MC_]CONFIRMED**. Certain parameters on the **[MC_]ALLOCATE**, **[MC_]SEND_CONVERSATION**, **[MC_]PREPARE_TO_RECEIVE**, and **[MC_]DEALLOCATE** are for confirmation processing. For Common Programming Interface Communications (CPIC) users, **sync_level** can be set by the **Set_Sync_Level** (CMSSL) call.

Incoming allocation requests contain a field that indicates whether a partner transaction program issues verbs or parameters for confirmation processing. The

attach manager checks the field on the incoming allocation request against the configured value in its list of transaction program definitions. If the values do not match, attach manager rejects the incoming allocation request. The possible configuration choices are:

NONE The transaction program does not issue any verb that relates to confirmation processing, in any of its conversations.

CONFIRM
The transaction program can perform confirmation processing on its conversations. The transaction program can issue verbs and recognize returned values that relate to confirmation. If the transaction program contains any of the verbs for confirmation processing, define **sync_level**(CONFIRM) to guarantee a compatible session.

EITHER The transaction program can participate in conversations with partners that do or do not specify confirmation processing. Do not pick EITHER if the transaction program being configured requires confirmation processing.

Conversation Type and Style

The **conv_type** parameter is used to determine both the conversation type and conversation style of the program to be started. The conversation type attribute determines whether the program to be started supports basic or mapped records when it sends and receives data. The conversation style attribute determines whether the program to be started supports half-duplex conversations. The attach manager checks whether a transaction program uses basic or mapped verbs and if it uses half-duplex. The conversation types are:

BASIC The transaction program issues only basic conversation verbs for its half-duplex conversations.

MAPPED The transaction program issues only mapped conversation verbs for its half-duplex conversations.

EITHER The transaction program issues either basic or mapped conversation verbs for a half-duplex conversation, depending on what arrives on the incoming allocation request.

Conversation Security for an Incoming Allocation Request

A transaction program definition can specify that incoming allocation requests must supply a password and user ID. The password and user ID are optional parameters on the **[MC_]ALLOCATE** and **[MC_]SEND_CONVERSATION** verbs or the CPIC calls Set_Conversation_Security_UserID (CMSCSU) and Set_Conversation_Security_PassWord (CMSCSP). If a local transaction program definition specifies conversation security, the attach manager validates the password and user ID of incoming allocation requests. The attach manager rejects the allocation request if a user ID and password are not present, or if they do not match a valid combination of passwords and user IDs.

The attach manager checks the validity of any incoming allocation requests that arrive with a password and user ID, even if the transaction program definition specifies that conversation security is not required. The allocation request is rejected if the password and user ID do not match a valid combination in the list. Thus, if a password or user ID arrives in an allocation request, it is never ignored.

Conversation Security for an Outgoing Allocation Request

A remotely started transaction program (one started by another transaction program) can validate a user ID and password before it allocates a conversation to a third transaction program. In such a case, the **security**(SAME) parameter in the **[MC_]ALLOCATE** and **[MC_]SEND_CONVERSATION** verbs can indicate that the conversation security is already verified. The second Attach automatically gets the user ID from the first Attach, which started the first conversation.

APPC can obtain the current user ID and send it, with an indicator that the user ID was validated. In the Attach for a locally started transaction program that uses the **security**(SAME) parameter in either the **[MC_]ALLOCATE** or the **[MC_]SEND_CONVERSATION** verb, the partner must be able to accept the already validated indication.

See *Personal Communications System Management Programming* for more information about using the user ID and password.

TPs and Local LUs

All transaction programs will be associated with the default local LU, which is the control point.

Chapter 4. Implementing APPC Transaction Programs

This chapter describes the implementation of APPC Transaction Programs using the dynamic link library (DLL) file provided.

Writing Transaction Programs

Personal Communication provides a dynamic link library (DLL) file, **WAPPC32.DLL**, that handles APPC verbs.

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

APPC verbs have a straightforward language interface. Your program fills in fields in a block of memory called a *verb control block*. Then it calls **WAPPC32.DLL** and passes a pointer to the verb control block. When its operation is complete, APPC 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.

Option Sets Supported

Personal Communications supports the following APPC option sets. See *SNA Transaction Programmer's Reference* for LU Type 6.2 for a fuller description of each option set.

102 Get attributes.

103 Post on receipt with test for posting (through the **RECEIVE_AND_POST** verb).

104 Post on receipt with wait (through the **RECEIVE_AND_POST** verb).

105 Prepare to receive.

106 Receive immediate.

109 Get transaction program name and instance identifier.

110 Get conversation type.

113 Nonblocking support.

201 Queued allocation of a contention-winner session.

203 Immediate allocation of a session.

204 Conversations between programs located at the same LU.

205 Queued allocation or when session is free

211 Session level LU-LU verification.

212 User ID verification.

213 Program-supplied user ID and password.

214 User ID authorization.

241 Send PIP data.

242 Receive PIP data

© Copyright IBM Corp. 1994, 1996

243 Accounting.

244 Long locks.

245 Test for request-to-send received.

247 User control data.

251 Extract translation and conversation correlator.

290 Logging of data in a system log.

291 Mapped conversation LU services component.

401 Reliable one-way brackets.

501 **CHANGE_SESSION_LIMIT** verb.

502 **ACTIVATE_SESSION** verb.

504 **DEACTIVATE_SESSION** verb.

505 **LU**-*definition* verb.

601 **MIN_CONWINNERS_TARGET** parameter.

602 **RESPONSIBLE(TARGET)** parameter.

603 **DRAIN_TARGET(NO)** parameter.

604 **FORCE** parameter.

605 LU-LU session limit.

606 Locally known LU names.

607 Uninterpreted LU names.

610 Maximum RU size bounds.

612 Contention winner automatic activation limit.

613 Local maximum (LU,mode) session limit.

616 CPSVCMG mode name support.

Queue-level Nonblocking

The Personal Communications APPC API supports queue-level nonblocking. This support is provided through the asynchronous interfaces (**WinAsyncAPPC()** and **WinAsyncAPPCEx**).

Nonblocking operation enables control to be returned to the application if processing of a verb cannot be completed immediately, so that the application can continue other processing until it is notified that the outstanding verb has completed. Queue-level nonblocking means that the application can issue nonblocking verbs for different queues and have the verbs processed simultaneously by Personal Communications. The application can also issue a succession of nonblocking verbs for a given queue without waiting for any of the verbs to complete.

Personal Communications maintains three queues for nonblocking verbs:

An allocate queue (one for each active transaction program)
A send/receive queue (one per conversation)
A send-expedited queue (one per conversation)

All three queue types can hold an unlimited number of verbs. Nonblocking verbs are queued if another (blocking or nonblocking) verb is being processed by Personal Communications. Verbs in an allocate queue are processed concurrently, whereas verbs in the other queues are processed one at a time, in the order in which they are received by Personal Communications.

The application notifies Personal Communications that it wants a verb to be processed in nonblocking mode by setting a flag in the **opext** field, **AP_NON_BLOCKING**. The application can supply an event handle with any nonblocking verb that is used to notify the application of asynchronous verb completion. This handle is passed to Personal Communications in the **SECONDARY_RC** field. If no handle is specified, the application is notified that the verb has completed when the next verb on that queue specifies that a handle completes.

It is guaranteed that all preceding verbs with no handle are complete when the event is signaled after completion of a verb on the same queue that does not specify a handle.

When a nonblocking verb returns the flag **AP_OPERATION_INCOMPLETE**, it set in the **opext** field.

The APPC verbs that can be issued in nonblocking mode on the allocate queue are:

(MC_)ALLOCATE
(MC_)SEND_CONVERSATION

The APPC verbs that can be issued in nonblocking mode on the send/receive queue are:

(MC_)CONFIRM
(MC_)CONFIRMED
(MC_)DEALLOCATE
(MC_)FLUSH
(MC_)PREPARE_TO_RECEIVE
(MC_)RECEIVE_AND_WAIT
(MC_)RECEIVE_IMMEDIATE
(MC_)SEND_DATA
(MC_)SEND_ERROR

The APPC verb that can be issued in nonblocking mode on the send-expedited queue is:

(MC_)REQUEST_TO_SEND

The following APPC verbs are always processed asynchronously but are not associated with any queue:

(MC_)RECEIVE_AND_POST
(MC_)TEST_RTS_AND_POST

The Personal Communications APPC verbs that cannot be issued in nonblocking mode (and are processed in blocking mode if the application sets the nonblocking flag) are:

(MC_)GET_ATTRIBUTES

GET_TP_PROPERTIES
GET_TYPE
RECEIVE_ALLOCATE
TEST_RTS
TP_ENDED
TP_STARTED
CNOS

An application cannot issue verbs in nonblocking mode for the send/receive queue or the send-expedited queue until an **ALLOCATE** or **RECEIVE_ALLOCATE** verb has returned successfully (Personal Communications returns AP_PARAMETER_CHECK, and AP_BAD_CONV_ID).

A nonblocking verb issued for the send/receive queue or the send-expedited queue, with another (blocking or nonblocking) verb currently outstanding on the same queue, is added to that queue, and is only processed when the outstanding verb has completed.

A blocking verb issued when any other verb (for the same conversation) is outstanding is rejected by Personal Communications (with **primary_rc** AP_TP_BUSY). Note that **RECEIVE_AND_POST** is treated as a blocking verb in this respect, but **TEST_RTS_AND_POST** can be issued with other verbs outstanding on the same conversation (and is not placed in any of the nonblocking verb queues).

Default Local LU Pool

Personal Communications supports a default local LU pool for dependent LU 6.2. See *Personal Communications AS/400 and 3270 for Windows 95 System Management Programming* for details on configuring LUs as members of this pool. LUs in the pool are accessed through the APPC API when the application issues **TP_STARTED** verbs (see “TP_STARTED” on page 60). If **TP_STARTED** is issued with a blank **lu_alias** field, Personal Communications chooses an LU from the default pool, or uses the control point LU, as follows:

If LUs have been configured as members of the default local LU pool, Personal Communications chooses an LU from the pool that is not in use. If all the LUs in the pool are in use, the **TP_STARTED** verb fails.

If No LUs have been configured as members of the default local LU pool, Personal Communications uses the control point LU.

Chapter 5. Entry Points

The following sections describe the procedure entry points for APPC.

APPC

You can use this as a synchronous entry point for all of the APPC verbs. Alternatively, you can use this entry point to issue nonblocking verbs by putting an event handle in the secondary return code field and setting the queue-level nonblocking flag in the **opext** field (AP_NON_BLOCKING).

Syntax

void WINAPI APPC(long)

Input is a pointer to a verb control block.

Returned Values

Examine the primary return code and secondary return code for any errors.

Usage Notes

See also: "WinAsyncAPPCEx()" on page 30.

© Copyright IBM Corp. 1994, 1996

WinAsyncAPPC()

This is an asynchronous entry point for all of the APPC verbs. An application uses this entry point if it chooses to be notified of completion through a Windows. message. Personal Communications provides this entry point for compatibility with existing applications.

Syntax

HANDLE WINAPI WinAsyncAPPC(hWnd,
long)

Returned Values

The return value specifies whether the asynchronous request completed successfully. If the request was successful, the actual return value is a handle. If the function was not successful, Personal Communications returns a .

Usage Notes

APPC verbs that can block are as follows:

[MC_]ALLOCATE
[MC_]CONFIRM
[MC_]CONFIRMED
[MC_]DEALLOCATE
[MC_]FLUSH
[MC_]PREPARE_TO_RECEIVE
RECEIVE_ALLOCATE
[MC_]RECEIVE_AND_WAIT
[MC_]REQUEST_TO_SEND
[MC_]SEND_CONVERSATION
[MC_]SEND DATA
[MC_]SEND_ERROR
TP_ENDED
TP_STARTED

The WinAsyncAPPC entry point permits the verb to be canceled, but does not support queue-level nonblocking. The APPC entry point supports queue-level nonblocking, but does not permit the application to cancel the verb.

This entry point does not support queue-level nonblocking. If the queue-level nonblocking flag AP_NON_BLOCKING is specified on the asynchronous interface, Personal Communications ignores it. When using the asynchronous entry point, an application can have only one outstanding function in progress on a conversation at a time. An attempt to initiate a second function results in the error code AP_CONV_BUSY. If an application needs to be notified of asynchronous completion through an event handle, it can use either the **WinAsyncAPPCEx** or **APPC** entry point. The exceptions to the previous paragraph are **RECEIVE_AND_POST** and **RECEIVE_AND_WAIT**. To enable full use to be made of the asynchronous support, Personal Communications alters asynchronously issued **RECEIVE_AND_WAIT** verbs to act like the **RECEIVE_AND_POST** verb. Specifically, while an asynchronous **RECEIVE_AND_POST** or **RECEIVE_AND_WAIT** is outstanding, an application can issue the following verbs on the same conversation:

REQUEST_TO_SEND
GET_TYPE

GET_ATTRIBUTES
TEST_RTS
DEALLOCATE (AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER)
SEND_ERROR
TP_ENDED

This enables an application, such as a server, to use an asynchronous **RECEIVE_AND_WAIT** to receive data. While the **RECEIVE_AND_POST** or **RECEIVE_AND_WAIT** is outstanding, the application can still use **SEND_ERROR** and **REQUEST_TO_SEND**.

When the asynchronous operation is complete, the application's window *hWnd* receives the message returned by **RegisterWindowMessage** with "**WinAsyncAPPC**" as the input string. The *wParam* argument contains the asynchronous task handle returned by the original function call. The *lParam* argument contains the original VCB pointer and can be used to determine the final return code.

WinAPPCCancelAsyncRequest permits an application to cancel any asynchronous APPC action, but terminates the related conversation or transaction program as appropriate. Any outstanding operations return with AP_CANCELED as the return code.

If the function returns successfully, Personal Communications posts a **WinAsyncAPPC()** message to the application when the operation completes or the conversation is canceled.

See also:

"WinAsyncAPPCEx()" on page 30
"WinAPPCCancelAsyncRequest()" on page 32

WinAsyncAPPCEx()

This is an asynchronous entry point for all of the APPC ve rbs. Use this call to enable multiple sessions to be handled on the same thread.

Use this entry point if you want the application to be notified of completion through an event and your application requires the ability to cancel outstanding verbs; otherwise, use the APPC queue-level nonblocking entry point.

Syntax

```
HANDLE WINAPI WinAsyncAPPCEx(HANDLE,
                             LPAPPC);
```

Returned Values

The return value specifies whether the asynchronous resolution request was successful. If the extension was successful, the actual return value is a handle. If the function was not successful, Personal Communications returns a .

Usage Notes

This verb is intended for use with **WaitForMultipleObjects** in the Win32 API.

APPC verbs that can block are as follows:

[MC_]ALLOCATE
[MC_]CONFIRM
[MC_]CONFIRMED
[MC_]DEALLOCATE
[MC_]FLUSH
[MC_]PREPARE_TO_RECEIVE
RECEIVE_ALLOCATE
[MC_]RECEIVE_AND_WAIT
[MC_]REQUEST_TO_SEND
[MC_]SEND_CONVERSATION
[MC_]SEND_DATA
[MC_]SEND_ERROR
TP_ENDED
TP_STARTED

This entry point does not suppport queue-level nonblocking. If the queue-level nonblocking flag AP_NON_BLOCKING is specified on the asynchronous interface, Personal Communications ignores it. When using the asynchronous entry point, an application can have only one outstanding function in progress on a conversation at a time. An attempt to initiate a second function results in the error code AP_CONV_BUSY.

The **WinAsyncAPPCEx** entry point permits the verb to be canceled, but does not support queue-level nonblocking. The APPC entry point supports queue-level nonblocking, but does not permit the application to cancel the verb. The exceptions to the previous paragraph are **RECEIVE_AND_POST** and **RECEIVE_AND_WAIT**. To enable full use to be made of the asynchronous support, Personal Communications alters asynchronously issued **RECEIVE_AND_WAIT** verbs to act like the **RECEIVE_AND_POST** verb. Specifically, while an asynchronous **RECEIVE_AND_POST** or **RECEIVE_AND_WAIT** is outstanding, an application can issue the following verbs on the same conversation:

REQUEST_TO_SEND
GET_TYPE
GET_ATTRIBUTES
TEST_RTS
DEALLOCATE (AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER)
SEND_ERROR
TP_ENDED

This enables an application, and in particular, a server application, to use an asynchronous **RECEIVE_AND_WAIT** to receive data. While the **RECEIVE_AND_POST** or **RECEIVE_AND_WAIT** is outstanding, the application can still use **SEND_ERROR** and **REQUEST_TO_SEND**.

When the asynchronous operation is complete, Personal Communications notifies the application by the signaling of the event. When the application receives the signal, it examines the primary return code and secondary return code for any error conditions.

See also:

"WinAsyncAPPC()" on page 28
"WinAPPCCancelAsyncRequest()" on page 32
"APPC" on page 27

WinAPPCCancelAsyncRequest()

This function cancels an outstanding **WinAsyncAPPC**-based request.

```
int WINAPI WinAPPCCancelAsyncRequest(HANDLE
                                     handle
```

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

Returned Values

The return value specifies whether the asynchronous request was canceled. If the value is 0, Personal Communications canceled the request. Otherwise, the value is one of the following error codes:

WAPPCINVALID
The specified asynchronous task ID was not valid.

WAPPCALREADY
The asynchronous routine to be canceled has already completed.

Usage Notes

An application program can cancel an asynchronous task that was previously issued with one of the **WinAsyncAPPC** functions prior to completion, by issuing the **WinAPPCCancelAsyncRequest()** call, and specifying the asynchronous event as returned by the initial function in the handle.

If the outstanding verb relates to a conversation (for example, **SEND_DATA** or **RECEIVE_AND_WAIT**), Personal Communications purges the verb and deactivates the session. If the verb relates to a transaction program (for example, **RECEIVE_ALLOCATE** or **TP_STARTED**), Personal Communications ends the transaction program. In both cases, although Personal Communications deactivates conversations and sessions as cleanly as possible, it does not flush send buffers or waiting-for-confirmations or other pending actions. This call is synchronous. After the processing described above is complete, Personal Communications posts a completion message for the canceled verb.

If an attempt to cancel an existing asynchronous **WinAsyncAPPC** routine fails with an error code of WAPPCALREADY, the original routine has already completed. Either the application has dealt with the resulting notification, or the application has not dealt with the completion notification. It is not possible to cancel an asynchronous verb issued through the APPC queue-level nonblocking entry point.

See also: "WinAsyncAPPC()" on page 28.

WinAPPCCancelBlockingCall()

This function cancels any outstanding blocking operation for its thread. If Personal Communications cancels an outstanding blocked call, it generates an error code of AP_CANCELLED. Use this call only from within a blocking hook function. Personal Communications provides this function for backward compatibility with existing applications.

Syntax

Int WINAPI WINAPPCCancelBlockingCall(void);

Returned Values

The return value specifies whether the cancellation request was successful. If the value is 0, Personal Communications canceled the request. Otherwise, the value is the following error code:

WAPPCINVALID
: There is no outstanding blocking call.

Usage Notes

If the outstanding verb relates to a conversation (for example, **SEND_DATA** or **RECEIVE_AND_WAIT**), Personal Communications purges the verb and deactivates the session. If the verb relates to a transaction program (for example, **RECEIVE_ALLOCATE** or **TP_STARTED**), Personal Communications ends the transaction program. In both cases, although Personal Communications deactivates conversations and sessions as cleanly as possible, it does not flush send buffers or waiting-for-confirmations or other pending actions. This call is synchronous. After the processing described above is complete, the function is finished.

A multithreaded application can have multiple blocking operations outstanding, but only one per thread. To distinguish between multiple outstanding calls, **WinAPPCCancelBlockingCall()** cancels the outstanding operation on the current, or calling, application thread if one exists; otherwise, it fails. APPC suspends the calling application thread while an operation is outstanding. As a result, the thread on which the blocking operation was initiated does not regain control (and therefore, is not be able to issue a call to **WinAPPCCancelBlockingCall()**) unless the application has previously registered a blocking hook for the thread by using **WinAPPCSetBlockingHook**.

WinAPPCCleanup()

This function terminates and deregisters an application from the APPC API.

Syntax

BOOL WINAPI WinAPPCCleanup(void);

Returned Values

The return value specifies whether the deregistration was successful. If the value is not 0, Personal Communications has successfully deregistered the application. If Personal Communications has not deregistered the application, it returns a value of .

Usage Notes

Use **WinAPPCCleanup()** to deregister a Personal Communications application from the APPC API.

Personal Communications terminates conversations that are still active and ends transaction programs. This function is equivalent to issuing **TP_ENDED(HARD)** on all transaction programs owned by the application.

See also: "WinAPPCStartup()" on page 36.

WinAPPCIsBlocking()

This function determines if a thread is executing while waiting for a previous blocking call to finish. Personal Communications provides this function for backward compatibility with existing applications.

Syntax

BOOL WlNAPI WinAPPCIsBlocking(void);

Returned Values

The return value specifies the outcome of the function. If the value is not 0, an outstanding blocking call is awaiting completion. A value of means there is no outstanding blocking call.

Usage Notes

The Personal Communications DLL prohibits more than one blocking call per thread; it returns AP_THREAD_BLOCKING if this occurs. A thread that is executing a blocking call is not reentered unless a blocking hook function has been set. In this case, **WinAPPCIsBlocking** returns true only from within a blocking hook function.

See also:

"WinAPPCCancelBlockingCall()" on page 33
"WinAPPCSetBlockingHook()" on page 37
"WinAPPCUnhookBlockingHook()" on page 38

WinAPPCStartup()

This function enables an application to specify the version of Personal Communications required and to retrieve version information from Personal Communications. This call is not required.

Syntax

```
int WINAPI WinAPPCStartup(WORD,
                          wVersionRequired,
                          LPWAPPCDATA);
```

Parameter	Description
wVersionRequired	Specifies the version of Personal Communications support required. The high-order byte specifies the minor version (revision) number; the low-order byte specifies the major version number.

Returned Values

The return value specifies whether Personal Communications successfully registered the application and whether it can support the specified version number. If the value is , Personal Communications supports the specified version and it successfully registers the application. Otherwise, the return value is one of the following:

WAPPCSYSNOTREADY
: The underlying network subsystem is not ready for network communication.

WAPPCVERNOTSUPPORTED
: This particular Personal Communications implementation does not support the version of Personal Communications support requested.

WAPPCINVALID
: Personal Communications could not determine the specified version.

Usage Notes

WinAPPCStartup() is intended to help with compatibility of future releases of the API. This DLL supports Version 1.0.

See also: "WinAPPCCleanup()" on page 34.

WinAPPCSetBlockingHook()

This function enables an APPC implementation of the APPC API to block APPC function calls.

Personal Communications provides this function for backwards compatibility with existing applications.

Syntax

```
FARPROC WINAPI WinAPPCSetBlockingHook(FARPROC,
                                      lpBlockFunc);
```

Parameter	Description
lpBlockFunc	Specifies the procedure instance address of the blocking function to be installed.

Returned Values

The return value points to the procedure instance of the previously installed blocking function. The application or library that calls the **SetBlockingHook** function should save this return value so that it can be restored if needed. (If nesting is not important, the application can simply discard the value returned by **WinAPPCSetBlockingHook()** and eventually use **WinAPPCUnhookBlockingHook** to restore the default mechanism.)

Usage Notes

A blocking function returns FALSE if it receives a WM_QUIT message so that Personal Communications can return control to the application to process the message and terminate gracefully. Otherwise, the function returns TRUE.

No default blocking hook is implemented. If an application does not set a blocking hook, the application thread waits indefinitely for the blocking call to return.

If the blocking hook function does not return TRUE, Personal Communications returns the blocking verb to the application with the primary return code AP_CANCELLED.

This function is implemented by thread. It provides for a particular thread to replace the blocking mechanism without affecting other threads.

See also:

"WinAPPCCancelBlockingCall()" on page 33
"WinAPPCIsBlocking()" on page 35
"WinAPPCUnhookBlockingHook()" on page 38

WinAPPCUnhookBlockingHook()

This function removes any previous blocking hook that has been installed.

Personal Communications provides this function for backward compatibility with existing applications.

Syntax

BOOL WINAPI WinAPPCUnhookBlockingHook (*void*);

Returned Values

The return value specifies the outcome of the function. It is not 0 if Personal Communications successfully reinstalled the default mechanism. The value is 0 if Personal Communications did not reinstall the default mechanism.

Usage Notes

After the function is called, this application thread waits indefinitely for all future blocking calls to complete.

See also: "WinAPPCSetBlockingHook()" on page 37.

GetAppcConfig()

This function is not implemented. However, an entry point is provided for backwards compatibility. If a valid set of parameters is specified, Personal Communications returns APPC_CFG_SUCESS_NO_DEFAULT_REMOTE and puts a NULL terminator in the first byte of the RemLu buffer.

Applications should use the NOF interface to retrieve this information. For information on using the NOF interface, refer to *Personal Communications AS/400 and 3270 for Windows 95 System Management Programming*.

GetAppcReturnCode()

This function converts the primary and secondary return codes in the VCB to a printable string. It provides a standard set of error strings for use by APPC applications.

Syntax

```
int WINAPI GetAppcReturnCode (struct appc_hdr
                                vcb  UINT
                              buffer_length,
                               unsigned char  buffer_addr);
```

Parameter	Description
vcb	Supplied parameter; specifies the address of the verb control block.
buffer_length	Supplied parameter; specifies the length of the buffer pointed to by **buffer_addr**. The recommended length is 256.
buffer_addr	Supplied/returned parameter; specifies the address of the buffer that will hold the formatted, null-terminated string. Length of the string in the specified buffer.

Returned Values

0x20000001
: The parameters are not valid; the function could not read from the specified verb control block or could not write to the specified buffer.

0x20000002
: The specified buffer is too small.

Usage Notes

The descriptive error string returned in **buffer_addr** does not terminate with a new line character (**n**).

GetAppcReturnCode

Chapter 6. APPC Verbs

This chapter documents the syntax of each verb passed across the APPC API, and describes the parameters passed in and returned for each verb.

Verb Control Blocks

This section contains a general description of the fields and indications for each verb.

Common Fields

Each VCB has a number of common fields, as follows:

opcode Verb operation code: an identifying field containing the name of 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 the following sections.

secondary_rc Secondary return code. This supplements the information provided by the primary return code. Possible values for each verb are listed in the following sections. Some VCBs also contain the following fields.

opext Verb extension code. This supplements the information provided by the verb operation code. If the verb signal is to be processed in nonblocking mode, the flag AP_NON_BLOCKING must be set. In the signals described below these common fields are included, but not explained individually.

TP Identifiers An 8-byte transaction program identifier is assigned to each active transaction program. This identifier is assigned by Personal Communications.

The verb control blocks for each signal are described below.

The transaction program identifier is used to route **TP_ENDED** verbs and as a correlator on conversation verbs.

APPC API Support

Verbs Supported

Personal Communications supports the following verbs at the APPC API.

Session Limit

INITIALIZE_SESSION_LIMIT
RESET_SESSION_LIMIT

© Copyright IBM Corp. 1994, 1996

Type Independent Verbs

GET_TP_PROPERTIES
GET_TYPE
RECEIVE_ALLOCATE
TP_ENDED
TP_STARTED

Basic Conversation Verbs

ALLOCATE
CONFIRM
CONFIRMED
DEALLOCATE
FLUSH
GET_ATTRIBUTES
PREPARE_TO_RECEIVE
RECEIVE_AND_POST
RECEIVE_AND_WAIT
RECEIVE_IMMEDIATE
REQUEST_TO_SEND
SEND_CONVERSATION
SEND_DATA
SEND_ERROR
TEST_RTS
TEST_RTS_AND_POST

Mapped Conversation Verbs

MC_ALLOCATE
MC_CONFIRM
MC_CONFIRMED
MC_DEALLOCATE
MC_FLUSH
MC_GET_ATTRIBUTES
MC_PREPARE_TO_RECEIVE
MC_RECEIVE_AND_POST
MC_RECEIVE_AND_WAIT
MC_RECEIVE_IMMEDIATE
MC_REQUEST_TO_SEND
MC_SEND_CONVERSATION
MC_SEND_DATA
MC_SEND_ERROR
MC_TEST_RTS
MC_TEST_RTS_AND_POST

Indication Verb Control Blocks

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, then 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, then the verb is forwarded to the LU associated with the CP (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.

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 may 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 32767.

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 32767.

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 32767.

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 32767.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters.

primary_rc AP_OK

secondary_rc AP_AS_SPECIFIED
AP_AS_NEGOTIATED

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, Personal Communications returns the following parameters.

primary_rc	AP_PARAMETER_CHECK
secondary_rc	AP_EXCEEDS_MAX_ALLOWED AP_INVALID_SET_NEGOTIABLE AP_INVALID_PLU_NAME AP_INVALID_MODE_NAME AP_INVALID_LU_NAME AP_INVALID_LU_ALIAS

Returned Parameters: State Check

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

primary_rc	AP_STATE_CHECK
secondary_rc	AP_MODE_NOT_RESET

Returned Parameters: Node not Started

If the verb does not execute because the Node has not yet been started, Personal Communications returns the following parameters.

primary_rc	AP_NODE_NOT_STARTED

Returned Parameters: Node Stopping

If the verb does not execute because a **STOP_NODE** verb has been issued, Personal Communications returns the following parameters.

primary_rc	AP_NODE_STOPPING

Returned Parameters: Allocation Error

If the verb does not execute because of an allocation error, Personal Communications returns the following parameters.

primary_rc	AP_ALLOCATION_ERROR
secondary_rc	AP_ALLOCATION_FAILURE_NO_RETRY
sense_data	Sense data associated with allocation error.

Returned Parameters: System Error

If the verb does not execute because of a system error, Personal Communications returns the following parameters.

primary_rc	AP_UNEXPECTED_SYSTEM_ERROR

Returned Parameters: Other Errors

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

primary_rc	AP_CONV_FAILURE_NO_RETRY AP_CNOS_PARTNER_LU_REJECT
secondary_rc	AP_CNOS_COMMAND_RACE_REJECT AP_CNOS_MODE_NAME_REJECT

RESET_SESSION_LIMIT

The **RESET_SESSION_LIMIT** verb requests that the mode session limits be reset.

VCB Structure

```
typedef struct reset_session_limit
{
 unsigned short   opcode;              /  verb operation code            /
 unsigned char    reserv2;             /  reserved                       /
 unsigned char    format;              /  format                         /
 unsigned short   primary_rc;          /  primary return code            /
 unsigned long    secondary_rc;        /  secondary return code          /
 unsigned char    lu_name[8];          /  local LU name                  /
 unsigned char    lu_alias[8];         /  local LU alias                 /
 unsigned char    plu_alias[8];        /  partner LU alias               /
 unsigned char    fqplu_name[17];      /  fully qual partner LU name     /
 unsigned char    reserv3;             /  reserved                       /
 unsigned char    mode_name[8];        /  mode name                      /
 unsigned char    mode_name_select;    /  select mode name               /
 unsigned char    set_negotiable;      /  set max negotiable limit?      /
 unsigned char    reserv4[8];          /  reserved                       /
 unsigned char    responsible;         /  responsible                    /
 unsigned char    drain_source;        /  drain source                   /
 unsigned char    drain_target;        /  drain target                   /
 unsigned char    force;               /  force                          /
 unsigned long    sense_data;          /  sense data                     /
} RESET_SESSION_LIMIT;
```

Supplied Parameters

The application supplies the following parameters.

opcode AP_RESET_SESSION_LIMIT

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

lu_name LU name of the local LU requested to reset session limits. This name is an 8-byte type-A EBCDIC character string. If this field is set to all zeros, then 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, then the verb is forwarded to the LU associated with the CP (the default LU).

plu_alias Alias by which the partner LU is known to the local LU. This name must match the name of a partner LU established during configuration. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. If this field is set to all zeros, the **fqplu_name** field is used to specify the required partner LU.

fqplu_name Fully qualified LU name for the partner LU. This name is 17 bytes long and is right-padded with EBCDIC spaces. It is composed of two type-A EBCDIC character strings concatenated by an EBCDIC dot. (Each name can have a

maximum length of 8 bytes with no embedded spaces.) This field is only significant if the **plu_alias** field is set to all zeros.

mode_name Name of a set of networking characteristics defined during configuration. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

mode_name_select Selects whether session limits should be reset on a single specified mode, or on all modes between the local and partner LUs.

AP_ONE
AP_ALL

set_negotiable Specifies whether the maximum negotiable session limit for this mode should be modified.

AP_YES
AP_NO

responsible Indicates whether the source (local) or target (partner) LU is responsible for deactivating sessions after the session limit is reset (AP_SOURCE or AP_TARGET).

drain_source Specifies whether the source LU satisfies waiting session requests before deactivating a session when session limits are changed or reset (AP_NO or AP_YES).

drain_target Specifies whether the target LU satisfies waiting session requests before deactivating a session when session limits are changed or reset (AP_NO or AP_YES).

force Specifies whether session limits will be set to zero even if CNOS negotiation fails (AP_YES or AP_NO).

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters.

primary_rc AP_OK

secondary_rc
AP_FORCED
AP_AS_SPECIFIED
AP_AS_NEGOTIATED

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, Personal Communications APPN returns the following parameters.

primary_rc AP_PARAMETER_CHECK

secondary_rc
AP_EXCEEDS_MAX_ALLOWED
AP_INVALID_PLU_NAME
AP_INVALID_MODE_NAME
AP_INVALID_MODE_NAME_SELECT
AP_INVALID_RESPONSIBLE
AP_INVALID_DRAIN_SOURCE
AP_INVALID_DRAIN_TARGET
AP_INVALID_FORCE
AP_INVALID_SET_NEGOTIABLE

AP_INVALID_LU_NAME
AP_INVALID_LU_ALIAS

Returned Parameters: State Check

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

primary_rc AP_STATE_CHECK

secondary_rc AP_MODE_RESET

Returned Parameters: Node not Started

If the verb does not execute because the Node has not yet been started, Personal Communications returns the following parameters.

primary_rc AP_NODE_NOT_STARTED

Returned Parameters: Node Stopping

If the verb does not execute because a **STOP_NODE** verb has been issued, Personal Communications returns the following parameters.

primary_rc AP_NODE_STOPPING

Returned Parameters: Allocation Error

If the verb does not execute because of an allocation error, Personal Communications returns the following parameters.

primary_rc AP_ALLOCATION_ERROR

secondary_rc AP_ALLOCATION_FAILURE_NO_RETRY

sense_data Sense data associated with allocation error.

Returned Parameters: System Error

If the verb does not execute because of a system error, Personal Commucations returns the following parameters.

primary_rc AP_UNEXPECTED_SYSTEM_ERROR

Returned Parameters: Other Errors

If the verb does not execute because of an error, Personal Communications 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

Type Independent Verb Control Blocks

GET_TP_PROPERTIES

GET_TP_PROPERTIES returns attributes associated with the transaction program.

VCB Structure

```
typedef struct get_tp_properties
{
 unsigned short      opcode;              / verb operation code        /
 unsigned char       reserv2[2]           / verb format                /
 unsigned short      primary_rc;          / primary return code        /
 unsigned long       secondary_rc;        / secondary return code      /
 unsigned char       tp_id[8];            / TP identifier              /
 unsigned char       tp_name[64];         / TP name                    /
 unsigned char       lu_alias[8];         / LU alias                   /
 unsigned char       luw_id[ ];           / LUW identifier             /
 unsigned char       fqlu_name[17];       / fully qualified LU name    /
 unsigned char       reserv3[ ];          / reserved                   /
 unsigned char       user_id[1 ];         / user id                    /
} GET_TP_PROPERTIES;

typedef struct luw_id_overlay
{
 unsigned char      fq_lu_name_len;       / fully qualified LU name length /
 unsigned char      fq_luw_name[17];
                                          / fully qualified LU name        /
 unsigned char      instance[6];          / instance number                /
 unsigned char      sequence[2];          / sequence number                /
} LUW_ID_OVERLAY;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_GET_TP_PROPERTIES

tp_id Identifier for the local transaction program. The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

tp_name Name of the local transaction program, that is, the transaction program issuing this verb. Personal Communications does not check the character set of this field.

lu_alias Alias of the local LU associated with the transaction program. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set.

The **luw_id** field is a Logical Unit of Work identifier associated with unprotected conversations (conversations with **sync_level** of AP_NONE or AP_CONFIRM_SYNC_LEVEL). The **luw_id_overlay** contains the following parameters:

luw_id_overlay.fq_lu_name_len
Length of fully qualified LU name associated with Logical Unit of Work.

luw_id_overlay.fq_lu_name
Fully qualified LU name associated with Logical Unit of Work. This name is up to 17 bytes long and is right-padded with EBCDIC blanks. 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 blanks. If the network ID is not present, then omit the dot.) If the name length is less than 17 bytes, **instance** and **sequence** immediately follow the name (note that this means the fields of the LUW_ID_OVERLAY structure cannot be used to access either **instance** or **sequence**).

luw_id_overlay.instance
Logical Unit of Work instance number. This is a binary string of length 6 bytes.

luw_id_overlay.sequence
Logical Unit of Work sequence number. This is a binary string of length 2 bytes.

If **luw_id_overlay.fq_lu_name_len** is less than 17, **luw_id_overlay** is right padded with EDCDIC blanks (after **instance** and **sequence**).

luw_id_overlay.fq_lu_name
Fully qualified name of the local LU associated with the transaction program. This name is 17 bytes long and is right-padded with EBCDIC blanks. 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 blanks. If the network ID is not present, then omit the dot.)

user_id User ID of the initiator of the transaction. This is a 10-byte type-AE EBCDIC character string, padded to the right with EBCDIC spaces.

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_TP_ID

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

GET_TYPE

The **GET_TYPE** verb returns the conversation type (basic or mapped) of a particular conversation.

VCB Structure

```
typedef struct get_type
{
 unsigned short     opcode;          /  verb operation code       /
 unsigned char      opext;           /  verb extension code       /
 unsigned char      format;          /  format                        /
 unsigned short     primary_rc;      /  primary return code       /
 unsigned long      secondary_rc;    /  secondary return code     /
 unsigned char      tp_id[8];        /  TP identifier             /
 unsigned long      conv_id;         /  conversation identifier  /
 unsigned char      conv_type;       /  conversation type         /
} GET_TYPE;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_GET_TYPE

opext AP_BASIC_CONVERSATION

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

tp_id Identifier for the local transaction program. The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier. The value of this parameter was returned by the **ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

conv_type Conversation type of the conversation identified by **conv_id**.

AP_BASIC_CONVERSATION
AP_MAPPED_CONVERSATION

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_TP_ID
AP_BAD_CONV_ID

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

RECEIVE_ALLOCATE

The **RECEIVE_ALLOCATE** verb requests information needed to establish a new transaction program for a conversation with a partner transaction program that has issued an **ALLOCATE** or **MC_ALLOCATE** verb.

VCB Structure

```
typedef struct receive_allocate
{
 unsigned shor    opcode;                  / verb operation code                  /
 unsigned char    opext;                   / verb extension code                  /
 unsigned shor    primary_rc;              / primary return code                  /
 unsigned long    secondary_rc;            / secondary return code                /
 unsigned char    tp_name[64];             / TP name                              /
 unsigned char    tp_id[8];                / TP identifier                        /
 unsigned long    conv_id;                 / conversation identifier              /
 unsigned char    sync_level;              / sync Level                           /
 unsigned char    conv_type;               / conversation type                    /
 unsigned char    user_id[1 ];             / user ID                              /
 unsigned char    lu_alias[8];             / LU alias                             /
 unsigned char    plu_alias[8];            / partner LU alias                     /
 unsigned char    mode_name[8];            / mode name                            /
 unsigned char    reserv3[2];              / reserved                             /
 unsigned long    conv_group_id;           / conversation group ID                /
 unsigned char    fqplu_name[17];          / fully qualified partner LU name /
 unsigned char    pip_incoming;            / received PIP data                    /
 unsigned char    reserv4[4];              / reserved                             /
} RECEIVE_ALLOCATE;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_RECEIVE_ALLOCATE

opext AP_BASIC_CONVERSATION

tp_name Name of the transaction program. Personal Communications does not check the character set of this field.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

tp_id Identifier for the local transaction program. This value is assigned by Personal Communications to the transaction program. The transaction program passes this identifier to Personal Communications on all subsequent APPC verbs.

conv_id Conversation identifier. This value identifies the conversation established between the two transaction programs.

sync_level Synchronization level of the conversation.

AP_CONFIRM_SYNC_LEVEL
AP_NONE

conv_type Conversation type of the conversation identified by **conv_id**.

AP_BASIC_CONVERSATION
AP_MAPPED_CONVERSATION

user_id User ID supplied by the partner transaction program. This is a 10-byte type-AE EBCDIC character string, padded to the right with EBCDIC spaces.

lu_alias Alias by which the local LU is known. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set.

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.

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.

conv_group_id Conversation group identifier for the session being used by this conversation.

fqplu_name Fully qualified LU name for the partner LU. This name is 17 bytes long and is right-padded with EBCDIC blanks. 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 blanks. If the network ID is not present, omit the dot.)

pip_incoming Specifies whether the partner transaction program-supplied Program Initialization Parameters (PIP) on the [MC_]ALLOCATE request. Set to AP_YES or AP_NO. If AP_YES, the PIP data will be received on the first **[MC_]RECEIVE_*** verb issued on this conversation.

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_UNDEFINED_TP_NAME

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_UNEXPECTED_SYSTEM_ERROR

TP_ENDED

The **TP_ENDED** verb notifies Personal Communications that a specified transaction program has ended.

VCB Structure

```
typedef struct tp_ended
{
 unsigned short     opcode;          /  verb operation code      /
 unsigned char      opext;           /  verb extension code      /
 unsigned char      format;          /  format                       /
 unsigned short     primary_rc;      /  primary return code      /
 unsigned long      secondary_rc;    /  secondary return code  /
 unsigned char      tp_id[8];        /  TP identifier              /
 unsigned char      type;            /  type of TP ended          /
} TP_ENDED;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_TP_ENDED

opext AP_BASIC_CONVERSATION

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

tp_id Identifier for the local transaction program. The value of this parameter was returned by the **TP_STARTED** verb for an invoking transaction program, or by the **RECEIVE_ALLOCATE** verb for an invoked transaction program.

type Type of **TP_ENDED**.

AP_HARD
AP_SOFT
AP_ABEND
AP_CANCEL

If type is AP_ABEND, Personal Communications does not reply to the **TP_ENDED** signal.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_TP_ID
AP_BAD_TYPE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

TP_STARTED

The **TP_STARTED** verb notifies Personal Communications that a program has requested resources for a transaction program initiated as a result of a local command, rather than an incoming allocation request.

VCB Structure

```
typedef struct tp_started
{
 unsigned char     opext;                  /  verb extension                  /
 unsigned short    primary_rc;             /  primary return code             /
 unsigned long     secondary_rc;           /  secondary return code           /
 unsigned char     lu_alias[8];            /  LU alias                        /
 unsigned char     tp_id[8];               /  TP identifier                   /
} TP_STARTED;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_TP_STARTED

opext AP_BASIC_CONVERSATION

lu_alias Alias by which the local LU is known. If it is set to zero, Personal Communications uses the control point LU. This is an 8-byte string in a locally displayable character set. All 8 bytes are significant and must be set. A blank **lu_alias** field is accepted. The control point LU is used in this case.

tp_name Name of the transaction program. Personal Communications does not check the character set of this field.

Returned Parameters: Successful Execution

If the verb was executed successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

tp_id Identifier for the local transaction program. This value is assigned by Personal Communications to the transaction program. The transaction program passes this identifier to Personal Communications on all subsequent APPC verbs.

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, Personal Communications returns the following parameters.

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_INVALID_LU_NAME
AP_INVALID_ENABLE_POOL

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_UNEXPECTED_SYSTEM_ERROR

Basic Conversation Verb Control Blocks

ALLOCATE

The **ALLOCATE** verb is issued by the invoking transaction program. This verb allocates a session between the local LU and the partner LU and then (in conjunction with the **RECEIVE_ALLOCATE** verb) establishes a conversation between the invoking transaction program and the invoked transaction program.

The **ALLOCATE** verb can establish either a basic or mapped conversation. The use of the **ALLOCATE** verb to establish a mapped conversation enables the transaction program to use basic conversation verbs to communicate with a mapped conversation partner transaction program.

Personal Communications generates a conversation identifier (**conv_id**) when this verb executes successfully. This identifier is a parameter that is required for all other APPC conversation verbs.

VCB Structure

```
typedef struct allocate
{
 unsigned short     opcode;                  / verb operation code                  /
 unsigned char      opext;                   / verb extension code                  /
 unsigned char      format;                  / format                                /
 unsigned short     primary_rc;              / primary return code                  /
 unsigned long      secondary_rc;            / secondary return code                /
 unsigned char      tp_id[8];                / TP identifier                        /
 unsigned long      conv_id;                 / conversation identifier              /
 unsigned char      conv_type;               / conversation type                    /
 unsigned char      sync_level;              / sync level                           /
 unsigned char      reserv3[2];              / reserved                             /
 unsigned char      rtn_ctl;                 / return control                       /
 unsigned char      reserv3a;                / reserved                             /
 unsigned long      conv_group_id;           / conversation group identifier /
 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];             / partner TP name                      /
 unsigned char      security;                / security level                       /
 unsigned char      pwd[1 ];                 / security password                    /
 unsigned char      user_id[1 ];             / security user_id                     /
 unsigned short     pip_dlen;                / PIP data length                      /
 unsigned char      pip_dptr;                / pointer to PIP data                  /
 unsigned char      reserv5a;                / reserved                             /
 unsigned char      fqplu_name[17];          / fully qualified partner LU           /
                                                / name                                /
 unsigned char      reserv6[8];              / reserved                             /
} ALLOCATE;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_B_ALLOCATE

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

opext AP_BASIC_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb for an invoking transaction program, or by the **RECEIVE_ALLOCATE** verb for an invoked transaction program.

conv_type Type of conversation to allocate.

AP_BASIC_CONVERSATION
AP_MAPPED_CONVERSATION

If the **ALLOCATE** verb establishes a mapped conversation, the local transaction program must issue basic-conversation verbs and provide its own mapping layer to convert data records to logical records and logical records to data records. The partner transaction program can issue basic-conversation verbs and provide the mapping layer, or it can use mapped-conversation verbs (if the implementation of APPC that the partner transaction program is using supports mapped-conversation verbs). For further information, see *IBM Systems Network Architecture: LU 6.2 Reference: Peer Protocols*.

sync_level Synchronization level of the conversation.

AP_CONFIRM_SYNC_LEVEL
AP_NONE

rtn_ctl Specifies when the local LU acting on a session request from the local transaction program is to return control to the local transaction program.

AP_IMMEDIATE
AP_WHEN_SESSION_ALLOCATED
AP_WHEN_SESSION_FREE
AP_WHEN_CONV_GROUP_ALLOC
AP_WHEN_CONWINNER_ALLOC
AP_WHEN_CONLOSER_ALLOC

conv_group_id Conversation group identifier for the session to be allocated. This parameter is only supplied if **rtn_ctl** is set to AP_WHEN_CONV_GROUP_ALLOC.

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 field is set to all zeros, Personal Communications uses the **fqplu_name** field to specify the required partner LU.

mode_name Name of a set of networking characteristics usually defined during configuration. 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. Personal Communications does not check the character set of this field. The value of **tp_name** specified by the **ALLOCATE** verb in the invoking transaction program must match the value of **tp_name** specified by the **RECEIVE_ALLOCATE** verb in the invoked transaction program.

security Specifies the information the partner LU requires in order to validate access to the invoked transaction program.

AP_NONE
AP_PGM
AP_SAME

pwd Password associated with **user_id**. This is a 10-byte type-AE EBCDIC character string, padded to the right with EBCDIC spaces. This is required if Security=Program (AP_PGM); otherwise, it is optional.

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. This is required if Security=Program (AP_PGM); otherwise, it is optional.

pip_dlen Length of the program initialization parameters (PIP) to be passed to the partner transaction program. Range: 0–32767

pip_dptr Address of buffer containing PIP data. Use this parameter only if **pip_dlen** is greater than zero.

fqplu_name Fully qualified LU name for the partner LU. This name is 17 bytes long and is right-padded with EBCDIC blanks. 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 blanks. If the network ID is not present, then omit the dot.) This field is only significant if the **plu_alias** field is set to all zeros.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

conv_id Conversation identifier. This value identifies the conversation established between the two transaction programs.

conv_group_id Conversation group identifier of the session allocated to the conversation.

Returned Parameters: No Session

If the **rtn_ctl** parameter was set to AP_IMMEDIATE, and no session is available immediately, Personal Communications returns the following parameters:

primary_rc AP_UNSUCCESSFUL

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc
AP_BAD_CONV_TYPE
AP_BAD_RETURN_CONTROL
AP_BAD_SECURITY
AP_BAD_SYNC_LEVEL
AP_BAD_TP_ID
AP_PIP_LEN_INCORRECT
AP_UNKNOWN_PARTNER_MODE

Returned Parameters: Other Conditions

sense_data Provides additional information on the reason why the **ALLOCATE** failed.

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_ALLOCATION_ERROR

- AP_ALLOCATION_FAILURE_NO_RETRY
- AP_ALLOCATION_FAILURE_RETRY

AP_TP_BUSY
AP_UNSUCCESSFUL
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

If the **primary_rc** is set to AP_ALLOCATION_ERROR, the **sense_data** field carries more information on the failure.

CONFIRM

The **CONFIRM** verb sends the contents of the local LUs send buffer and a confirmation request to the partner transaction program. In response to the **CONFIRM** verb, the partner transaction program normally issues the **CONFIRMED** verb to confirm that it has received the data without error. (If the partner transaction program encounters an error, it issues the **SEND_ERROR** verb or abnormally deallocates the conversation.)

The transaction program can issue the **CONFIRM** verb only if the conversation's synchronization level, established by the **ALLOCATE** verb, is AP_CONFIRM_SYNC_LEVEL.

VCB Structure

```
typedef struct confirm
{
 unsigned short     opcode;          /  verb operation code          /
 unsigned char      opext;           /  verb extension code          /
 unsigned char      format;          /  format                           /
 unsigned short     primary_rc;      /  primary return code          /
 unsigned long      secondary_rc;    /  secondary return code        /
 unsigned char      tp_id[8];        /  TP identifier                /
 nsigned long       conv_id;         /  conversation identifier    /
 unsigned char      rts_rcvd;        /  request to send received  /
} CONFIRM;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_B_CONFIRM

opext AP_BASIC_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

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

tp_id Identifier for the local transaction program. The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier. The value of this parameter was returned by the **ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

rts_rcvd Request-to-send-received indicator.

AP_YES
AP_NO

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID
AP_CONFIRM_ON_SYNC_LEVEL_NONE

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_CONFIRM_BAD_STATE
AP_CONFIRM_NOT_LL_BDY

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_PROG_ERROR_PURGING
AP_SVC_ERROR_PURGING
AP_TP_BUSY
AP_CANCELLED

CONFIRMED

The **CONFIRMED** verb replies to a confirmation request from the partner transaction program. It informs the partner transaction program that the local transaction program has not detected an error in the received data.

Because the transaction program issuing the confirmation request waits for a confirmation, the **CONFIRMED** verb synchronizes the processing of the two TPs.

VCB Structure

```
typedef struct confirmed
{
 unsigned short     opcode;          /  verb operation code        /
 unsigned char      opext;           /  verb extension code        /
 unsigned char      format;          /  format                        /
 unsigned long      secondary_rc;    /  secondary return code      /
 unsigned char      tp_id[8];        /  TP identifier              /
 unsigned long      conv_id;         /  conversation identifier  /
} CONFIRMED;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_B_CONFIRMED

opext AP_BASIC_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

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

tp_id Identifier for the local transaction program. The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier. The value of this parameter was returned by the **ALLOCATE** verb in the invoking transaction processor or by **RECEIVE_ALLOCATE** in the invoked transaction processor.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameter

primary_rc AP_OK

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction processor issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_CONFIRMED_BAD_STATE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR
AP_CONVERSATION_TYPE_MIXED

DEALLOCATE

The **DEALLOCATE** verb deallocates a conversation between two transaction programs. Before deallocating the conversation, this verb performs the equivalent of one of the following verbs:

The **FLUSH** verb, which sends the contents of the local LU's send buffer to the partner LU (and transaction processor).

The **CONFIRM** verb, which sends the contents of the local LU's send buffer and a confirmation request to the partner transaction programs.

After this verb has successfully executed, the conversation ID is no longer valid.

VCB Structure

```
typedef struct deallocate
{
 unsigned short      opcode;          /  verb operation code         /
 unsigned char       opext;           /  verb extension code         /
 unsigned char       format;          /  format                          /
 unsigned short      primary_rc;      /  primary return code         /
 unsigned long       secondary_rc;    /  secondary return code       /
 unsigned char       tp_id[8];        /  TP identifier               /
 unsigned long       conv_id;         /  conversation identifier  /
 unsigned char       dealloc_type;    /  deallocate type             /
 unsigned short      log_dlen;        /  log data length             /
 unsigned char       log_dptr;        /  pointer to log data         /
} DEALLOCATE;
```

Supplied Parameters

The transaction programs supplies the following parameters to Personal Communications.

opcode AP_B_DEALLOCATE

opext AP_BASIC_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

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

tp_id Identifier for the local transaction program. The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction processor or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier. The value of this parameter was returned by the **ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

dealloc_type Specifies how to perform the deallocation.

AP_ABEND_PROG
AP_ABEND_SVC
AP_ABEND_TIMER
AP_FLUSH
AP_SYNC_LEVEL
AP_TP_NOT_AVAIL_NO_RETRY
AP_TP_NOT_AVAIL_RETRY
AP_TPN_NOT_RECOGNIZED

AP_PIP_DATA_NOT_ALLOWED
AP_PIP_DATA_INCORRECT
AP_RESOURCE_FAILURE_NO_RETRY
AP_CONV_TYPE_MISMATCH
AP_SYNC_LVL_NOT_SUPPORTED
AP_SECURITY_PARAMS_INVALID

log_dlen Number of bytes of data to be sent to the error log file.

Range: 0–32767

The application can append data to the end of the VCB, in which case this field will be greater than zero and **log_dptr** must be set to NULL. (A length of zero indicates that there is no error log data.)

log_dptr Address of data buffer containing error information. The application can append data to the end of the VCB, in which case **log_dptr** must be set to NULL.

This data is sent to the local error log and to the partner LU. The transaction processor must format the error data as a General Data Stream (GDS) error log variable. For further information, see *IBM Systems Network Architecture: LU 6.2 Reference: Peer Protocols*.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameter:

primary_rc AP_OK

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID
AP_DEALLOC_BAD_TYPE
AP_DEALLOC_LOG_LL_WRONG

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction processor issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_DEALLOC_CONFIRM_BAD_STATE
AP_DEALLOC_FLUSH_BAD_STATE
AP_DEALLOC_NOT_LL_BDY

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_PROG_ERROR_PURGING
AP_SVC_ERROR_PURGING
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

FLUSH

The **FLUSH** verb sends the contents of the local LU's send buffer to the partner LU (and transaction program). If the send buffer is empty, no action takes place.

VCB Structure

```
typedef struct flush
{
 unsigned short     opcode;          /  verb operation code        /
 unsigned char      opext;           /  verb extension code        /
 unsigned char      format;          /  format                        /
 unsigned short     primary_rc;      /  primary return code       /
 unsigned long      secondary_rc;    /  secondary return code     /
 unsigned char      tp_id[8];        /  TP identifier             /
 unsigned long      conv_id;         /  conversation identifier  /
} FLUSH;
```

Supplied Parameters

The transaction processor supplies the following parameters to Personal Communications.

opcode AP_B_FLUSH

opext AP_BASIC_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

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

tp_id Identifier for the local transaction program. The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier. The value of this parameter was returned by the **ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameter:

primary_rc AP_OK

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_FLUSH_NOT_SEND_STATE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR

GET_ATTRIBUTES

The **GET_ATTRIBUTES** verb returns the attributes of the conversation.

VCB Structure

```
typedef struct get_attributes
{
 unsigned short     opcode;                  / verb operation code                /
 unsigned char      opext;                   / verb extension code               /
 unsigned short     primary_rc;              / primary return code               /
 unsigned long      secondary_rc;            / secondary return code             /
 unsigned char      tp_id[8];                / TP identifier                     /
 unsigned long      conv_id;                 / conversation identifier           /
 unsigned char      reserv3;                 / reserved                          /
 unsigned char      sync_level;              / sync_level                        /
 unsigned char      mode_name[8];            / mode name                         /
 unsigned char      net_name[8];             / network name of local LU          /

 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      plu_un_name[8];
                                             / partner LU uninterpreted name /
 unsigned char      reserv4[2];              / reserved                          /
 unsigned char      fqplu_name[17];
                                             / fully qualified partner LU        /
                                             / name                              /
 unsigned char      reserv5;                 / reserved                          /
 unsigned char      user_id[1 ];             / user identifier                   /
 unsigned long      conv_group_id;           / conversation group identifier /
 unsigned char      conv_corr_len;           / conversation correlator           /
                                             / length                            /
 unsigned char      conv_corr[8];            / conversation correlator           /
 unsigned char      reserv6[13];             / reserved                          /
} GET_ATTRIBUTES;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_B_GET_ATTRIBUTES

opext AP_BASIC_CONVERSATION.

tp_id Identifier for the local transaction program

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

mode_name Name of the set of networking characteristics associated with the session allocated to the conversation. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

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

lu_alias Alias by which the local 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.

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.

plu_un_name Uninterpreted name of partner LU, that is, the name of the partner LU as defined at the system services control point (SSCP). This is an 8-byte type-A EBCDIC character string.

fqplu_name Fully qualified name of the partner LU. This name is 17 bytes long and is right-padded with EBCDIC blanks. 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 blanks. If the network ID is not present, then omit the dot.)

user_id User ID sent by the invoking transaction program through the **ALLOCATE** verb to access the invoked transaction program (if applicable). This is a 10-byte type-AE EBCDIC character string, padded to the right with EBCDIC spaces.

conv_group_id The conversation group identifier of the session allocated to the conversation.

conv_corr_len Always set to 0.

Range: 0–8

conv_corr Always set to 0.

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR

PREPARE_TO_RECEIVE

The **PREPARE_TO_RECEIVE** verb changes the state of the conversation for the local transaction program from SEND or SEND_PENDING to RECEIVE.

Before changing the conversation state, this verb performs the equivalent of one of the following verbs:

The **FLUSH** verb, which sends the contents of the local LU's send buffer to the partner LU (and transaction program).

The **CONFIRM** verb, which send the contents of the local LU's send buffer and a confirmation request to the partner transaction program.

After this verb has successfully executed, the local transaction program can receive data.

VCB Structure

```
typedef struct prepare_to_receive
{
 unsigned short      opcode;                 /  verb operation code          /
 unsigned char       opext;                  /  verb extension code          /
 unsigned char       format;                 /  format                           /
 unsigned short      primary_rc;            /  primary return code          /
 unsigned long       secondary_rc;          /  secondary return code       /
 unsigned char       tp_id[8];              /  TP identifier                 /
 unsigned long       conv_id;                /  conversation identifier    /
 unsigned char       ptr_type;              /  prepare to receive type    /
 unsigned char       locks;                  /  prepare to receive locks  /
} PREPARE_TO_RECEIVE;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_B_PREPARE_TO_RECEIVE

opext AP_BASIC_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

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

tp_id Identifier for the local transaction program. The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

ptr_type Specifies how to perform the state change.

AP_FLUSH
AP_SYNC_LEVEL
AP_TO_R_CONFIRM

locks Specifies when Personal Communications is to return control to the local transaction processor.

AP_LONG
AP_SHORT

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameter:

primary_rc AP_OK

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID
AP_P_TO_R_INVALID_TYPE

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction processor issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_TO_R_NOT_LL_BDY
AP_P_TO_R_NOT_SEND_STATE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_PROG_ERROR_PURGING
AP_SVC_ERROR_PURGING
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

RECEIVE_AND_POST

The **RECEIVE_AND_POST** verb receives application data and status information asynchronously. This enables the transaction program to proceed with processing while data is still arriving at the local LU.

VCB Structure

```
typedef struct receive_and_post
{
 unsigned short      opcode;          /  verb operation code          /
 unsigned char       opext;           /  verb extension code          /
 unsigned short      primary_rc;      /  primary return code          /
 unsigned long       secondary_rc;    /  secondary return code        /
 unsigned char       tp_id[8];        /  TP identifier                /
 unsigned long       conv_id;         /  conversation identifier      /
 unsigned short      what_rcvd;       /  what received                /
 unsigned char       rtn_status;      /  return status with data      /
 unsigned char       fill;            /  data fill                    /
 unsigned char       rts_rcvd;        /  request to send received     /
 unsigned char       reserv4;         /  reserved                     /
 unsigned short      max_len;         /  maximum length of received  /
                                         /  data                     /
 unsigned short      dlen;            /  actual length of received   /
                                         /  data                     /
 unsigned char         dptr;          /  pointer to data buffer      /
 unsigned long         sema;          /  post handle for verb        /
 unsigned char         reserv5;       /  reserved                    /
} RECEIVE_AND_POST;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_B_RECEIVE_AND_POST

opext AP_BASIC_CONVERSATION

tp_id Identifier for the local transaction program. The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

rtn_status Indicates whether status information and data can be returned on the same verb.

AP_YES
AP_NO

fill Indicates the manner in which the local transaction program receives data.

AP_BUFFER
AP_LL

max_len Maximum number of bytes of data the local transaction program can receive.

Range: 0–65535

This value must not exceed the length of the buffer to contain the received data.

dptr Address of the buffer to contain the data received by the local LU. The application can append data to the end of the VCB in which case **dptr** must be set to NULL.

sema The API can use this field as it requires. Personal Communications returns the same value in the verb reply. Typically the API stores some posting information in this field, as for example, a callback address or a semaphore handle.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc
AP_OK
AP_DEALLOC_NORMAL

what_rcvd Status information received with the incoming data. If **rtn_status** is set to AP_NO, this field always contains a value from the first part of the following list. If **rtn_status** is set to AP_YES, this field can contain any value from the list.

AP_NONE
AP_CONFIRM_DEALLOCATE
AP_CONFIRM_SEND
AP_CONFIRM_WHAT_RECEIVED
AP_DATA
AP_DATA_COMPLETE
AP_DATA_INCOMPLETE
AP_SEND

AP_DATA_CONFIRM
AP_DATA_COMPLETE_CONFIRM
AP_DATA_CONFIRM_DEALLOCATE
AP_DATA_COMPLETE_CONFIRM_DEALL
AP_DATA_CONFIRM_SEND
AP_DATA_COMPLETE_CONFIRM_SEND
AP_DATA_SEND
AP_DATA_COMPLETE_SEND

rts_rcvd Request-to-send-received indicator.

AP_YES
AP_NO

dlen Number of bytes of data received (the data is stored in the buffer specified by the **dptr** parameter). A length of zero indicates that no data was received. This parameter is only used if the **what_rcvd** parameter indicates that data was received.

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_RETURN_STATUS_WITH_DATA
AP_BAD_TP_ID
AP_RCV_AND_POST_BAD_FILL

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_RCV_AND_POST_BAD_STATE
AP_RCV_AND_POST_NOT_LL_BDY

Returned Parameters: Verb Cancelled

If the verb did not execute because it was canceled by another verb issued by the transaction program, Personal Communications returns the following parameter:

primary_rc AP_CANCELLED

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_DEALLOC_NORMAL
AP_PROG_ERROR_NO_TRUNC
AP_PROG_ERROR_PURGING
AP_PROG_ERROR_TRUNC
AP_SVC_ERROR_NO_TRUNC
AP_SVC_ERROR_PURGING
AP_SVC_ERROR_TRUNC
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED

AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

RECEIVE_AND_WAIT

The **RECEIVE_AND_WAIT** verb receives any data that is currently available from the partner transaction program. If no data is currently available, the local transaction program waits for data to arrive.

VCB Structure

```
typedef struct receive_and_wait
{
 unsigned short      opcode;          / verb operation code          /
 unsigned char       opext;           / verb extension code          /
 unsigned short      primary_rc;      / primary return code          /
 unsigned long       secondary_rc;    / secondary return code        /
 unsigned char       tp_id[8];        / TP identifier                /
 unsigned long       conv_id;         / conversation identifier      /
 unsigned short      what_rcvd;       / what received                /
 unsigned char       rtn_status;      / return status with data      /
 unsigned char       fill;            / data fill                    /
 unsigned char       rts_rcvd;        / request to send received     /
 unsigned char       reserv4;         / reserved                     /
 unsigned short      max_len;         / maximum length of received   /
                                        / data                       /
 unsigned short      dlen;            / actual length of received    /
                                        / data                       /
 unsigned char       dptr;            / pointer to data buffer       /
 unsigned char       reserv5[5];      / reserved                     /
} RECEIVE_AND_WAIT;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_B_RECEIVE_AND_WAIT

opext AP_BASIC_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

rtn_status Indicates whether status information and data can be returned on the same verb.

AP_YES
AP_NO

fill Indicates the manner in which the local transaction program receives data.

AP_BUFFER
AP_LL

max_len Maximum number of bytes of data the local transaction program can receive.

Range: 0–65535

This value must not exceed the length of the buffer to contain the received data.

dptr Address of the buffer to contain the data received by the local LU. The application can append data to the end of the VCB, in which case **dptr** must be set to NULL.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc
AP_OK
AP_DEALLOC_NORMAL

what_rcvd Status information received with the incoming data. If **rtn_status** is set to AP_NO, this field always contains a value from the first part of the following list. If **rtn_status** is set to AP_YES, this field can contain any value from the list.

AP_NONE
AP_CONFIRM_DEALLOCATE
AP_CONFIRM_SEND
AP_CONFIRM_WHAT_RECEIVED
AP_DATA
AP_DATA_COMPLETE
AP_DATA_INCOMPLETE
AP_SEND

AP_DATA_CONFIRM
AP_DATA_COMPLETE_CONFIRM
AP_DATA_CONFIRM_DEALLOCATE
AP_DATA_COMPLETE_CONFIRM_DEALL
AP_DATA_CONFIRM_SEND
AP_DATA_COMPLETE_CONFIRM_SEND
AP_DATA_SEND
AP_DATA_COMPLETE_SEND

rts_rcvd Request-to-send-received indicator.

AP_YES
AP_NO

dlen This parameter is only used if the **what_rcvd** parameter indicates that data was received. Number of bytes of data received (the data is stored in the buffer specified by the **dptr** parameter). A length of zero indicates that no data was received.

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_RETURN_STATUS_WITH_DATA
AP_BAD_TP_ID
AP_RCV_AND_WAIT_BAD_FILL

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_RCV_AND_WAIT_BAD_STATE
AP_RCV_AND_WAIT_NOT_LL_BDY

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_DEALLOC_NORMAL
AP_PROG_ERROR_NO_TRUNC
AP_PROG_ERROR_PURGING
AP_PROG_ERROR_TRUNC
AP_SVC_ERROR_NO_TRUNC
AP_SVC_ERROR_PURGING
AP_SVC_ERROR_TRUNC
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

RECEIVE_IMMEDIATE

The **RECEIVE_IMMEDIATE** verb receives any data or status information that is currently available from the partner transaction program. If none is currently available, the local transaction program returns immediately and does not wait.

VCB Structure

```
typedef struct receive_immediate
{
 unsigned short     opcode;          / verb operation code          /
 unsigned char      opext;           / verb extension code          /
 unsigned short     primary_rc;      / primary return code          /
 unsigned long      secondary_rc ;   / secondary return code        /
 unsigned char      tp_id[8];        / TP identifier                /
 unsigned long      conv_id;         / conversation identifier      /
 unsigned short     what_rcvd;       / what received                /
 unsigned char      rtn_status;      / return status with data      /
 unsigned char      fill;            / data fill                    /
 unsigned char      rts_rcvd;        / request to send received     /
 unsigned char      reserv4;         / reserved                     /
 unsigned short     max_len;         / maximum length of received /
                                        / data                       /
 unsigned short     dlen;            / actual length of received   /
                                        / data                       /
 unsigned char      dptr;            / pointer to data buffer       /
 unsigned char      reserv5[5];      / reserved                     /
} RECEIVE_IMMEDIATE;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_B_RECEIVE_IMMEDIATE

opext AP_BASIC_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

rtn_status Indicates whether status information and data can be returned on the same verb.

AP_YES
AP_NO

fill Indicates the manner in which the local transaction program receives data.

AP_BUFFER
AP_LL

max_len Maximum number of bytes of data the local transaction program can receive.

Range: 0–65535

This value must not exceed the length of the buffer to contain the received data.

dptr Address of the buffer to contain the data received by the local LU. The application can append data to the end of the VCB, in which case **dptr** must be set to NULL.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc
AP_OK
AP_DEALLOC_NORMAL

what_rcvd Status information received with the incoming data. If **rtn_status** is set to AP_NO, this field always contains a value from the first part of the following list. If **rtn_status** is set to AP_YES, this field can contain any value from the list.

AP_NONE
AP_CONFIRM_DEALLOCATE
AP_CONFIRM_SEND
AP_CONFIRM_WHAT_RECEIVED
AP_DATA
AP_DATA_COMPLETE
AP_DATA_INCOMPLETE
AP_SEND

AP_DATA_CONFIRM
AP_DATA_COMPLETE_CONFIRM
AP_DATA_CONFIRM_DEALLOCATE
AP_DATA_COMPLETE_CONFIRM_DEALL
AP_DATA_CONFIRM_SEND
AP_DATA_COMPLETE_CONFIRM_SEND
AP_DATA_SEND
AP_DATA_COMPLETE_SEND

rts_rcvd Request-to-send-received indicator.

AP_YES
AP_NO

dlen This parameter is only used if the **what_rcvd** parameter indicates that data was received. Number of bytes of data received (the data is stored in the buffer specified by the **dptr** parameter). A length of zero indicates that no data was received.

Returned Parameters: No Data Available

If no data is immediately available from the partner transaction program, Personal Communications returns the following parameter.

primary_rc AP_UNSUCCESSFUL

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_RETURN_STATUS_WITH_DATA
AP_BAD_TP_ID
AP_RCV_IMMD_BAD_FILL

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_RCV_IMMD_BAD_STATE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_DEALLOC_NORMAL
AP_PROG_ERROR_NO_TRUNC
AP_PROG_ERROR_PURGING
AP_PROG_ERROR_TRUNC
AP_SVC_ERROR_NO_TRUNC
AP_SVC_ERROR_PURGING
AP_SVC_ERROR_TRUNC
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

REQUEST_TO_SEND

The **REQUEST_TO_SEND** verb notifies the partner transaction program that the local transaction program wants to send data.

VCB Structure

```
typedef struct request_to_send
{
 unsigned short     opcode;                  / verb operation code        /
 unsigned char      opext;                   / verb extension code        /
 unsigned short     primary_rc;              / primary return code        /
 unsigned long      secondary_rc;            / secondary return code      /
 unsigned char      tp_id[8];                / TP identifier              /
 unsigned long      conv_id;                 / conversation identifier /
} REQUEST_TO_SEND;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_B_REQUEST_TO_SEND

opext AP_BASIC_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameter:

primary_rc AP_OK

Returned Parameters: Conversation Ends

If **REQUEST_TO_SEND** is issued in nonblocking mode (see “Queue-level Nonblocking” on page 24), and the conversation ends while processing a verb on the send/receive queue, Personal Communications returns the following parameter:

primary_rc AP_CONVERSATION_ENDED

The application should not issue any more verbs for this conversation.

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_R_T_S_BAD_STATE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

SEND_CONVERSATION

The **SEND_CONVERSATION** verb allocates a conversation to a session between the local LU and partner LU (causing a transaction program on the partner LU to start), sends a single data record on this conversation, then deallocates the conversation without waiting for confirmation. It is equivalent to an **ALLOCATE**, **SEND_DATA**, **DEALLOCATE** (**FLUSH**) sequence of verbs (commonly termed a "single one-way bracket").

VCB Structure

```
typedef struct send_conversation
{
 unsigned short     opcode;                  / verb operation code                 /
 unsigned char      opext;                   / verb extension code                 /
 unsigned short     primary_rc               / primary return code                 /
 unsigned long      secondary_rc;            / secondary return code               /
 unsigned char      tp_id[8];                / TP identifier                       /
 unsigned char      reserv3[8];              / reserved                            /
 unsigned char      rtn_ctl;                 / return control                      /
 unsigned char      reserv4;                 / reserved                            /
 unsigned long      conv_group_id;           / conversation group identifier       /
 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];             / TP name                             /
 unsigned char      security;                / security                            /
 unsigned char      reserv5[11];             / reserved                            /
 unsigned char      pwd[1 ];                 / security password                   /
 unsigned char      user_id[1 ];             / security user_id                    /
 unsigned short     pip_dlen;                / PIP data length                     /
 unsigned char      pip_dptr;                / pointer to PIP data                 /
 unsigned char      reserv5a;                / reserved                            /
 unsigned char      fqplu_name[17];          / fully qualified partner LU          /
                                               / name                             /
 unsigned char      reserv6[8];              / reserved                            /
 unsigned short     dlen;                    / data length                         /
 unsigned char      dptr;                    / pointer to data buffer              /
} SEND_CONVERSATION;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_B_SEND_CONVERSATION

opext AP_BASIC_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb for an invoking transaction program, or by the **RECEIVE_ALLOCATE** verb for an invoked transaction program.

rtn_ctl Specifies when the local LU acting on a session request from the local transaction processor is to return control to the local transaction program.

AP_IMMEDIATE
AP_WHEN_SESSION_ALLOCATED
AP_WHEN_SESSION_FREE
AP_WHEN_CONV_GROUP_ALLOC
AP_WHEN_CONWINNER_ALLOC
AP_WHEN_CONLOSER_ALLOC

conv_group_id The conversation group identifier for the session to be allocated. This parameter is only supplied if **rtn_ctl** is set to AP_WHEN_CONV_GROUP_ALLOC.

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 field is set to all zeros, Personal Communications uses the **fqplu_name** field 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.

tp_name Name of the invoked transaction program. Personal Communications does not check the character set of this field. The value of **tp_name** specified by the **ALLOCATE** verb in the invoking transaction program must match the value of **tp_name** specified by the **RECEIVE_ALLOCATE** verb in the invoked transaction program.

security Specifies the information the partner LU requires in order to validate access to the invoked transaction program.

AP_NONE
AP_PGM
AP_SAME

pwd Password associated with **user_id**. This is a 10-byte type-AE EBCDIC character string, padded to the right with EBCDIC spaces. This is required if Security=Program (AP_PGM); otherwise, it is optional.

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. This is required if Security=Program (AP_PGM); otherwise, it is optional.

pip_dlen Length of the program initialization parameters (PIP) to be passed to the partner transaction program.

Range: 0–32767

pip_dptr Address of buffer containing PIP data. Use this parameter only if **pip_dlen** is greater than zero.

fqplu_name The fully qualified LU name for the partner LU. This name is 17 bytes long and is right-padded with EBCDIC blanks. 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 blanks. If the network ID is not present, then omit the dot.) This field is only significant if the **plu_alias** field is set to all zeros.

dlen Number of bytes of data to send.

Range: 0–65535

dptr Address of the buffer containing the data to send. The application can append data to the end of the VCB, in which case **dptr** must be set to NULL.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

conv_group_id
The conversation group identifier of the session allocated to the conversation.

Returned Parameters: No Session

If the **rtn_ctl** parameter was set to AP_IMMEDIATE, and no session is available immediately, Personal Communications returns the following parameters:

primary_rc AP_UNSUCCESSFUL

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc
AP_BAD_TP_ID
AP_BAD_LL
AP_BAD_RETURN_CONTROL
AP_BAD_SECURITY
AP_PIP_LEN_INCORRECT
AP_UNKNOWN_PARTNER_MODE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_UNSUCCESSFUL
AP_ALLOCATION_ERROR
  AP_ALLOCATION_FAILURE_NO_RETRY
  AP_ALLOCATION_FAILURE_RETRY
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

If the **primary_rc** is set to AP_ALLOCATION_ERROR, the **sense_data** field carries more information on the failure.

SEND_DATA

The **SEND_DATA** verb puts data in the local LU's send buffer for transmission to the partner transaction program.

VCB Structure

```
typedef struct send_data
{
 unsigned short      opcode;                 /  verb operation code          /
 unsigned char       opext;                  /  verb extension code          /
 unsigned short      primary_rc;             /  primary return code          /
 unsigned long       secondary_rc;           /  secondary return code        /
 unsigned char       tp_id[8];               /  TP identifier                 /
 unsigned long       conv_id;                /  conversation identifier      /
 unsigned char       rts_rcvd;               /  request to send received     /
 unsigned char       reserv3;                /  reserved                     /
 unsigned short      dlen;                   /  data length                  /
 unsigned char       dptr;                   /  pointer to data              /
 unsigned char       type;                   /  send data type               /
 unsigned char       reserv4;                /  reserved                     /
} SEND_DATA;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_B_SEND_DATA

opext AP_BASIC_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

dlen Number of bytes of data to be put in the local LU's send buffer.

Range: 0–65535

dptr Address of the buffer containing the data to be put in the local LU's send buffer. The application can append data to the end of the VCB, in which case **dptr** must be set to NULL.

type Specifies whether to perform the function of another verb in addition to **SEND_DATA**.

AP_NONE
AP_SEND_DATA_CONFIRM
AP_SEND_DATA_FLUSH
AP_SEND_DATA_P_TO_R_FLUSH
AP_SEND_DATA_P_TO_R_SYNC_LEVEL
AP_SEND_DATA_P_TO_R_CONFIRM
AP_SEND_DATA_DEALLOC_FLUSH

AP_SEND_DATA_DEALLOC_SYNC_LEVE
AP_SEND_DATA_DEALLOC_CONFIRM
AP_SEND_DATA_DEALLOC_ABEND

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

rts_rcvd Request-to-send-received indicator.

AP_YES
AP_NO

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID
AP_BAD_LL
AP_SEND_DATA_INVALID_TYPE
AP_SEND_DATA_CONFIRM_SYNC_NONE

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_SEND_DATA_NOT_SEND_STATE
AP_SEND_DATA_NOT_LL_BDY

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_ALLOCATION_ERROR

AP_SECURITY_NOT_VALID
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
AP_TP_NAME_NOT_RECOGNIZED
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_CONVERSATION_TYPE_MISMATCH
AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_PROG_ERROR_PURGING

AP_SVC_ERROR_PURGING
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

SEND_ERROR

The **SEND_ERROR** verb notifies the partner transaction program that the local transaction program has encountered an application-level error.

VCB Structure

```
typedef struct send_error
{
 unsigned short       opcode;                 /  verb operation code          /
 unsigned char        opext;                  /  verb extension code          /
 unsigned short       primary_rc;             /  primary return code          /
 unsigned long        secondary_rc;           /  secondary return code        /
 unsigned char        tp_id[8];               /  TP identifier                 /
 unsigned long        conv_id;                /  conversation identifier      /
 unsigned char        rts_rcvd;               /  request to send received     /
 unsigned char        err_type;               /  error type                   /
 unsigned char        err_dir;                /  error direction              /
 unsigned char        reserv3;                /  reserved                     /
 unsigned short       log_dlen;               /  log data length              /
 unsigned char        log_dptr;               /  pointer to log data          /
} SEND_ERROR;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_B_SEND_ERROR

opext AP_BASIC_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

err_type Indicates the type of the error being reported—application program or service program.

AP_PROG
AP_SVC

err_dir Indicates whether the error being reported is in the data received from the partner transaction program, or in the data the local transaction program was about to send.

This parameter is used only when the **SEND_ERROR** verb is being issued in SEND_PENDING state.

AP_RCV_DIR_ERROR
AP_SEND_DIR_ERROR

log_dlen Number of bytes of data to be sent to the error log file.

Range: 0–32767

The application can append data to the end of the VCB, in which case this field will be greater than zero and **log_dptr** must be set to NULL. (A length of zero indicates that there is no error log data.)

log_dptr Address of data buffer containing error information. The application can append data to the end of the VCB, in which case **log_dptr** must be set to NULL.

This data is sent to the local error log and to the partner LU. This parameter is used by the **SEND_ERROR** verb if **log_dlen** is greater than zero.

The transaction program must format the error data as a General Data Stream (GDS) error log variable. For further information, see *IBM Systems Network Architecture: LU 6.2 Reference: Peer Protocols*.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

rts_rcvd Request-to-send-received indicator.

AP_YES
AP_NO

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc
AP_BAD_CONV_ID
AP_BAD_ERROR_DIRECTION
AP_BAD_TP_ID
AP_SEND_ERROR_BAD_TYPE
AP_SEND_ERROR_LOG_LL_WRONG

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

Verb Issued in Any Permitted State: The following return codes can be generated when the **SEND_ERROR** verb is issued in any permitted state:

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

Verb Issued in SEND State: The following return codes can be generated only if the **SEND_ERROR** verb is issued in SEND state:

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED

AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_PROG_ERROR_PURGING
AP_SVC_ERROR_PURGING

Verb Issued in RECEIVE State: The following return code can be generated only if the verb is issued in RECEIVE state:

AP_DEALLOC_NORMAL

TEST_RTS

The **TEST_RTS** verb determines whether a request-to-send notification has been received from the partner transaction program.

VCB Structure

```
typedef struct test_rts
{
 unsigned short     opcode;          / verb operation code      /
 unsigned char      opext;           / verb extension code      /
 unsigned short     primary_rc;      / primary return code      /
 unsigned long      secondary_rc;    / secondary return code    /
 unsigned char      tp_id[8];        / TP identifier            /
 unsigned long      conv_id;         / conversation identifier  /
 unsigned char      reserv3;         / reserved                 /
} TEST_RTS;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_B_TEST_RTS

opext AP_BASIC_CONVERSATION

tp_id Identifier for the local transaction program. The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameter:

primary_rc Indicates whether a request-to-send notification has been received from the partner transaction program.

AP_OK
AP_UNSUCCESSFUL

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR

TEST_RTS_AND_POST

The **TEST_RTS_AND_POST** verb asynchronously determines whether a request-to-send notification has been received from the partner transaction program. A transaction program can issue a **TEST_RTS_AND_POST** at any time, even when there is another verb outstanding on the conversation. **TEST_RTS_AND_POST** returns when a request-to-send notification is received, or when the conversation ends, or when a conversation failure is detected.

VCB Structure

```
typedef struct test_rts_and_post
{
 unsigned short        opcode;          /  verb operation code         /
 unsigned char         opext;           /  verb extension code         /
 unsigned short        primary_rc;      /  primary return code         /
 unsigned long         secondary_rc;    /  secondary return code       /
 unsigned char         tp_id[8];        /  TP identifier               /
 unsigned long         conv_id;         /  conversation identifier     /
 unsigned char         reserv3;         /  reserved                    /
 unsigned long         sema;            /  post handle for verb        /
} TEST_RTS_AND_POST;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_B_TEST_RTS_AND_POST

opext AP_BASIC_CONVERSATION

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

sema The API can use this field as it requires. Personal Communications returns the same value in the verb reply. Typically the API stores some posting information in this field, as for example, a callback address or a semaphore handle.

Returned Parameters: Successful Execution

If the verb executes successfully (that is, a request-to-send notification is received), Personal Communications returns the following parameter:

primary_rc AP_OK

Returned Parameters: Conversation Ended

If the verb returns because the conversation has ended or a conversation failure has been detected, Personal Communications returns the following parameter:

primary_rc AP_UNSUCCESSFUL

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

Mapped Conversation Verb Control Blocks

.

MC_ALLOCATE

The **MC_ALLOCATE** verb is issued by the invoking transaction program. This verb allocates a session between the local LU and partner LU and (in conjunction with the **RECEIVE_ALLOCATE** verb) establishes a conversation between the invoking transaction program and the invoked transaction program.

Upon successful execution of this verb, Personal Communications generates a conversation identifier (**conv_id**). This identifier is a required parameter for all other conversation verbs.

VCB Structure

```
typedef struct mc_allocate
{
 unsigned short     opcode;                  / verb operation code                  /
 unsigned char      opext;                   / verb extension code                  /
 unsigned short     primary_rc;              / primary return code                  /
 unsigned long      secondary_rc;            / secondary return code                /
 unsigned char      tp_id[8];                / TP identifier                        /
 unsigned long      conv_id;                 / conversation identifier              /
 unsigned char      reserv3;                 / reserved                             /
 unsigned char      sync_level;              / sync level                           /
 unsigned char      reserv4[2];              / reserved                             /
 unsigned char      rtn_ctl;                 / return control                       /
 unsigned char      reserv4a[1];             / reserved                             /
 unsigned long      conv_group_id;           / conversation group identifier /
 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];             / partner TP name                      /
 unsigned char      security;                / security level                       /
 unsigned char      reserv6[11];             / reserved                             /
 unsigned char      pwd[1 ];                 / security password                    /
 unsigned char      user_id[1 ];             / security user_id                     /
 unsigned short     pip_dlen;                / PIP data length                      /
 unsigned char      pip_dptr;                / pointer to PIP data                  /
 unsigned char      reserv6a;                / reserved                             /
 unsigned char      fqplu_name[17];          / fully qualified partner LU           /
                                                  / name                                  /
 unsigned char      reserv7[8];              / reserved                             /
} MC_ALLOCATE;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_ALLOCATE

opext AP_MAPPED_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb for an invoking transaction program, or by the **RECEIVE_ALLOCATE** verb for an invoked transaction program.

sync_level Synchronization level of the conversation.

AP_CONFIRM_SYNC_LEVEL
AP_NONE

rtn_ctl Specifies when the local LU acting on a session request from the local transaction program is to return control to the local transaction program.

AP_IMMEDIATE
AP_WHEN_SESSION_ALLOCATED
AP_WHEN_SESSION_FREE
AP_WHEN_CONV_GROUP_ALLOC
AP_WHEN_CONWINNER_ALLOC
AP_WHEN_CONLOSER_ALLOC

conv_group_id Conversation group identifier for the session to be allocated. This parameter is only supplied if **rtn_ctl** is set to AP_WHEN_CONV_GROUP_ALLOC.

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 field is set to all zeros, Personal Communications uses the **fqplu_name** field 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.

tp_name Name of the invoked transaction program. Personal Communications does not check the character set of this field. The value of **tp_name** specified by the **MC_ALLOCATE** verb in the invoking transaction program must match the value of **tp_name** specified by the **RECEIVE_ALLOCATE** verb in the invoked transaction program.

security Specifies the information the partner LU requires in order to validate access to the invoked transaction program.

AP_NONE
AP_PGM
AP_SAME

pwd Password associated with **user_id**. This is a 10-byte type-AE EBCDIC character string, padded to the right with EBCDIC spaces. This is required if Security=Program (AP_PGM); otherwise, it is optional.

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. This is required if Security=Program (AP_PGM); otherwise, it is optional.

pip_dlen Length of the program initialization parameters (PIP) to be passed to the partner transaction program.

Range: 0–32767

pip_dptr Address of buffer containing PIP data. Use this parameter only if **pip_dlen** is greater than zero.

fqplu_name Fully qualified LU name for the partner LU. This name is 17 bytes long and is right-padded with EBCDIC blanks. 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 blanks. If the network ID is not present, then omit the dot.) This field is only significant if the **plu_alias** field is set to all zeros.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

conv_id Conversation identifier.

This value identifies the conversation established between the two transaction programs.

conv_group_id Conversation group identifier of the session allocated to the conversation.

Returned Parameters: No Session

If the **rtn_ctl** parameter was set to AP_IMMEDIATE, and no session is available immediately, Personal Communications returns the following parameters:

primary_rc AP_UNSUCCESSFUL

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_RETURN_CONTROL
AP_BAD_SECURITY
AP_BAD_SYNC_LEVEL
AP_BAD_TP_ID
AP_NO_USE_OF_SNASVCMG_CPSVCMG
AP_PIP_LEN_INCORRECT
AP_UNKNOWN_PARTNER_MODE

Returned Parameters: Other Conditions

sense_data Provides additional information on the reason why the **MC_ALLOCATE** failed.

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, "APPC Common Return Codes."

AP_ALLOCATION_ERROR

- AP_ALLOCATION_FAILURE_NO_RETRY
- AP_ALLOCATION_FAILURE_RETRY

AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

If the **primary_rc** is set to AP_ALLOCATION_ERROR, the **sense_data** field carries more information on the failure.

MC_CONFIRM

The **MC_CONFIRM** verb sends the contents of the local LU's send buffer and a confirmation request to the partner transaction program.

In response to the **MC_CONFIRM** verb, the partner transaction program normally issues the **MC_CONFIRMED** verb to confirm that it has received the data without error. (If the partner transaction program encounters an error, it issues the **MC_SEND_ERROR** verb or abnormally deallocates the conversation.)

The transaction program can issue the **MC_CONFIRM** verb only if the conversation's synchronization level, established by the **MC_ALLOCATE** verb, is AP_CONFIRM_SYNC_LEVEL.

VCB Structure

```
typedef struct mc_confirm
{
 unsigned short        opcode;              /  verb operation code          /
 unsigned char         opext;               /  verb extension code          /
 unsigned short        primary_rc;          /  primary return code          /
 unsigned long         secondary_rc;        /  secondary return code        /
 unsigned char         tp_id[8];            /  TP identifier                /
 unsigned long         conv_id;             /  conversation identifier      /
 unsigned char         rts_rcvd;            /  request to send received  /
} MC_CONFIRM;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_CONFIRM

opext AP_MAPPED_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **MC_ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

rts_rcvd Request-to-send-received indicator.

AP_YES
AP_NO

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID
AP_CONFIRM_ON_SYNC_LEVEL_NONE

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_CONFIRM_BAD_STATE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND
AP_PROG_ERROR_PURGING
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

MC_CONFIRMED

The **MC_CONFIRMED** verb replies to a confirmation request from the partner transaction program. It informs the partner transaction program that the local transaction program has not detected an error in the received data.

Because the transaction program issuing the confirmation request waits for a confirmation, the **MC_CONFIRMED** verb synchronizes the processing of the two transaction programs.

VCB Structure

```
typedef struct mc_confirmed
{
 unsigned short        opcode;                 /  verb operation code         /
 unsigned char         opext;                  /  verb extension code         /
 unsigned short        primary_rc;             /  primary return code         /
 unsigned long         secondary_rc;           /  secondary return code       /
 unsigned char         tp_id[8];               /  TP identifier               /
 unsigned long         conv_id;                /  conversation identifier  /
} MC_CONFIRMED;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_CONFIRMED

opext AP_MAPPED_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking TP or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **MC_ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameter:

primary_rc AP_OK

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_CONFIRMED_BAD_STATE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR

MC_DEALLOCATE

The **MC_DEALLOCATE** verb deallocates a conversation between two transaction programs.

Before deallocating the conversation, this verb performs the equivalent of one of the following verbs:

The **MC_FLUSH** verb, which sends the contents of the local LU's send buffer to the partner LU (and transaction program).

The **MC_CONFIRM** verb, which sends the contents of the local LU's send buffer and a confirmation request to the partner transaction program.

After this verb has successfully executed, the conversation ID is no longer valid.

VCB Structure

```
typedef struct mc_deallocate
{
 unsigned short     opcode;                  / verb operation code        /
 unsigned char      opext;                   / verb extension code        /
 unsigned char      format;                  / format                     /
 unsigned short     primary_rc;              / primary return code        /
 unsigned long      secondary_rc;            / secondary return code      /
 unsigned char      tp_id[8];                / TP identifier              /
 unsigned long      conv_id;                 / conversation identifier /
 unsigned char      dealloc_type;            / deallocate type            /
 unsigned char      reserv4[2];              / reserved                   /
 unsigned char      reserv5[4];              / reserved                   /
} MC_DEALLOCATE;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_DEALLOCATE

opext AP_MAPPED_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program. The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier. The value of this parameter was returned by the **MC_ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

dealloc_type Specifies how to perform the deallocation.

AP_ABEND
AP_ABEND_SVC
AP_ABEND_TIMER
AP_FLUSH
AP_SYNC_LEVEL
AP_TP_NOT_AVAIL_NO_RETRY
AP_TP_NOT_AVAIL_RETRY
AP_TPN_NOT_RECOGNIZED
AP_PIP_DATA_NOT_ALLOWED

AP_PIP_DATA_INCORRECT
AP_RESOURCE_FAILURE_NO_RETRY
AP_CONV_TYPE_MISMATCH
AP_SYNC_LVL_NOT_SUPPORTED
AP_SECURITY_PARAMS_INVALID

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameter:

primary_rc AP_OK

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID
AP_DEALLOC_BAD_TYPE

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_DEALLOC_CONFIRM_BAD_STATE
AP_DEALLOC_FLUSH_BAD_STATE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND
AP_PROG_ERROR_PURGING
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

MC_FLUSH

The **MC_FLUSH** verb sends the contents of the local LU's send buffer to the partner LU (and transaction program). If the send buffer is empty, no action takes place.

VCB Structure

```
typedef struct mc_flush
{
 unsigned short      opcode;          /  verb operation code       /
 unsigned char       opext;           /  verb extension code       /
 unsigned short      primary_rc;      /  primary return code       /
 unsigned long       secondary_rc;    /  secondary return code     /
 unsigned char       tp_id[8];        /  TP identifier             /
 unsigned long       conv_id;         /  conversation identifier   /
} MC_FLUSH;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_FLUSH

opext AP_MAPPED_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **MC_ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameter:

primary_rc AP_OK

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_FLUSH_NOT_SEND_STATE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR

MC_GET_ATTRIBUTES

The **MC_GET_ATTRIBUTES** verb returns the attributes of the conversation.

VCB Structure

```
typedef struct mc_get_attributes
{
 unsigned short     opcode;                  / verb operation code                  /
 unsigned char      opext;                   / verb extension code                 /
 unsigned short     primary_rc;              / primary return code                 /
 unsigned long      secondary_rc;            / secondary return code               /
 unsigned char      tp_id[8];                / TP identifier                       /
 unsigned long      conv_id;                 / conversation identifier             /
 unsigned char      reserv3;                 / reserved                            /
 unsigned char      sync_level;              / sync_level                          /
 unsigned char      mode_name[8];            / mode name                           /
 unsigned char      net_name[8];             / network name of local LU            /
 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      plu_un_name[8];          / partner LU uninterpreted name       /
 unsigned char      reserv4[2];              / reserved                            /
 unsigned char      fqplu_name[17];          / fully qualified partner LU          /
                                                 / name                             /
 unsigned char      reserv5;                 / reserved                            /
 unsigned char      user_id[1 ];             / user identifier                     /
 unsigned long      conv_group_id;           / conversation group identifier       /
 unsigned char      conv_corr_len;           / conversation correlator             /
                                                 / length                           /
 unsigned char      conv_corr[8];            / conversation correlator             /
 unsigned char      reserv6[13];             / reserved                            /
} MC_GET_ATTRIBUTES;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_GET_ATTRIBUTES

opext AP_MAPPED_CONVERSATION

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **MC_ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

sync_level Synchronization level of the conversation.

AP_CONFIRM_SYNC_LEVEL
AP_NONE

mode_name Name of the set of networking characteristics associated with the session allocated to the conversation. This is an 8-byte alphanumeric type-A EBCDIC string (starting with a letter), padded to the right with EBCDIC spaces.

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

lu_alias Alias by which the local 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.

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.

plu_un_name Uninterpreted name of partner LU, that is, the name of the partner LU as defined at the system services control point (SSCP). This is an 8-byte type-A EBCDIC character string.

fqplu_name Fully qualified name of the partner LU. This name is 17 bytes long and is right-padded with EBCDIC blanks. 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 blanks. If the network ID is not present, then omit the dot.)

user_id User ID sent by the invoking transaction program through the **ALLOCATE** verb to access the invoked transaction program (if applicable). This is a 10-byte type-AE EBCDIC character string, padded to the right with EBCDIC spaces.

conv_group_id Conversation group identifier of the session allocated to the conversation.

conv_corr_len Always set to 0.

Range: 0–8

conv_corr Always set to 0.

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR

MC_PREPARE_TO_RECEIVE

The **MC_PREPARE_TO_RECEIVE** verb changes the state of the conversation for the local transaction processor from SEND or SEND_PENDING to RECEIVE.

Before changing the conversation state, this verb performs one of the following actions:

The verb sends the contents of the local LU's send buffer to the partner LU and transaction program (equivalent to sending the **MC_FLUSH** verb).

The verb sends the contents of the local LU's send buffer and a confirmation request to the partner transaction program (equivalent to sending the **MC_CONFIRM** verb).

After this verb has successfully executed, the local transaction program can receive data.

VCB Structure

```
typedef struct mc_prepare_to_receive
{
 unsigned short        opcode;                 /  verb operation code          /
 unsigned char         opext;                  /  verb extension code          /
 unsigned short        primary_rc;             /  primary return code          /
 unsigned long         secondary_rc;           /  secondary return code        /
 unsigned char         tp_id[8];               /  TP identifier                 /
 unsigned long         conv_id;                /  conversation identifier      /
 unsigned char         ptr_type;               /  prepare to receive type      /
 unsigned char         locks;                  /  prepare to receive locks     /
} MC_PREPARE_TO_RECEIVE;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_PREPARE_TO_RECEIVE

opext AP_MAPPED_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **MC_ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

ptr_type Specifies how to perform the state change.

AP_FLUSH
AP_SYNC_LEVEL
TP_P_TO_R_CONFIRM

locks Specifies when Personal Communications is to return control to the local transaction program.

AP_LONG

AP_SHORT

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameter:

primary_rc AP_OK

Returned Parameters: Parameter Check

If the verb does not execute because of a parameter error, Personal Communications returns the following parameters

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID
AP_P_TO_R_INVALID_TYPE

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_P_TO_R_NOT_SEND_STATE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes”:

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND
AP_PROG_ERROR_PURGING
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

MC_RECEIVE_AND_POST

The **MC_RECEIVE_AND_POST** verb receives application data and status information asynchronously. This enables the transaction program to proceed with processing while data is still arriving at the local LU.

VCB Structure

```
typedef struct mc_receive_and_post
{
 unsigned short     opcode;              / verb operation code            /
 unsigned char      opext;               / verb extension code            /
 unsigned short     primary_rc;          / primary return code            /
 unsigned long      secondary_rc;        / secondary return code          /
 unsigned char      tp_id[8];            / TP identifier                  /
 unsigned long      conv_id;             / conversation identifier        /
 unsigned short     what_rcvd;           / what received                  /
 unsigned char      rtn_status;          / return status with data        /
 unsigned char      reserv4;             / reserved                       /
 unsigned char      rts_rcvd;            / request to send received       /
 unsigned char      reserv5;             / reserved                       /
 unsigned short     max_len;             / maximum length of received     /
                                           / data                         /
 unsigned short     dlen;                / actual length of received      /
                                           / data                         /
 unsigned char      dptr;                / pointer to data buffer         /
 unsigned long      sema;                / post handle for verb           /
 unsigned char      reserv6;             / reserved                       /
} MC_RECEIVE_AND_POST;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_RECEIVE_AND_POST

opext AP_MAPPED_CONVERSATION

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **MC_ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

rtn_status Indicates whether status information and data can be returned on the same verb.

AP_YES
AP_NO

max_len Maximum number of bytes of data the local transaction program can receive.

Range: 0–65535

This value must not exceed the length of the buffer to contain the received data.

dptr Address of the buffer to contain the data received by the local LU. The application can append data to the end of the VCB in which case **dptr** must be set to NULL.

sema The API may use this field as it requires. Personal Communications returns the same value in the verb reply. Typically the API stores some posting information in this field, as for example, a callback address or a semaphore handle.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc
AP_OK
AP_DEALLOC_NORMAL

what_rcvd Status information received with the incoming data. If **rtn_status** is set to AP_NO, this field always contains a value from the first part of the following list. If **rtn_status** is set to AP_YES, this field may contain any value from the list.

AP_NONE
AP_CONFIRM_DEALLOCATE
AP_CONFIRM_SEND
AP_CONFIRM_WHAT_RECEIVED
AP_SEND
AP_DATA_COMPLETE
AP_DATA_INCOMPLETE
AP_USER_CONTROL_DATA_COMPLETE
AP_USER_CONTROL_DATA_INCMP
AP_PS_HEADER_COMPLETE
AP_PS_HEADER_INCOMPLETE
AP_DATA_COMPLETE_CONFIRM
AP_DATA_COMPLETE_CONFIRM_DEALL
AP_DATA_COMPLETE_CONFIRM_SEND
AP_DATA_COMPLETE_SEND
AP_UC_DATA_COMPLETE_CONFIRM
AP_UC_DATA_COMPLETE_CNFM_DEALL
AP_UC_DATA_COMPLETE_CNFM_SEND
AP_UC_DATA_COMPLETE_SEND
AP_PS_HDR_COMPLETE_CONFIRM
AP_PS_HDR_COMPLETE_CNFM_DEALL
AP_PS_HDR_COMPLETE_CNFM_SEND
AP_PS_HDR_COMPLETE_SEND

rts_rcvd Request-to-send-received indicator.

AP_YES
AP_NO

dlen This parameter is only used if the **what_rcvd** parameter indicates that data was received. Number of bytes of data received (the data is stored in the buffer specified by the **dptr** parameter). A length of zero indicates that no data was received.

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_RETURN_STATUS_WITH_DATA
AP_BAD_TP_ID

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_RCV_AND_POST_BAD_STATE

Returned Parameters: Verb Cancelled

If the verb did not execute because it was canceled by another verb issued by the transaction program, Personal Communications returns the following parameter:

primary_rc AP_CANCELLED

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND
AP_DEALLOC_NORMAL
AP_PROG_ERROR_NO_TRUNC
AP_PROG_ERROR_PURGING
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

MC_RECEIVE_AND_WAIT

The **MC_RECEIVE_AND_WAIT** verb receives any data that is currently available from the partner transaction program. If no data is currently available, the local transaction program waits for data to arrive.

VCB Structure

```
typedef struct mc_receive_and_wait
{
 unsigned short      opcode;          / verb operation code          /
 unsigned char       opext;           / verb extension code          /
 unsigned short      primary_rc;      / primary return code          /
 unsigned long       secondary_rc;    / secondary return code        /
 unsigned char       tp_id[8];        / TP identifier                /
 unsigned long       conv_id;         / conversation identifier      /
 unsigned short      what_rcvd;       / what received                /
 unsigned char       rtn_status;      / return status with data      /
 unsigned char       reserv4;         / reserved                     /
 unsigned char       rts_rcvd;        / request to send received     /
 unsigned char       reserv5;         / reserved                     /
 unsigned short      max_len;         / maximum length of received /
                                        / data                        /
 unsigned short      dlen;            / actual length of received   /
                                        / data                        /
 unsigned char       dptr;            / pointer to data buffer      /
 unsigned char       reserv6[5];      / reserved                    /
} MC_RECEIVE_AND_WAIT;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_RECEIVE_AND_WAIT

opext AP_MAPPED_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **MC_ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

rtn_status Indicates whether status information and data can be returned on the same verb.

AP_YES
AP_NO

max_len Maximum number of bytes of data the local transaction program can receive.

Range: 0–65535

This value must not exceed the length of the buffer to contain the received data.

dptr Address of the buffer to contain the data received by the local LU. The application can append data to the end of the VCB in which case **dptr** must be set to NULL.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc
AP_OK
AP_DEALLOC_NORMAL

what_rcvd Status information received with the incoming data. If **rtn_status** is set to AP_NO, this field always contains a value from the first part of the following list. If **rtn_status** is set to AP_YES, this field can contain any value from the list.

AP_NONE
AP_CONFIRM_DEALLOCATE
AP_CONFIRM_SEND
AP_CONFIRM_WHAT_RECEIVED
AP_SEND
AP_DATA_COMPLETE
AP_DATA_INCOMPLETE
AP_USER_CONTROL_DATA_COMPLETE
AP_USER_CONTROL_DATA_INCMP
AP_PS_HEADER_COMPLETE
AP_PS_HEADER_INCOMPLETE

AP_DATA_COMPLETE_CONFIRM
AP_DATA_COMPLETE_CONFIRM_DEALL
AP_DATA_COMPLETE_CONFIRM_SEND
AP_DATA_COMPLETE_SEND
AP_UC_DATA_COMPLETE_CONFIRM
AP_UC_DATA_COMPLETE_CNFM_DEALL
AP_UC_DATA_COMPLETE_CNFM_SEND
AP_UC_DATA_COMPLETE_SEND
AP_PS_HDR_COMPLETE_CONFIRM
AP_PS_HDR_COMPLETE_CNFM_DEALL
AP_PS_HDR_COMPLETE_CNFM_SEND
AP_PS_HDR_COMPLETE_SEND

rts_rcvd Request-to-send-received indicator.

AP_YES
AP_NO

dlen This parameter is only used if the **what_rcvd** parameter indicates that data was received. Number of bytes of data received (the data is stored in the buffer specified by the **dptr** parameter). A length of zero indicates that no data was received.

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_RETURN_STATUS_WITH_DATA
AP_BAD_TP_ID

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_RCV_AND_WAIT_BAD_STATE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND
AP_DEALLOC_NORMAL
AP_PROG_ERROR_NO_TRUNC
AP_PROG_ERROR_PURGING
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

MC_RECEIVE_IMMEDIATE

The **MC_RECEIVE_IMMEDIATE** verb receives any data or status information that is currently available from the partner transaction processor. If none is currently available, the local transaction program returns immediately and does not wait.

VCB Structure

```
typedef struct mc_receive_immediate
{
 unsigned short      opcode;          / verb operation code          /
 unsigned char       opext;           / verb extension code          /
 unsigned short      primary_rc;      / primary return code          /
 unsigned long       secondary_rc;    / secondary return code        /
 unsigned char       tp_id[8];        / TP identifier                /
 unsigned long       conv_id;         / conversation identifier      /
 unsigned short      what_rcvd;       / what received                /
 unsigned char       rtn_status;      / return status with data      /
 unsigned char       reserv4;         / reserved                     /
 unsigned char       rts_rcvd;        / request to send received     /
 unsigned char       reserv5;         / reserved                     /
 unsigned short      max_len;         / maximum length of received   /
                                        / data                       /
 unsigned short      dlen;            / actual length of received    /
                                        / data                       /
 unsigned char       dptr;            / pointer to data buffer       /
 unsigned char       reserv6[5];      / reserved                     /
} MC_RECEIVE_IMMEDIATE;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_RECEIVE_IMMEDIATE

opext AP_MAPPED_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **MC_ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

rtn_status Indicates whether status information and data can be returned on the same verb.

AP_YES
AP_NO

max_len Maximum number of bytes of data the local transaction program can receive.

Range: 0–65535

This value must not exceed the length of the buffer to contain the received data.

dptr Address of the buffer to contain the data received by the local LU. The application can append data to the end of the VCB in which case **dptr** must be set to NULL.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc
AP_OK
AP_DEALLOC_NORMAL

what_rcvd Status information received with the incoming data. If **rtn_status** is set to AP_NO, this field always contains a value from the first part of the following list. If **rtn_status** is set to AP_YES, this field can contain any value from the list.

AP_NONE
AP_CONFIRM_DEALLOCATE
AP_CONFIRM_SEND
AP_CONFIRM_WHAT_RECEIVED
AP_SEND
AP_DATA_COMPLETE
AP_DATA_INCOMPLETE
AP_USER_CONTROL_DATA_COMPLETE
AP_USER_CONTROL_DATA_INCMP
AP_PS_HEADER_COMPLETE
AP_PS_HEADER_INCOMPLETE

AP_DATA_COMPLETE_CONFIRM
AP_DATA_COMPLETE_CONFIRM_DEALL
AP_DATA_COMPLETE_CONFIRM_SEND
AP_DATA_COMPLETE_SEND
AP_UC_DATA_COMPLETE_CONFIRM
AP_UC_DATA_COMPLETE_CNFM_DEALL
AP_UC_DATA_COMPLETE_CNFM_SEND
AP_UC_DATA_COMPLETE_SEND
AP_PS_HDR_COMPLETE_CONFIRM
AP_PS_HDR_COMPLETE_CNFM_DEALL
AP_PS_HDR_COMPLETE_CNFM_SEND
AP_PS_HDR_COMPLETE_SEND

rts_rcvd Request-to-send-received indicator.

AP_YES
AP_NO

dlen This parameter is only used if the **what_rcvd** parameter indicates that data was received. Number of bytes of data received (the data is stored in the buffer specified by the **dptr** parameter). A length of zero indicates that no data was received.

Returned Parameters: No Data Available

If no data is immediately available from the partner transaction program, Personal Communications returns the following parameter.

primary_rc AP_UNSUCCESSFUL

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_RETURN_STATUS_WITH_DATA
AP_BAD_TP_ID

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_RCV_IMMD_BAD_STATE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND
AP_DEALLOC_NORMAL
AP_PROG_ERROR_NO_TRUNC
AP_PROG_ERROR_PURGING
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

MC_REQUEST_TO_SEND

The **MC_REQUEST_TO_SEND** verb notifies the partner transaction program that the local transaction program wants to send data.

VCB Structure

```
typedef struct mc_request_to_send
{
 unsigned short        opcode;              /  verb operation code        /
 unsigned char         opext;               /  verb extension code        /
 unsigned short        primary_rc;          /  primary return code        /
 unsigned long         secondary_rc;        /  secondary return code      /
 unsigned char         tp_id[8];            /  TP identifier              /
 unsigned long         conv_id;             /  conversation identifier  /
} MC_REQUEST_TO_SEND;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_REQUEST_TO_SEND

opext AP_MAPPED_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **MC_ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameter:

primary_rc AP_OK

Returned Parameters: Conversation Ends

If **MC_REQUEST_TO_SEND** is issued in nonblocking mode (see “Queue-level Nonblocking” on page 24), and the conversation ends while processing a verb on the send/receive queue, Personal Communications returns the following parameter:

primary_rc AP_CONVERSATION_ENDED

The application should not issue any more verbs for this conversation.

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_R_T_S_BAD_STATE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, "APPC Common Return Codes."

AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

MC_SEND_CONVERSATION

The **MC_SEND_CONVERSATION** verb allocates a conversation to a session between the local LU and partner LU (causing a transaction program on the partner LU to start), sends a single data record on this conversation, then deallocates the conversation without waiting for confirmation. It is equivalent to an **MC_ALLOCATE**, **MC_SEND_DATA**, **MC_DEALLOCATE** (**FLUSH**) sequence of verbs (commonly termed a "single one-way bracket").

VCB Structure

```
typedef struct mc_send_conversation
{
 unsigned short     opcode;                  / verb operation code                  /
 unsigned char      opext;                   / verb extension code                  /
 unsigned char      format;                  / format                                /
 unsigned short     primary_rc;              / primary return code                  /
 unsigned long      secondary_rc;            / secondary return code                /
 unsigned char      tp_id[8];                / TP identifier                        /
 unsigned char      reserv3[8];              / reserved                              /
 unsigned char      rtn_ctl;                 / return control                       /
 unsigned char      reserv4;                 / reserved                              /
 unsigned long      conv_group_id;           / conversation group identifier /
 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];             / TP name                                 /
 unsigned char      security;                / security                              /
 unsigned char      reserv6[11];             / reserved                              /
 unsigned char      pwd[1 ];                 / security password                     /
 unsigned char      user_id[1 ];             / security user_id                     /
 unsigned short     pip_dlen;                / PIP data length                      /
 unsigned char      pip_dptr;                / pointer to PIP data                  /
 unsigned char      reserv6a;                / reserved                              /
 unsigned char      fqplu_name[17];          / fully qualified partner LU        /
                                                  / name                                  /
 unsigned char      reserv7[8];              / reserved                              /
 unsigned short     dlen;                    / data length                           /
 unsigned char      dptr;                    / pointer to data buffer               /
} MC_SEND_CONVERSATION;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_SEND_CONVERSATION

opext AP_MAPPED_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb for an invoking transaction program, or by the **RECEIVE_ALLOCATE** verb for an invoked transaction program.

rtn_ctl Specifies when the local LU acting on a session request from the local transaction program is to return control to the local transaction program.

AP_IMMEDIATE
AP_WHEN_SESSION_ALLOCATED
AP_WHEN_SESSION_FREE
AP_WHEN_CONV_GROUP_ALLOC
AP_WHEN_CONWINNER_ALLOC
AP_WHEN_CONLOSER_ALLOC

conv_group_id Conversation group identifier for the session to be allocated. This parameter is only supplied if **rtn_ctl** is set to AP_WHEN_CONV_GROUP_ALLOC.

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 field is set to all zeros, Personal Communications uses the **fqplu_name** field 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.

tp_name Name of the invoked transaction program. Personal Communications does not check the character set of this field. The value of **tp_name** specified by the **ALLOCATE** verb in the invoking transaction program must match the value of **tp_name** specified by the **RECEIVE_ALLOCATE** verb in the invoked transaction program.

security Specifies the information the partner LU requires in order to validate access to the invoked transaction program.

AP_NONE
AP_PGM
AP_SAME

pwd Password associated with **user_id**. This is a 10-byte type-AE EBCDIC character string, padded to the right with EBCDIC spaces. This is required if Security=Program (AP_PGM); otherwise, it is optional.

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. This is required if Security=Program (AP_PGM); otherwise, it is optional.

pip_dlen Length of the program initialization parameters (PIP) to be passed to the partner transaction program.

Range: 0–32767

pip_dptr Address of buffer containing PIP data. Use this parameter only if **pip_dlen** is greater than zero.

fqplu_name Fully qualified LU name for the partner LU. This name is 17 bytes long and is right-padded with EBCDIC blanks. 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 blanks. If the network ID is not present, then omit the dot.) This field is only significant of the **plu_alias** field is set to all zeros.

dlen Number of bytes of data to send.

Range: 0–65535

dptr Address of the buffer containing the data to send. The application can append data to the end of the VCB, in which case **dptr** must be set to NULL.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

conv_group_id Conversation group identifier of the session allocated to the conversation.

Returned Parameters: No Session

If the **rtn_ctl** parameter was set to AP_IMMEDIATE, and no session is available immediately, Personal Communications returns the following parameters:

primary_rc AP_UNSUCCESSFUL

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_TP_ID
AP_BAD_RETURN_CONTROL
AP_BAD_SECURITY
AP_PIP_LEN_INCORRECT
AP_NO_USE_OF_SNASVCMG

AP_UNKNOWN_PARTNER_MODE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_ALLOCATION_ERROR

- AP_ALLOCATION_FAILURE_NO_RETRY
- AP_ALLOCATION_FAILURE_RETRY

AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

If the **primary_rc** is set to AP_ALLOCATION_ERROR, the **sense_data** field carries more information on the failure.

MC_SEND_DATA

The **MC_SEND_DATA** verb puts data in the local LU's send buffer for transmission to the partner transaction program.

VCB Structure

```
typedef struct mc_send_data
{
 unsigned short      opcode;              /  verb operation code          /
 unsigned char       opext;               /  verb extension code          /
 unsigned short      primary_rc;          /  primary return code          /
 unsigned long       secondary_rc;        /  secondary return code        /
 unsigned char       tp_id[8];            /  TP identifier                /
 unsigned long       conv_id;             /  conversation identifier      /
 unsigned char       rts_rcvd;            /  request to send received     /
 unsigned char       data_type;           /  data type                    /
 unsigned short      dlen;                /  data length                  /
 unsigned char       dptr;                /  pointer to data              /
 unsigned char       type;                /  send data type               /
 unsigned char       reserv4;             /  reserved                     /
} MC_SEND_DATA;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_SEND_DATA

opext AP_MAPPED_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **MC_ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

data_type Specifies the type of the data to send. This can be set to one of the following parameters:

AP_APPLICATION
AP_USER_CONTROL_DATA
AP_PS_HEADER

dlen Number of bytes of data to be put in the local LU's send buffer.

Range: 0–65535

dptr Address of the buffer containing the data to be put in the local LU's send buffer. The application can append data to the end of the VCB in which case **dptr** must be set to NULL.

type Specifies whether to perform the function of another APPC verb in addition to SEND_DATA.

AP_NONE

AP_SEND_DATA_CONFIRM
AP_SEND_DATA_FLUSH
AP_SEND_DATA_P_TO_R_FLUSH
AP_SEND_DATA_P_TO_R_SYNC_LEVEL
AP_SEND_DATA_P_TO_R_CONFIRM
AP_SEND_DATA_DEALLOC_FLUSH
AP_SEND_DATA_DEALLOC_SYNC_LEVE
AP_SEND_DATA_DEALLOC_CONFIRM
AP_SEND_DATA_DEALLOC_ABEND

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

rts_rcvd Request-to-send-received indicator.

AP_YES
AP_NO

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID
AP_SEND_DATA_INVALID_TYPE
AP_SEND_DATA_CONFIRM_SYNC_NONE

Returned Parameters: State Check

If the conversation is in the wrong state when the transaction program issues this verb, Personal Communications returns the following parameters:

primary_rc AP_STATE_CHECK

secondary_rc AP_SEND_DATA_NOT_SEND_STATE

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_UNEXPECTED_SYSTEM_ERROR
AP_ALLOCATION_ERROR

AP_SECURITY_NOT_VALID
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
AP_TP_NAME_NOT_RECOGNIZED
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_CONVERSATION_TYPE_MISMATCH
AP_SYNC_LEVEL_NOT_SUPPORTED

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_DEALLOC_ABEND
AP_PROG_ERROR_PURGING
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

MC_SEND_ERROR

The **MC_SEND_ERROR** verb notifies the partner transaction program that the local transaction program has encountered an application-level error.

VCB Structure

```
typedef struct mc_send_error
{
 unsigned short     opcode;                 /  verb operation code          /
 unsigned char      opext;                  /  verb extension code          /
 unsigned short     primary_rc;             /  primary return code          /
 unsigned long      secondary_rc;           /  secondary return code        /
 unsigned char      tp_id[8];               /  TP identifier                /
 unsigned long      conv_id;                /  conversation identifier      /
 unsigned char      rts_rcvd;               /  request to send received     /
 unsigned char      err_type;               /  error type                   /
 unsigned char      err_dir;                /  error direction              /
 unsigned char      reserv4;                /  reserved                     /
 unsigned char      reserv5[2];             /  reserved                     /
 unsigned char      reserv6[4];             /  reserved                     /
} MC_SEND_ERROR;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_SEND_ERROR

opext AP_MAPPED_CONVERSATION. For nonblocking operation, this flag can be ORed together with AP_NON_BLOCKING.

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **MC_ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

err_type Indicates the type of the error being reported—application program or service program.

AP_PROG

err_dir Indicates whether the error being reported is in the data received from the partner transaction program, or in the data the local transaction program was about to send.

This parameter is used only when the **SEND_ERROR** verb is being issued in SEND_PENDING state.

AP_RCV_DIR_ERROR
AP_SEND_DIR_ERROR

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameters:

primary_rc AP_OK

rts_rcvd Request-to-send-received indicator.

AP_YES
AP_NO

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_ERROR_DIRECTION
AP_BAD_TP_ID

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, "APPC Common Return Codes."

Verb Issued in Any Permitted State: The following return codes can be generated when the **MC_SEND_ERROR** verb is issued in any permitted state:

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

Verb Issued in SEND State: The following return codes can be generated only if the MC_SEND_ERROR verb is issued in SEND state:

AP_ALLOCATION_ERROR

- AP_SECURITY_NOT_VALID
- AP_TRANS_PGM_NOT_AVAIL_RETRY
- AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
- AP_TP_NAME_NOT_RECOGNIZED
- AP_PIP_NOT_ALLOWED
- AP_PIP_NOT_SPECIFIED_CORRECTLY
- AP_CONVERSATION_TYPE_MISMATCH
- AP_SYNC_LEVEL_NOT_SUPPORTED

AP_DEALLOC_ABEND
AP_PROG_ERROR_PURGING

Verb Issued in RECEIVE State: The following return code can be generated only if the verb is issued in RECEIVE state:

AP_DEALLOC_NORMAL

MC_TEST_RTS

The **MC_TEST_RTS** verb determines whether a request-to-send notification has been received from the partner transaction program.

VCB Structure

```
typedef struct mc_test_rts
{
 unsigned short      opcode;              /  verb operation code          /
 unsigned char       opext;               /  verb extension code          /
 unsigned short      primary_rc;          /  primary return code          /
 unsigned long       secondary_rc;        /  secondary return code        /
 unsigned char       tp_id[8];            /  TP identifier                /
 unsigned long       conv_id;             /  conversation identifier      /
 unsigned char       reserv3;             /  reserved                     /
} MC_TEST_RTS;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_TEST_RTS

opext AP_MAPPED_CONVERSATION

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **MC_ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

Returned Parameters: Successful Execution

If the verb executes successfully, Personal Communications returns the following parameter:

primary_rc Indicates whether a request-to-send notification has been received from the partner transaction program.

AP_OK
AP_UNSUCCESSFUL

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_TP_BUSY
AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR

MC_TEST_RTS_AND_POST

The **MC_TEST_RTS_AND_POST** verb asynchronously determines whether a request-to-send notification has been received from the partner transaction program. A transaction program can issue an **MC_TEST_RTS_AND_POST** at any time, even when there is another verb outstanding on the conversation. **MC_TEST_RTS_AND_POST** returns when a request-to-send notification is received, or when the conversation ends, or when a conversation failure is detected.

VCB Structure

```
typedef struct mc_test_rts_and_post
{
 unsigned short      opcode;              /  verb operation code        /
 unsigned char       opext;               /  verb extension code        /
 unsigned char       format;              /  format                        /
 unsigned short      primary_rc;          /  primary return code        /
 unsigned long       secondary_rc;        /  secondary return code      /
 unsigned char       tp_id[8];            /  TP identifier              /
 unsigned long       conv_id;             /  conversation identifier  /
 unsigned char       reserv3;             /  reserved                     /
 unsigned long       sema;                /  post handle for verb       /
} MC_TEST_RTS_AND_POST;
```

Supplied Parameters

The transaction program supplies the following parameters to Personal Communications.

opcode AP_M_TEST_RTS_AND_POST

opext AP_MAPPED_CONVERSATION

tp_id Identifier for the local transaction program.

The value of this parameter was returned by the **TP_STARTED** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

conv_id Conversation identifier.

The value of this parameter was returned by the **MC_ALLOCATE** verb in the invoking transaction program or by **RECEIVE_ALLOCATE** in the invoked transaction program.

sema The API can use this field as it requires. Personal Communications returns the same value in the verb reply. Typically the API stores some posting information in this field, as for example, a callback address or a semaphore handle.

Returned Parameters: Successful Execution

If the verb executes successfully (that is, a request-to-send notification is received), Personal Communications returns the following parameter:

primary_rc AP_OK

Returned Parameters: Conversation Ended

If the verb returns because the conversation has ended or a conversation failure has been detected, Personal Communications returns the following parameter:

primary_rc AP_UNSUCCESSFUL

Returned Parameters: Parameter Check

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

primary_rc AP_PARAMETER_CHECK

secondary_rc AP_BAD_CONV_ID
AP_BAD_TP_ID

Returned Parameters: Other Conditions

The conditions generating the following possible primary return codes (**primary_rc**) and indented secondary return codes (**secondary_rc**) are described in Appendix A, “APPC Common Return Codes.”

AP_CONVERSATION_TYPE_MIXED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CANCELLED

Part 2. Common Services API

Chapter 7. Procedure Entry Points and Common Services Verbs (CSV) 147
Procedure Entry Points 147
ACSSVC 147
WinCSV() 147
WinCSVCleanup() 148
WinAsyncCSV() 149
WinCSVStartup() 150
GetCsvReturnCode() 151
TrnsDt 152
Common Services Verbs 156
GET_CP_CONVERT_TABLE 157
CONVERT 160

© Copyright IBM Corp. 1994, 1996

Chapter 7. Procedure Entry Points and Common Services Verbs (CSV)

Personal Communications provides a common services programming interface. This API consists of common services verbs (CSVs) that can be used by application programs that use Personal Communications APIs.

Any Personal Communications application program can use these common services verbs to do one or more of the following:

Maintain a code page translation table for single byte languages (**GET_CP_CONVERT_TABLE**)

Convert an ASCII string to EBCDIC or EBCDIC to ASCII (**CONVERT**)

Convert a double byte character string from one code page to another (**TRNSDT**)

The language interface for the common services verbs is similar; your program calls **WINCSV32.DLL** instead.

Procedure Entry Points

The following sections describe the entry points for common services.

ACSSVC

This is a synchronous entry point for all CSV verbs. Personal Communications provides this entry point for compatibility with existing applications.

Syntax

void ACSSVC (long)

Input is a verb control block pointer.

Returned Values

Check the primary and secondary return codes for returned values.

WinCSV()

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

Syntax

void WINAPI WinCSV(long vcb)

Returned Values

vcb is the address of the verb control block.

Note: See also **WinAsyncCSV()** on page 149.

© Copyright IBM Corp. 1994, 1996

WinCSVCleanup()

This function terminates and deregisters an application from the CSV API.

Syntax

BOOL WINAPI WinCSVCleanup(void);

Returned Values

The return value specifies whether the deregistration was successful. If the value is not , Personal Communications successfully deregistered the application. Personal Communications did deregister the application if the value is .

Usage Notes

Use **WinCSVCleanup()** to deregister a CSV API application from the CSV API. For example, to free resources allocated to the specific application.

WinAsyncCSV()

The function provides an asynchronous entry point for **TRANSFER_MS_DATA** only. If an application uses this function for any other verb, the behavior is synchronous.

Syntax

```
HANDLE WlNAPI WinAsyncCSV(hWnd,
                          long);
```

Returned Values

The return value indicates whether the asynchronous resolution request was successful. If the function was successful, the actual return value is an asynchronous task handle. If the function was not successful, Personal Communications returns a .

Usage Notes

Upon completion of the asynchronous operation, the application's window *hWnd* receives the message returned by **RegisterWindowMessage** with "**WinAsyncCSV**" as the input string. The *wParam* argument contains the asynchronous task handle returned by the original function call. The *IParam* argument contains the original VCB pointer and can be dereferenced to determine the final return code.

If the function returns successfully, Personal Communications posts a **WinAsyncCSV()** message to the application when the operation completes or the conversation is canceled.

WinCSVStartup()

This function allows an application to specify the version of the Common Services Verbs API required and to retrieve details of the specific CSV API. This call is not required, but if used, the **WinCSVCleanup** call should be used also.

Syntax

```
int WINAPI WinCSVStartup (WORD wVersionRequired,
                          LPWCSVDATA);
```

Parameter	Description
wVersionRequired	Specifies the version of CSV API support required. The high-order byte specifies the minor version (revision) number; the low-order byte specifies the major version number.
LPWCSVDATA	Contains information about the underlying CSV API DLL.

Returned Values

The return value indicates whether the CSV API successfully registered the application and whether it can support the provided version number. If the value returned is , the CSV API does support the specified version and it successfully registered the application. Otherwise, one of the following values is returned.

WCSVVERNOTSUPPORTED
: This particular CSV API does not provide the version of CSV API support requested.

WCSVINVALID
: The CSV API could not determine the requested version.

Usage Notes

WinCSVStartup() is intended to help with compatibility with future releases of the API. The current version supported is 1.0.

The following structure describes details of the actual CSV API implementation.

```
typedef struct tagWCSVDATA { WORD wVersion;
                    char  szDescription[WCSVDESCRIPTION_LEN+l]:
                     } WCSVDATA, FAR  LPWCSVDATA:
```

When an application has made its last CSV API call, it calls **WinCSVCleanup()**.

GetCsvReturnCode()

Use this entry point to convert the primary and secondary return codes in the VCB to a printable string. It returns a standard set of error strings for use by application programs.

Syntax

```
int WINAPI GetCsvReturnCode (struct csv_hdr  vcb,
                             UINT  buffer_length,
                             unsigned char  buffer_addr);
```

Parameter	Description
vcb	The address of the verb control block.
buffer_length	The length of the buffer pointed to by **buffer_addr**. The recommended length is 256.
buffer_addr	The address of the buffer that will hold the formatted, null-terminated string (length of the string in the specified buffer).

Returned Values

x2 1 The parameters are not valid; the function could not read from the specified **vcb** or could not write to the specified buffer.

X2 2 The specified buffer is too small.

Usage Notes

The descriptive error string returned in **buffer_addr** does not terminate with a new line character (**n**).

TrnsDt

This function converts the SBCS and DBCS strings from one code page to another. Personal Communications provides **TrnsDt** in the **TRNSDT.DLL** file. **TransDt** is available only on a DBCS session.

Syntax

TrnsDt (PASSSTRUCT passparm);

Function

This function converts the SBCS and DBCS strings from one code page to another. In the following table, "X" indicates that Personal Communications supports the conversion between the pair of code pages and "**-**" (hyphen) indicates that Personal Communications does not support that conversion.

Japan		932	930	931	939	290	037	1027
	932	-	X	X	X	X	X	X
	930	X	-	-	-	-	-	-
	931	X	-	-	-	-	-	-
	939	X	-	-	-	-	-	-
	290	X	-	-	-	-	-	-
	037	X	-	-	-	-	-	-
	1027	X	-	-	-	-	-	-
Korea		949	833	834	933			
	949	-	X	X	X			
	833	X	-	-	-			
	834	X	-	-	-			
	933	X	-	-	-			
Taiwan		950	037	835	937			
	950	-	X	X	X			
	037	X	-	-	-			
	835	X	-	-	-			
	937	X	-	-	-			
China		1381	836	837	935			
	1381	-	X	X	X			
	835	X	-	-	-			
	837	X	-	-	-			
	935	X	-	-	-			

Use the header file **TRNSDT.H** to compile, and use the **TRNSDT.LIB** file from the Personal Communications LIB subdirectory to link.

PassParm Format

The *passparm* format is as follows:

WORD *parm_length*
Length of this structure (input)

WORD *exit_code*
Exit code (output)

0000H Normal end.

0001H Not supported conversion specified.

000CH Exit_code field is not initialized to .

0080H The last character is the left half of a DCBS. Null character is filled instead.

WORD *in_length*
Length of the source buffer (input)

LPBYTE *in_addr*
Source buffer address (input)

WORD *out_length*
Length of target buffer (input)

If the specified length is too small to return all of the converted data, the required length is returned.

LPBYTE *out_addr*
Target address buffer (input)

WORD *trns_id*
Reserved to zero (input)

WORD *in_page*
Source code page (input)

WORD *out_page*
Target code page (input)

WORD *option*
Option (input/output)

Input Input options are as follows;

Bit 15-9	Reserved to zero
Bit 8	Target string has SO/SI
Bit 7-3	Reserved to zero
Bit 2	Use non-editable SBCS table
Bit 1	Source string starts with DBCS
Bit 0	Source string has SO/SI

Output Output options are as follows:

4	End at DBCS
0	End at non-DBCS

Notes:

1. Bit 8 and Bit 0 should be set as follows:

   Conversion from PC to host Bit 8=1
   Conversion from PC to host Bit 0=0

   Conversion from host to PC Bit 8=0
   Conversion from host to PC Bit 0=1

2. Use **SYSCTBL.EXE** to specify the name of the customized table that **TrnsDt** uses. To convert an SBCS string, **TrnsDt** uses the customized table with the **Option** parameter bit 2 set to FALSE. **TrnsDt** uses the default table if bit 2 is set but the name of the table is not specified. To convert a DBCS string when the name of the table is specified using **SYSCTBL.EXE**, **TrnsDt** always uses the customized table. In this case, the **Option** parameter for bit 2 is not used.
3. Generally, **TrnsDt** requires that the host data include SO/SI control characters as a pair. However, to convert a part of a mixed data string, the data must start with a double-byte character without an SO control character. In this case, data does not identify the double-byte character. Bit 1 is useful in such a case. When you set bit 1 to 1, **TrnsDt** processes the start of the buffer as a double-byte character or SO control character.

Return Code

0 NO_ERROR

2 ERROR_FILE_NOT_FOUND
TrnsDt cannot find the table used for converting the specified code.

87 ERROR_INVALID_PARAMETER
Parameter is not valid.

111 ERROR_BUFFER_OVERFLOW
The target buffer is too small.

150 ERROR_MEMORY_ALLOCATE
Memory allocation error.

Additional Information

Even a small buffer can handle a large data conversion successfully by using the exit code and option parameters of **TrnsDt**. First, start **TrnsDt** using a small source buffer and a double- or triple-sized destination buffer (for cases from PC to host), and see how the conversion ends, based on the exit code you receive. Then proceed accordingly.

For example, when the conversion divides a double-byte character into two parts, or it ends incompletely between SO and SI control characters, define the buffer pointer and its position, then perform the next call.

VCB Structure

The following example translates the host code x4 4 to PC code.

```
#include "trnsdt.h"

PASSSTRUCT          passparm;
char                bufs[2 ],  buft[2 ];
int                 rc;

//Setup the string to be translated
bufs[ ] =  x e;
bufs[1] =  x4 ;
bufs[2] =  x4 ;
bufs[3] =  x4f;

//Setup the parameter
passaparm.parm_length = 24;
passparm.exit_code      =  ;
passaparm.in_length     =  4;
passaparm.in_addr       =  &bufs[ ];
passaparm.out_length    =  2 ;
passaparm.out_addr      =  &buft[ ];

 passaparm.trns_id      =  ;
 passaparm.in_page      = 93 ;
 passaparm.out_page     = 932;
 passaparm.option       = 1;

//Translate the string via TrnsDt
if (rc = TrnsDt(&passaparm))
     printf("Error Return Code = %d\n\r", rc);
     printf("Exit Code = %d\n\r", passaparm.exit_code);
    exit( );
else
  .....
```

Common Services Verbs

Personal Communications provides the following verbs for the Common Services API.

GET_CP_CONVERT_TABLE

This verb provides a utility service that builds a conversion table from one code page to another. This verb returns a 256-byte conversion table that applications can use to perform table lookups on characters to convert character strings.

A program might need to perform data conversion when it communicates with a node that expects data encoded in a different code page.

VCB Structure

```
struct get_cp_convert_table
  {
   unsigned short opcode;                 / Verb identifying operation code.          /
  unsigned  char   opext;                 /  Reserved.                                /
  unsigned  char   reserv2;               /  Reserved.                                /
   unsigned short primary_rc;             / Primary return code from verb.           /
   unsigned long    secondary_rc;         / Secondary (qualifying) return code.      /
   unsigned short source_cp;              / Source code page for conversion table. /
   unsigned short target_cp;              / Target code page for conversion table. /
   unsigned char    conv_tbl_addr;        / Address to put conversion table at.      /
   unsigned char    char_not_fnd;         / Character not found option: either       /
                                            / substitute character or round trip.    /
  unsigned  char   substitute_char;  /  Substitute  character  to  use.              /
  };
```

Supplied Parameters

source_code_page

The code page number from which the replacement characters are drawn. The number for the code page can be one of the following:

ASCII code pages (in decimal)
- 437 US IBM PC
- 813 Greece
- 819 ANSI Standard
- 850 Multilingual
- 852 Czechoslovakia/Hungary/Poland/Yugoslavia
- 855 Cyrillic
- 857 Turkey
- 860 Portuguese
- 861 Iceland
- 862 Hebrew
- 863 Canada-French
- 864 Arabic
- 865 Nordic
- 866 Cyrillic
- 874 Thai
- 912 Latin 2
- 915 Cyrillic
- 920 Turkey
- 1250 Latin 2
- 1251 ANSI

EBCDIC code pages (in decimal)
- 037 United States/Canada-French/Netherlands/Portugal/Brazil
- 273 Germany/Austria
- 277 Denmark/Norway
- 278 Finland/Sweden

- 280 Italy
- 284 Latin America/Spain
- 285 United Kingdom
- 297 France
- 500 Belgium/Switzerland-French/Switzerland-German
- 870 Czechoslovakia/Hungary/Poland/Yugoslavia
- 871 Iceland
- 875 Greece
- 1025 Cyrillic
- 1026 Turkey

User defined code pages

- 65280 through 65534
- When using user-defined code pages, first define the registry entry with the user-defined path to the CPT files as follows:

  HKEY_LOCAL_MACHINE/Software/IBM/Personal_Communications/Current_Version/COMCPT

Note: Only identical characters in the source and target code pages are guaranteed to be converted into each other. Character pairs designated in the standards that merely resemble each other are not usually converted into each other.

target_code_page
The code page number for the target strings to be converted. The number can be any of those shown for **source_code_page**.

convert_table addr
The address of the buffer that is to receive the 256-byte conversion table. This buffer must be in a read/write segment.

character_not_found
The action to be taken if a character in the source code page does not exist in the target code page. You must specify one of the following values:

ROUND_TRIP This option causes the values to be stored in the conversion table so that if a conversion table is generated by reversing the source and target code pages, the result of a conversion from source to target code page and back again results in the original character. You must select the **ROUND_TRIP** option for both table generations for this option to run.

SUBSTITUTE Store the character specified in the parameter **substitute_character** in the conversion table.

substitute_character
The byte stored in the conversion table if a character in the source code page does not exist on the target code page and if the **character_not_found** parameter is set to **SUBSTITUTE**.

Returned Parameters: Successful Execution

The OK return code indicates that the **GET_CP_CONVERT_TABLE** verb ran successfully.

The following parameter is returned when the return code is OK:

convert_table The conversion table was built at the address specified by **convert_table_addr**.

Returned Parameters: Parameter Check

primary_rc	SV_PARAMETER_CHECK
secondary_rc	SV_INVALID_CHAR_NOT_FOUND SV_INVALID_DATA_SEGMENT SV_INVALID_SOURCE_CODE_PAGE SV_INVALID_TARGET_CODE_PAGE

CONVERT

This verb converts ASCII character strings to EBCDIC and EBCDIC character strings to ASCII.

A program might perform data conversion when it communicates with a node that expects EBCDIC data or when it must convert names to pass over an interface, such as APPC, that requires EBCDIC names.

Note: The **CONVERT** verb is not supported by DBCS. You can use **TrnsDt** to convert strings that have double-byte characters.

VCB Structure

```
struct convert
  {
   unsigned short opcode;              / Verb identifying operation code.          /
   unsigned char opext;                / Reserved.                                 /
   unsigned char reserv2;              / Reserved.                                 /
   unsigned short primary_rc;          / Primary return code from verb.            /
   unsigned long secondary_rc;         / Secondary (qualifying) return code.       /
   unsigned char direction;            / Direction of conversion - ASCII to        /
                                         / EBCDIC or vice-versa.                   /
   unsigned char char_set;             / Character to use for the conversion       /
                                         / A, AE, or user-defined G.               /
   unsigned short len;                 / Length of string to be converted.         /
   unsigned char       source;         / Pointer to string to be converted.        /
   unsigned char       target;         / Address to put converted string at.       /
  };
```

Returned Parameter Syntax

return_code OK error codes

Supplied Parameters

direction The nature of the code conversion.

ASCII_TO_EBCDIC Converts ASCII characters to EBCDIC

EBCDIC_TO_ASCII Converts EBCDIC characters to ASCII

character_set The set of characters permitted in the source string. You can specify three types of ASCII/EBCDIC conversion tables for use by the **CONVERT** verb: A, AE, and G. The type-A and type-AE tables are defined within Personal Communications. The type-G table is user-defined; its file name is specified on the Personal Communications configuration menus. The type-G table can be used to convert any character.

The format of a conversion table consists of 32 lines of 32 characters each. Each line represents 16 printable hexadecimal characters followed by a carriage return and line feed. The first 16 lines provide the information for ASCII-to-EBCDIC conversion. The second 16 lines provide the information for EBCDIC-to-ASCII conversion. The table must include all 32 lines.

When Personal Communications performs a conversion, it uses the numeric equivalent of each incoming character as a 0-origin index into the conversion table. This index specifies the table location containing the hexadecimal value of the converted

character. For example, assume the 48th position in the table contains a value of X' F0'. Personal Communications converts incoming characters with a value of 48 (X ' 30 ') to a value of 240 (X ' F0 ').

Table A Table A converts uppercase letters A through Z, numeric characters 0 through 9, and special characters $, #, and @. The first character of the source string must be either an uppercase letter or one of the three special characters; if it is not, no conversion is done, and the INVALID_FIRST_CHARACTER secondary return code is returned. In the ASCII-to-EBCDIC direction, lowercase ASCII characters are converted to uppercase EBCDIC characters.

Trailing blanks (blanks at the end of the source string) are converted to blanks in both directions. In contrast, embedded blanks are converted to X'00'.

If any source character is converted to X'00', CONVERSION_ERROR is returned. However, the entire conversion is completed.

Table AE Table AE converts alphanumeric characters (A through Z, 0 through 9), special characters $, #, and @, and the period (.). There are no restrictions on the first character of the string.

Trailing blanks (blanks at the end of the source string) are converted to blanks in either direction. In contrast, embedded blanks are converted to X'00'.

If any source character is converted to X'00', CONVERSION_ERROR is returned. However, the entire conversion is completed.

Table G You can use a G table to convert from any character to any other character (not just from ASCII to EBCDIC or EBCDIC to ASCII). However, you must specify ASCII_TO_EBCDIC on the **CONVERT** verb to use the top half of the table and specify EBCDIC_TO_ASCII to use the bottom half.

Personal Communications will look in the registry under

HKEY_LOCAL_MACHINE/Software/IBM/Personal_Communications/Current_Version/COMTBLG

to get the full path name to the G table. The other supplied parameters for the **CONVERT** verb are:

length The number of characters to be converted.

The length of the string must not extend beyond the segment size allocated for **source_addr** or **target_addr**.

source_addr
The address of the character string converted.

target_addr address
The address receiving the converted character string.

Note: If the application does not require preservation of the source string, it can specify the same variable for **source_addr** and **target_addr**.

Returned Parameters: Successful Execution

The OK return code indicates that the **CONVERT** verb ran successfully.

The following shows the primary and secondary error return codes associated with the **CONVERT** verb and the location of the return code's description.

Returned Parameters: Parameter Check

primary_rc	SV_PARAMETER_CHECK
secondary_rc	SV_INVALID_DIRECTION SV_TABLE_ERROR SV_INVALID_CHARACTER_SET SV_INVALID_FIRST_CHARACTER SV_CONVERSION_ERROR SV_INVALID_DATA_SEGMENT

Returned Parameters: Other Conditions

primary_rc	SV_UNEXPECTED_DOS_ERROR

Part 3. LUA API

Chapter 8. Introducing the IBM Conventional LU Application 165
Understanding LUA and SNA 165
Connection Capabilities 165
LUA Application Programs 165
LUA Verbs 165
LUs, Local LUs, and Partner LUs 166
SNA Layers 166
Data Link Control Layer 166
Path Control Layer 166
Transmission Control Layer 167
Data Flow Control Layer 167
Presentation Services Layer 167
Using SNA Sessions 167
Prerequisites to an SNA Session 168
Starting Sessions 168
Transferring Data on an LU-LU Session 169
Stopping Sessions 169
Disconnecting the Host Link 170
Message Numbers 170
Restarting and Resynchronizing a Session 170
Using Protocols to Control Requests and Responses 171
Using the Pacing Protocol 171
Using the Half-Duplex Contention/Flip-Flop Protocol 172
Using the Bracket Protocol 172
Using the Data-Chaining Protocol 173
Configuration 173
LUA Verbs 174
Verb Summary 174
Asynchronous Verb Completion 174
Sample LUA Communication Sequence 175
LUA Compatibility 176
BIND Checking 176
Negative Responses and SNA Sense Codes 177
Pacing 178
Segmentation 178
Courtesy Acknowledgments 178
Purging Data to End of Chain 178
Configuration Information 179
LUA LU Pool (optional) 179

Chapter 9. Understanding LUA 181
RUI Sessions 181
Issuing RUI Verbs 181

Chapter 10. Writing LUA Application Programs 183
Calling the Dynamic Link Libraries 183
Procedure Entry Points 183
WinRUI 183
WinRUICleanup() 185
WinRUIGetLastInitStatus() 186

© Copyright IBM Corp. 1994, 1996

WinRUIStartup() 189
RUI() 190
GetLuaReturnCode() 191
Using the DLL Interface 193
Load-Time Linking 193
Run-Time Linking 193
Multiple Processes 193
Multiple Threads 193
LUA Verb Postings 193
Converting to EBCDIC from ASCII 194

Chapter 11. RUI Verbs 195
LUA Verb Control Block Format 195
Common Verb Header 196
RUI_BID Data Structure 200
RUI_BID 201
RUI_INIT 206
RUI_PURGE 210
RUI_INIT_STATUS 213
RUI_READ 214
RUI_TERM 220
RUI_WRITE 223

Chapter 8. Introducing the IBM Conventional LU Application

This chapter introduces the IBM conventional logical unit application (LUA) access method and describes its relationship to Systems Network Architecture (SNA).

Understanding LUA and SNA

The IBM LUA access method provides Personal Communications with an application programming interface (API) for secondary dependent logical units (LUs). LUA consists of system software and interfaces that supply input/output (I/O) service routines to support communications using LU types 0, 1, 2, and 3 SNA protocols.

The services that LUA provides to application programs include only those that support data communications; LUA does not provide any device emulation facilities. However, LUA does provide a unique subset of presentation services layer functions.

Personal Communications AS/400 and 3270 for Windows 95 must be installed and configured before an LUA application program can run on the workstation. Refer to *Personal Communications Up and Running* for information about installing and configuring Personal Communications.

Connection Capabilities

The main objective of any communications system is to connect with other systems. The goal of SNA is to supply common protocols that give universal connectivity. LUA communication and connectivity requirements include the System/370* (S/370*).

LUA Application Programs

In this book, the term *LUA application program* means an application program, or a portion of an application program, that uses LUA communications functions. Application programs use these functions to communicate with application programs on other systems that support LU types 0, 1, 2, or 3.

As a local LUA application program runs, it exchanges data with a remote host application program. The local and remote application programs are called *partner* application programs.

LUA Verbs

A verb is a formatted request that is processed by LUA. An application program issues a verb to request that LUA take some action. LUA verbs are coded as control blocks. Each verb control block has a precisely defined format. To use the LUA facilities, an application program supplies verb control blocks to the LUA API.

An LUA verb always returns immediately to its caller. If the return code is IN_PROGRESS, the application needs to wait for completion of the verb, using the

© Copyright IBM Corp. 1994, 1996

posting method specified in the verb request. See Chapter 10, “Writing LUA Application Programs” for a description of LUA verb postings.

Verb control block layouts are available in the Personal Communications **INCLUDE** directory. You can use the verb control block layouts and sample programs to help you write LUA application programs.

LUs, Local LUs, and Partner LUs

A logical unit (LU) manages the exchange of data between application programs. Every LUA application program gains access to an SNA network through an LU, which acts as an intermediary between the LUA application program and the SNA network.

In LUA, there is a one-to-many relationship between LUA application program processes and LUs. One LUA application program process can own multiple LUs simultaneously, but a given LU can be owned by only one LUA application program process simultaneously. Before a second application program process can use an LU, the first application program must release the LU.

An LUA application program issues LUA verbs to its local LU. These verbs cause commands and data to flow across the network to the partner LU.

Note: You need to define your local LU only once for each machine, as described in *Up and Running*.

SNA Layers

SNA is a hierarchical structure that consists of seven well-defined layers. Each layer in the architecture performs a specific function. Understanding the layered structure of SNA helps in understanding the various functions that LUA supplies. The following descriptions of the five highest-level SNA layers show the relationship between LUA and SNA.

Data Link Control Layer

The data link control (DLC) layer consists of the elements that provide an interface to the hardware. The DLC elements supply support for various DLC protocols, such as Synchronous Data Link Control (SDLC) and the IBM Token-Ring Network. The DLC layer supplies a common link appearance to the elements in the path control (PC) layer. The DLC layer is common to all Personal Communications LU implementations, including LUA.

Path Control Layer

The path control (PC) layer of SNA in a peripheral node supplies basic functions, such as routing to and from multiple half-sessions within its node. SNA permits the PC layer to route to and from only one data link at a time. The PC layer is common to all Personal Communications LU implementations, including LUA.

Transmission Control Layer

The transmission control (TC) layer of SNA supplies the connection-point-manager function and the session-control function for each locally supported half-session. The connection-point-manager function controls sequence-number checking, pacing, and other support functions that relate to half-session data flows. The session-control function supplies session-specific support for starting, pacing, enciphering, deciphering, and other support functions that relate to session-related data flows. LUA contains an implementation of the TC layer for LU types 0, 1, 2, and 3 within Personal Communications.

Data Flow Control Layer

The data flow control (DFC) layer of SNA controls the flow of function management data (FMD) requests and FMD responses between FMD pairs that are in sessions and between sessions. The data flow control layer supplies various functions, such as request/response formatting, data-chaining protocol, request/response correlation, send- and receive-mode protocols, bracket protocol, error-recovery protocol, stop-bracket-initiation protocol, and queued-response protocol. LUA contains an implementation of the data flow control layer for LU types 0, 1, 2, and 3 within Personal Communications.

Presentation Services Layer

The presentation services (PS) layer of SNA contains the function that presents the communications data interface to the user. The presentation services layer is defined in the architecture for all LU types except LU 0. LUA contains a unique subset of the presentation services layer within Personal Communications. For more information about the presentation services layer, refer to *Systems Network Architecture Concepts and Products*.

The LU services functions are a part of the SNA-session message flow layers. These functions supply support before session establishment, build session structures, and take down session structures. LUA functions interface with common Personal Communications support to define LUs and to start and stop SNA sessions.

Using SNA Sessions

Before an LUA application program can communicate with a partner host application program, the respective LUs must be connected in a mutual relationship called a session. An SNA session is a logical connection that enables two network addressable units (NAUs) to communicate with each other; an LU is one kind of NAU. Because the session connects two LUs, it is called an *LU-LU session*. LU-LU sessions enable end users to exchange data.

A session manages how data moves between a pair of LUs in an SNA network. Therefore, sessions are concerned with such things as the quantity of data being transmitted, data security, network routing, data loss, and traffic congestion. Session characteristics are determined by the SNA **BIND** command originating from the primary LU, when the secondary LU accepts the **BIND** command.

Prerequisites to an SNA Session

An LU-LU session consists of communication between a primary logical unit (PLU) and a secondary logical unit (SLU). The SLU is implemented by the LUA application program. Before data can be transferred between a PLU and an SLU on an LU-LU session, the following events must occur:

1. Personal Communications activates the data link.
2. When the data link is ready, the system services control point (SSCP) establishes a session between itself and a physical unit (SSCP-PU session) by sending an Activate Physical Unit (**ACTPU**) command and reading a positive response from Personal Communications. Personal Communications sends a positive response if the PU address from the **ACTPU** command corresponds to the configuration information.
3. The SSCP establishes a session between itself and the logical unit (SSCP-LU session) by sending an Activate Logical Unit (**ACTLU**) command and reading a positive response from Personal Communications. Personal Communications sends a positive response if the LU address from the **ACTLU** command corresponds to the configuration information.

Starting Sessions

Either the SLU or the PLU can start an LU-LU session.

Starting an LU-LU Session from an SLU

After the SSCP-LU session is established, the SLU program can request an LU-LU session by sending the Initiate Self (**INITSELF**) command to the SSCP. The SSCP receives the **INITSELF** command and checks whether the named host application program is valid. A host application program is valid if it is known and active. If the host application program is valid, the SSCP sends a positive response to the SLU, and the PLU starts the session. If the host application program is not valid, the SSCP sends a negative response to the SLU, and the PLU does not start the session.

If the SSCP sends a positive response to an **INITSELF** command but the session cannot be established, the SSCP sends a Network Services Procedure Error (**NSPE**) command to the SLU to stop the attempt to establish a session. The SLU can reissue the **INITSELF** command after the **NSPE** command.

Starting an LU-LU Session from a PLU

The PLU program can start unsolicited LU-LU sessions. The PLU starts sessions by generating a **BIND** command. A subsequent positive response establishes the agreement to communicate. A data field that is associated with the **BIND** command contains the name of the PLU application program and the session **BIND** parameters. For more information about the format of this data field, refer to *Systems Network Architecture: Formats*.

For nonnegotiable **BIND**s, the SLU returns a positive response if the parameters are acceptable. If the parameters are unacceptable, the SLU returns a negative response with sense data to the PLU.

The negotiable **BIND** command permits the SLU to return a positive response with a minimum of 26 bytes of updated session parameters indicating compatibility with the PLU parameters. If the PLU finds the returned parameters acceptable, it sends

a Start Data Traffic (**SDT**) command. If the returned parameters are unacceptable, the PLU sends an **UNBIND** command that indicates unacceptable negotiable **BIND** command parameters from the SLU.

Transferring Data on an LU-LU Session

After the LU-LU session is established and the SLU program responds to the **SDT** command, data transfer can begin. For a data transmission operation, a message moves from end-user storage to Personal Communications storage until it is transmitted. For a data-reception operation, Personal Communications places a message in its own storage and then moves the message into end-user storage.

Quiesce protocols suspend the transfer of data in an LU-LU session. The PLU or the SLU can send the following Quiesce protocol commands:

Quiesce at End of Chain (**QEC**). This command requests that the receiver of this command stop sending data after sending the last part in a data chain. A data chain is a series of related messages. For more information about data chaining, see "Using the Data-Chaining Protocol" on page 173.

Quiesce Complete (**QC**). This command notifies a **QEC** command that data transfer is suspended. When the SLU sends the **QC** command, Personal Communications prevents the SLU from sending any normal-flow messages until the Release Quiesce (**RELQ**) command is received.

Release Quiesce (**RELQ**). This command notifies the receiver that data can again be transferred.

Stopping Sessions

When all data has been transferred and verified, the session can end. An SLU must end one session before it can begin a different session with either the same or another PLU.

Stopping an LU-LU Session by an SLU

An SLU can end an LU-LU session in either of two ways:

By sending a Terminate-Self (**TERMSELF**) command or an **UNBIND** command. Either command results in an immediate ending.

By sending a Request Shutdown (**RSHUTD**) command. This command solicits an **UNBIND** from the PLU.

To end a session immediately, the SLU sends the **TERMSELF** command to the SSCP, which checks whether the named LUA application program is the one participating in this session. If it is, the SSCP sends a positive, non-data response. Depending on the host SNA version being used, the SSCP can send a **CLEAR** command, which purges all messages from the LU-LU session, and can then send an **UNBIND** command to end the session. Alternatively, the SLU can send an **UNBIND** command to the PLU.

Stopping an LU-LU Session by a PLU

A PLU can end an LU-LU session in either of two ways:

By sending a **CLEAR** command and then an **UNBIND** command, or an **UNBIND** command only. Either method results in an immediate ending.

By sending the Shutdown (**SHUTD**) command. This command results in an orderly session termination. The SLU and PLU have a dialog that tells each to stop sending data and that ensures that data already sent is received.

Ending the LU-LU session has no effect on the SSCP-LU session.

Stopping an SSCP-LU Session and an SSCP-PU Session

The SSCP-LU session ends when the host sends the Deactivate Logical Unit (**DACTLU**) command to the SLU. When the last SSCP-LU session for Personal Communications ends, the SSCP can end the SSCP-PU session by sending a Deactivate Physical Unit (**DACTPU**) command.

Disconnecting the Host Link

When the host receives the response to the **DACTPU** command, it returns a command to Personal Communications such as the Set Disconnect Response Mode (**SDRM**) command when using SDLC protocol. The SSCP can also disconnect immediately at any time by sending the same command to Personal Communications, which ends all sessions. When sessions are ended in this manner, all SLUs that were active earlier receive a loss-of-contact indication.

Message Numbers

All normal-flow messages that are transmitted between the SLU and the PLU during an LU-LU session are numbered in sequence. The SLU maintains a sequence number for normal-flow messages from the SLU to the PLU and another sequence number for normal-flow messages from the PLU to the SLU. Each normal-flow message gets a sequence number one greater than the sequence number of the preceding normal-flow message. There is one pair of sequence numbers for each session that is established between an SLU and a PLU.

For LU-LU expedited-flow messages and for all SSCP-LU and SSCP-PU messages, unsequenced identifiers are used instead of sequence numbers.

When a session is reestablished or a **CLEAR** command is sent, the PLU and the SLU set their sequence numbers to 0. The PLU can change the sequence numbers with the Set and Test Sequence Numbers (**STSN**) command. This enables correct sequence numbering when a session is recovered or restarted.

When the SLU encounters a sequence number error, it sends a negative response to the PLU if a response was requested. When the SLU reads a response, the SLU uses the response sequence number to correlate the response with the original request. When the SLU writes a response, the SLU must supply the sequence number of the original request.

Restarting and Resynchronizing a Session

If the PLU or the SLU encounters an irrecoverable error, such as a line failure, you might need to resynchronize the LU-LU session after restarting it. Resynchronizing the LU-LU session includes reprocessing recoverable messages and (optionally) resetting the message sequence numbers. The application programs can include routines to retransmit lost messages.

When a session is restarted and resynchronized, the PLU sends the **BIND**, the **STSN**, and the **SDT** commands. When the **STSN** command is sent, a dialog can occur to establish the sequence numbers that are acceptable to both the PLU and the SLU. This dialog consists of a series of **STSN** messages and positive responses.

If the SLU determines that resynchronization is required, the SLU can send a Request Recovery (**RQR**) command, a negative response, or an LU-Status command (**LUSTAT**) with a description of the failure in the user sense bytes. If the PLU discovers the failure or receives an **RQR** command from the SLU, the PLU sends a **CLEAR** command to purge all LU-LU messages from the network, an **STSN** command to establish new sequence numbers, and then an **SDT** command.

Using Protocols to Control Requests and Responses

Various protocols can control the sequencing rules for requests and responses. This section describes some of the protocols used for managing the SNA network, transferring data, and synchronizing the states of network components.

Using the Pacing Protocol

To avoid a message-flow rate that is too fast for Communications Manager or the host, you can specify pacing in the **BIND** command. Pacing applies to the LU-LU normal flow only. While pacing, Personal Communications permits a specified number of messages to flow and waits for a response before permitting additional messages to be sent. You can specify pacing on the Personal Communications-to-host flow, the host-to-Personal Communications flow, or both. Once the LU-LU session starts, LUA handles all pacing with no participation by the application program.

Receive-Pacing Protocol

Receive-pacing protocol gives the PLU control over the number and the frequency of messages sent from the SLU on an LU-LU session. When the SLU receives pacing values in the **BIND** command, Personal Communications automatically enforces pacing for each SLU that communicates with the host.

During a positive response to a negotiable **BIND** command, you can change the pacing values to any number except 0. When the SLU sends the first message of a sequence, Personal Communications sets a bit in the request/response header (RH) that indicates a pacing response is to be returned. If the pacing count is exhausted before Personal Communications receives a pacing response from the PLU, Personal Communications cannot send additional data messages. If the application program issues a write operation and no pacing response is received, Personal Communications defers the write operation.

Send-Pacing Protocol

The SLU automatically controls the send-pacing protocol. If the pacing indicator is set on in a message from the PLU to the SLU, the SLU issues a pacing response when the application program reads the message. The message response can contain the pacing indicator or, if no response is required for the received message, the pacing response can be an isolated pacing response (IPR). The PLU can then send another pacing window of messages.

Using the Half-Duplex Contention/Flip-Flop Protocol

The change-direction (CD) indicator is used with both of the following protocols:

Half-duplex contention protocol, which is a normal-flow send/receive mode in which either half-session can send normal-flow requests at the beginning of the session or after sending or receiving the last request of a chain.

Half-duplex flip-flop protocol, which is a normal-flow send/receive mode in which one half-session sets the CD indicator in the response header (RH) on an end-of-chain to enable the other half-session to begin sending.

A CD indicator tells the receiver that transmitting can begin.

For example, if the SLU initiates a transaction, the SLU begins by sending the messages that completely describe the transaction. On the last message, the SLU sets the CD indicator to tell the PLU that it can begin transmitting a reply. If the PLU needs additional information to complete the transaction, it sends an inquiry and sets the CD indicator. The dialog proceeds in this half-duplex mode until the transaction is complete. During a half-duplex dialog, the SLU can use the **SIG** command to tell the PLU to stop sending data and to change the direction of the data flow.

Using the Bracket Protocol

Bracket protocol gives the SLU and the PLU context control of the data transmission, indicating that a session concerns a single transaction. Bracket protocol protects a current session from interruption by a concurrent transaction. A bracket encompasses the duration of a transaction.

The first message in the bracket contains a begin-bracket (BB) indicator, and the last message in the bracket contains an end-bracket (EB) indicator. A single message can be a bracket if it contains both indicators.

For a bracket session, the **BIND** command specifies one LU as the first speaker, and the other LU as the bidder. The first speaker can begin a bracket without permission from the other LU. The bidder, however, must request and receive permission from the first speaker to begin a bracket.

A **BID** command is a normal-flow request that is issued by the bidder to request permission to begin a bracket. A positive response to a **BID** command indicates that the first speaker will not begin a bracket but will wait for the bidder to begin a bracket. A negative response to a **BID** command indicates that the first speaker denies permission for the bidder to begin a bracket. The first speaker can send a Ready-to-Receive (**RTR**) command when permission is granted to start a bracket.

The first speaker indicates a negative response to a **BID** command with one of two response codes:

Bracket-Bid-Reject-RTR-Forthcoming
Indicates that an **RTR** command for that **BID** command will be sent later (granting permission to start a bracket). The bidder can wait for the **RTR** command or send the **BID** command again.

Bracket-Bid-Reject-No-RTR-Forthcoming
Indicates that no **RTR** command for that **BID** command will be sent later. The bidder must send the **BID** command again if the bidder still wants to begin a bracket.

Instead of sending a **BID** command followed by a first-in-chain FMD with a BB indicator, the bidder can attempt to initiate a bracket by sending a first-in-chain FMD with a BB indicator. The first speaker can grant the attempt with a positive response or it can refuse the attempt with a negative response that indicates either of the negative response codes. However, if the bidder stops the chain that carries the BB indicator by sending the **CANCEL** command, the bracket is not initiated, regardless of the response. The **RTR** command can be issued by the first speaker either to grant permission to the bidder to begin a bracket or to find out if the bidder wants to begin a bracket.

A positive response to an **RTR** command indicates that the bidder will initiate the next bracket. If the bidder does not want to initiate a bracket, the bidder issues a negative response with the RTR-Not-Required sense code.

Using the Data-Chaining Protocol

Data chaining is an optional protocol for transmitting a group of related messages. To send chained messages from the SLU, the SLU sets to 1 the begin-chain (BC) indicator for the message to indicate the first message in a chain. For all messages between the first and the last in the chain, the SLU sets the BC and the end-chain (EC) indicators to 0. For the last message in the chain, the EC is set to 1 again. When the SLU receives messages, it tests the chaining indicator to determine if the messages are chained.

The data-chaining protocol comprises three types of chains, as follows:

No-response chain. Each request in the chain is marked *no response*.

Exception-response chain. Each request in the chain is marked *exception response*.

Definite-response chain. The last request in the chain is marked *definite response*; all other requests in the chain are marked *exception response*.

When sending a message chain to the PLU, the SLU can send a **CANCEL** command if the SLU or the PLU finds a message error. If the SLU sends a **CANCEL** command to the PLU, the PLU discards all messages in the chain that it has received. If the PLU sends a negative response to any element of a chain, the SLU ends the chain normally or sends a **CANCEL** command.

Configuration

Each LU used by an LUA application must be configured using Personal Communications NOF verbs or through the SNA Node Configuration program. (Refer to *Personal Communications System Management Programming* for more information.) In addition, the Personal Communications configuration may include LUA LU pools. A pool is a group of LUs with similar characteristics, such that an application can use any free LU from the group. This can be used to allocate LUs on a first-come, first-served basis when there are more applications than LUs available, or to provide a choice of LUs on different links.

LUA Verbs

An application accesses LUA through LUA verbs. Each verb supplies parameters to LUA, which performs the desired function and returns parameters to the application.

Verb Summary

The following is a brief summary of the seven LUA verbs that an application can use. (For a detailed explanation of each verb, see Chapter 11, “RUI Verbs.”)

RUI_BID Enables the application to determine when information from the host is available to be read.

RUI_INIT Sets up the LU-SSCP session for an LUA application.

RUI_PURGE Cancels an outstanding **RUI_READ** verb.

RUI_READ Receives data or status information sent from the host to the LUA application's LU, on either the LU-SSCP session or the LU-LU session.

RUI_TERM Ends the LU-SSCP session for an LUA application. It also brings down the LU-LU session if it is active.

RUI_WRITE Sends data to the host on either the LU-SSCP session or the LU-LU session.

In addition, Personal Communications can return the RUI_INIT_STATUS indication to the LUA application. It carries information about the state reached in processing an outstanding **RUI_INIT** verb.

Asynchronous Verb Completion

Some LUA verbs complete quickly, after some local processing (for example the **RUI_PURGE** verb); however, most verbs take some time to complete because they require messages to be sent to and received from the host application. Because of this, LUA is implemented as an asynchronous interface; control can be returned to the application while a verb is still in progress, so the application is free to continue with further processing (including issuing other LUA verbs). The way that LUA returns control to the application is by way of a Windows 95 event handle in the verb.

If Personal Communications's verb response signal is delayed (for example, because it needs to wait for information from the remote node), then the stub should return the verb “asynchronously.” The API does this by setting the primary return code to LUA_IN_PROGRESS, and the **lua_flag2** to LUA_ASYNC. The application can now either perform other processing, or wait for notification from the API that the verb has completed. When the verb completes, the application is notfied by the setting of the primary return code in the VCB to its final value, and leaving the **lua_flag2** set to LUA_ASYNC.

Sample LUA Communication Sequence

The following is an example of an LUA communication sequence. It shows the LUA verbs used to start a session, exchange data, and end the session, and the SNA messages sent and received. The arrows indicate the direction in which SNA messages flow.

The following abbreviations are used:

SSCP norm	LU-SSCP session, normal flow
LU norm	LU-LU session, normal flow
LU exp	LU-LU session, expedited flow
+rsp	Positive response to the indicated message
BC	Begin chain
MC	Middle of chain
EC	End chain
CD	Change direction indicator set
RQD	Definite response required

Verb issued by LUA application	SNA message	Flow direction Application	Host
RUI_INIT	(ACTLU)	<-----	
	(ACTLU +rsp)	----->	
RUI_WRITE (SSCP norm)	INITSELF	----->	
RUI_READ (SSCP norm)	INITSELF +rsp	<-----	
RUI_READ (LU exp)	BIND	<-----	
RUI_WRITE (LU exp)	BIND +rsp	----->	
RUI_READ (LU exp)	SDT	<-----	
RUI_WRITE (LU exp)	SDT +rsp	----->	
RUI_WRITE (LU norm)	data, BC	----->	
RUI_WRITE (LU norm)	data, MC	----->	
RUI_WRITE (LU norm)	data, EC, CD, RQD	----->	
RUI_READ (LU norm)	data +rsp	<-----	
RUI_READ (LU norm)	data, BC	<-----	
RUI_READ (LU norm)	data, MC	<-----	
RUI_READ (LU norm)	data, EC, RQD	<-----	
RUI_WRITE (LU norm)	data +rsp	----->	
RUI_READ (LU exp)	UNBIND	<-----	
RUI_WRITE (LU exp)	UNBIND +rsp	----->	
RUI_TERM	(NOTIFY)	----->	
	(NOTIFY +rsp)	<-----	

In this example, the application performs the following steps:

1. Issues the **RUI_INIT** verb to establish the LU-SSCP session. (The **RUI_INIT** verb does not complete until Persoanl Communications has received an ACTLU message from the host and sent a positive response; however, these messages are handled by Personal Communications and not exposed to the LUA application.)
2. Sends an **INITSELF** message to the SSCP to request a **BIND**, and reads the response.

3. Reads a **BIND** message from the host, and writes the response. This establishes the LU-LU session.
4. Reads an **SDT** message from the host, which indicates that initialization is complete and data transfer can begin.
5. Sends a chain of data consisting of three RUs (the last indicates that a definite response is required), and reads the response.
6. Reads a chain of data consisting of three RUs, and writes the response.
7. Reads an **UNBIND** message from the host, and writes the response. This terminates the LU-LU session.
8. Issues the **RUI_TERM** verb to terminate the LU-SSCP session. (Personal Communications sends a NOTIFY message to the host and waits for a positive response; however, these messages are handled by Personal Communications and are not exposed to the LUA application.)

LUA Compatibility

The Personal Communications implementation of LUA is designed to be binary compatible with Microsoft* NT SNA Server and similar to the implementation of the RUI interface of OS/2 Communications Manager/2 version 1.0 LUA.

BIND Checking

During initialization of the LU-LU session, the host sends a **BIND** message to the Personal Communications LUA application that contains information such as RU sizes to be used by the LU-LU session. Personal Communications returns this message to the LUA application on an **RUI_READ** verb. It is the responsibility of the LUA application to check that the parameters specified on the **BIND** are suitable. The application has the following options:

Accept the **BIND** as it is, by issuing an **RUI_WRITE** verb containing an OK response to the **BIND**. No data needs to be sent on the response.

Try to negotiate one or more **BIND** parameters (this is only permitted if the **BIND** is negotiable). To do this, the application issues an **RUI_WRITE** verb containing an OK response, but including the modified **BIND** as data.

Reject the **BIND** by issuing an **RUI_WRITE** verb containing a negative response, using an appropriate SNA sense code as data.

See Chapter 11, “RUI Verbs,” for more information on the **RUI_WRITE** verb.

Note: Validation of the **BIND** parameters, and ensuring that all messages sent are consistent with them, is the responsibility of the LUA application. However, the following two restrictions apply:

Personal Communications rejects any **RUI_WRITE** verb that specifies an RU length greater than the size specified on the **BIND**.

Personal Communications requires the **BIND** to specify that the secondary LU is the contention winner, and that error recovery is the responsibility of the contention loser.

Negative Responses and SNA Sense Codes

SNA sense codes may be returned to an LUA application in the following cases:

When the host sends a negative response to a request from the LUA application, this includes an SNA sense code indicating the reason for the negative response. This is reported to the application on a subsequent **RUI_READ** verb, as follows:

- The primary return code is LUA_OK.
- The Request/Response Indicator, Response Type Indicator, and Sense Data Included Indicator are all set to 1, indicating a negative response which includes sense data.
- The data returned by the **RUI_READ** verb is the SNA sense code.

When Personal Communications receives invalid data from the host, it generally sends a negative response to the host and does not pass the invalid data to the LUA application. This is reported to the application on a subsequent **RUI_READ** or **RUI_BID** verb, as follows:

- The primary return code is LUA_NEGATIVE_RSP.
- The secondary return code is the SNA sense code sent to the host.

In some cases, Personal Communications detects that data supplied by the host is invalid, but cannot determine the correct sense code to send. In this case, it passes the invalid data in an Exception Request (EXR) to the LUA application on an **RUI_READ** verb in the following way:

- The Request/Response Indicator is set to zero, indicating a request.
- The Sense Data Included Indicator is set to one, indicating that sense data is included (this indicator is normally used only for a response).
- The message data is replaced by a suggested SNA sense code.

The application must then send a negative response to the message; it may use the sense code suggested by Personal Communications, or may alter it.

Personal Communications may send a sense code to the application to indicate that data supplied by the application was invalid. This is reported to the application on the **RUI_WRITE** verb that supplied the data, as follows:

- The primary return code is LUA_UNSUCCESSFUL.
- The secondary return code is the SNA sense code.

Distinguishing SNA Sense Codes from Other Secondary Return Codes

For a secondary return code which is not a sense code, the first two bytes of this value are always zero. For an SNA sense code, the first two bytes are non-zero; the first byte gives the sense code category, and the second identifies a particular sense code within that category. (The third and fourth bytes may contain additional information, or may be zero.)

Information on SNA Sense Codes

If you need information on a returned sense code, refer to *IBM Systems Network Architecture: Formats*. The sense codes are listed in numerical order by category.

Pacing

Pacing is handled by LUA; an LUA application does not need to control pacing, and should never set the Pacing Indicator flag.

If pacing is being used on data sent from the LUA application to the host (this is determined by the **BIND**), an **RUI_WRITE** verb may take some time to complete. This is because Personal Communications has to wait for a pacing response from the host before it can send more data.

If an LUA application is used to transfer large quantities of data in one direction, either to the host or from the host (for example, a file transfer application), then the host configuration should specify that pacing is used in that direction; this is to ensure that the node receiving the data is not flooded with data and does not run out of data storage.

Segmentation

Segmentation of RUs is handled by LUA. LUA always passes complete RUs to the application, and the application should pass complete RUs to LUA.

Courtesy Acknowledgments

Personal Communications keeps a record of requests received from the host in order to correlate any response sent by the application with the appropriate request. When the application sends a response, Personal Communications correlates this with the data from the original request, and can then free the storage associated with it.

If the host specifies exception response only (a negative response may be sent, but a positive response should not be sent), Personal Communications must still keep a record of the request in case the application subsequently sends a negative response. If the application does not send a response, the storage associated with this request cannot be freed.

Because of this, Personal Communications enables the LUA application to issue a positive response to an exception-response-only request from the host (this is known as a courtesy acknowledgment). The response is not sent to the host, but is used by Personal Communications to clear the storage associated with the request.

Purging Data to End of Chain

When the host sends a chain of request units to an LUA application, the application may wait until the last RU in the chain is received before sending a response, or it may send a negative response to an RU which is not the last in the chain. If a negative response is sent mid-chain, Personal Communications purges all subsequent RUs from this chain, and does not send them to the application.

When Personal Communications receives the last RU in the chain, it indicates this to the application by setting the primary return code of an **RUI_READ** or **RUI_BID** verb to LUA_NEGATIVE_RSP with a zero secondary return code.

Note: The host may terminate the chain by sending a message such as CANCEL while in mid-chain. In this case, the CANCEL message is returned to the application on an **RUI_READ** verb, and the LUA_NEGATIVE_RSP return code is not used.

Configuration Information

In order to run an LUA application over Personal Communications, it is necessary to configure the Personal Communications node with the following Node Operator Facility verbs:

1. Issue a **DEFINE_LS** verb.

   The **pu_name** parameter specifies the local PU that will manage the link to the host.
   The **solicit_sscp_sessions** parameter should be set to NAP_YES.

2. Issue a **DEFINE_LU_0_TO_3** verb for each LU.

3. If you wish to define one or more LU pools (described in "LUA LU Pool (optional)"), issue a **DEFINE_LU_POOL** verb for each LU pool. Each **DEFINE_LU_POOL** verb specifies the names of any LUs (defined in Step 2) that will be included in the pool.

For more information, refer to *Personal Communications AS/400 and 3270 for Windows 95 System Management Programming*.

LUA LU Pool (optional)

If required, you can configure more than one LUA LU for use by the application, and group the LUs into a pool. This means that an application can specify the pool rather than a specific LU when attempting to start a session, and will be assigned the first available LU from the pool.

An LUA application indicates to Personal Communications that it wants to start a session by issuing an **RUI_INIT** verb with an LU name. This name must match the name of an LUA LU or LU pool that has previously been defined in *Personal Communications System Management Programming*. Personal Communications uses this name as follows:

If the name supplied is the name of an LU that is not in a pool, a session will be assigned using that LU if it is available (that is, if it is not already in use by an LUA application).

If the name supplied is the name of an LU pool, or the name of a specific LU within the pool that is already in use, then a session will be assigned using the first available LU in the pool (if one is available).

Note: This may not be the LU whose name was specified on the **RUI_INIT** verb.

Chapter 9. Understanding LUA

LUA shares physical units (PUs) with the 3270 emulator API and the advanced program-to-program communications (APPC) API. LUA extends the capabilities of Personal Communications with the Request-Unit Interface (RUI) API, which is a basic interface.

LUA provides secondary support only for LU types 0, 1, 2, or 3. The types of SNA BIND profiles that are supported for LU-LU sessions depend on the access method that is used. Table 2 shows the permissible LU-LU function management profiles (FMPs) and transmission services profiles (TSPs) for the RUI.

Table 2. Permissible LU-LU FMPs and TSPs for the RUI

	Permissible FMPs	**Permissible TSPs**
RUI	2, 3, 4, 7, 18	2, 3, 4, 7
SLI	3, 4	3, 4

Note: The permissible FMP for the SSCP-LU for the RUI is 0; the permissible TSP is 1.

The number of sessions for each LUA application program depends on the address architecture, the available storage, and the LUA network definition. Each session has a unique LU; LUA does not support parallel sessions.

The RUI provides for message exchange with a partner LU at the SNA request-unit level.

RUI Sessions

An RUI session consists of the ownership of an LU for a period of time determined by the application, which can include establishing a session between an SSCP and an LU (called an SSCP-LU session). An RUI session can also include establishing one or more non-overlapped LU-LU sessions. If the SSCP-LU session fails because of a loss-of-contact or another reset condition, the RUI session ends. An RUI session begins with an **RUI_INIT** verb and ends normally with an **RUI_TERM** verb.

Issuing RUI Verbs

Table 3 on page 182 shows the valid conditions under which an RUI application program can issue verbs to the RUI API for a given LU. The entries in the leftmost column represent incoming verbs. The entries in the first row represent verbs that are in progress. If an entry in the table is “OK,” the combination of verbs represents a valid condition. If an entry in the table is “Error,” the combination of verbs represents an incorrect condition and an error code is returned to the LUA application program.

© Copyright IBM Corp. 1994, 1996

Table 3. RUI Verb Conditions

	In-Progress Commands						
Incoming Commands	**No Current Session**	**RUI_INIT**	**RUI_TERM**	**RUI_WRITE**	**RUI_READ**	**RUI_PURGE**	**RUI_BID**
RUI_INIT	OK	Error	Error	Error	Error	Error	Error
RUI_TERM	Error	OK	Error	OK	OK	OK	OK
RUI_WRITE	Error	Error	Error	OK (See Note 1)	OK	OK	OK
RUI_READ	Error	Error	Error	OK	OK (See Note 2)	OK	OK
RUI_PURGE	Error	Error	Error	OK	OK	Error	OK
RUI_BID	Error	Error	Error	OK	OK	OK	Error

Notes:

1. The RUI permits a maximum of two active **RUI_WRITE** verbs per session at the same time. The active **RUI_WRITE** verbs must be for different session flows. Four session flows are possible:

   SSCP-LU expedited
   SSCP-LU normal
   LU-LU expedited
   LU-LU normal

2. The RUI permits a maximum of four active **RUI_READ** verbs per session at the same time. The active **RUI_READ** verbs must be for different session flows.

Chapter 10. Writing LUA Application Programs

This chapter describes some of the aspects of writing LUA application programs. It includes the following topics:

Calling the dynamic link libraries (DLLs)
Using the DLL interface

Calling the Dynamic Link Libraries

The LUA contains one main DLL, **WINRUI32.DLL**, for the RUI verbs. An LUA application program calls this DLL to issue verbs.

The LUA application program sets selected fields in a verb control block and calls the RUI, passing a pointer to the verb control block. The fields in the verb control block define the requested action to the LUA. The LUA modifies fields in the verb control block to indicate the results of the action before the LUA returns control to the application program. The application program can then use the returned parameters from the verb control block in subsequent processing.

Procedure Entry Points

WINRUI32.DLL defines the following procedure entry points:

WinRUI

Provides asynchronous message notification for all RUI verbs.

Syntax

```
int WINAPI WinRUI (HWND,
                   LUA_VERB_RECORD );
```

Returned Values

The function returns a value indicating whether the request was accepted by the RUI for processing. A returned value of 0 indicates that the request was accepted and will be processed. A value other than 0 indicates an error. Possible error codes are as follows:

WLUAINVALIDHANDLE
The window handle provided is not valid.

The value returned in **lua_flag2.async** indicates whether asynchronous notification will occur. If the flag is set (nonzero), asynchronous notification will occur through a message posted to the application's message queue. If the flag is not set, the request completed synchronously. Examine the primary return code and secondary return code for any error conditions.

Usage Notes

Upon completion of the verb, the application's window *hWind* receives the message returned by **RegisterWindowMessage** with **"WinRUI"** as the input string. The **lParam** argument contains the address of the VCB being posted as complete. The *wParam* argument is undefined. It is possible for the request to be accepted for processing (the function call returns 0), but rejected later with a primary return code

© Copyright IBM Corp. 1994, 1996

and secondary return code set in the VCB. Examine the primary return code and secondary return code for any error conditions.

If the application calls **WinRUI** without first initializing the session using **WinRUIStartup**, an error is returned.

See also: "RUI()" on page 190.

WinRUICleanup()

Terminates and deregisters an application from the RUI API.

Syntax

BOOLWINAPI WinRUICleanup (void);

Returned Values

The return value indicates success or failure of the deregistration. If the value is not , the application was successfully deregistered. If the value is , the application was not deregistered.

Usage Notes

Use **WinRUICleanup** to deregister the RUI API, for example, to free up resources allocated to the specific application.

If **WinRUICleanup** is called while LUs are in session (**RUI_TERM** not issued), the cleanup code issues **RUI_TERM** close type ABEND for the application for all open sessions. **See also:** “WinRUIStartup()” on page 189.

WinRUIGetLastInitStatus()

This function provides a way for an application to determine the status of an **RUI_INIT** so that the application can determine whether the **RUI_INIT** should be timed out. Use this call to initiate the reporting of status, terminate the reporting of status, or find the current status. For details, see the **Usage Notes** section.

Syntax

```
int WINAPI WinRUIGetLastInitStatus (DWORD dwSid,
                                    HANDLE  hStatusHandle,
                                    DWORD  dwNotifyType,
                                    BOOL  bClearPrevious);
```

Parameter	Description
dwSid	Session identifier of the session for which status will be determined. If the value is , *hStatusHandle* is used to report status on all sessions. The *lua_sid* parameter in the **RUI_INIT** VCB is valid as soon as the call to **RUI**() or **WinRUI**() for the **RUI_INIT** returns.
hStatusHandle	A handle used for signaling the application that the status for the session has changed. Can be a window handle, an event handle, or NULL; *dwNotifyType* must be set accordingly: If *hStatusHandle* is a window handle, status is sent to the application through a window message. The program obtains the message from **RegisterWindowMessage** using the string **WinRUI**. The parameter wParam contains the session status (see **Return Codes**). Depending on the value of *dwNotifyType*, **lParam** contains either the RUI session ID of the session, or the value of **lua_correlator** from the **RUI_INIT** verb. If *hStatusHandle* is an event handle, when the status for the session specified by *dwSid* changes, the event is put into the signaled state. The application must then make a further call to **WinRUIGetLastInitStatus()** to find out the new status. The event should not be the same as one used for signaling completion of any **RUI** verb. If *hStatusHandle* is NULL, the status of the session specified by *dwSid* is returned in the return code. In this case, *dwSid* must not be 0 unless *bClearPrevious* is TRUE. If *hStatusHandle* is NULL, *dwNotifyType* is ignored.
dwNotifyType	The type of indication required. This determines the contents of the **lParam** of the window message and how **WinRUIGetLastInitStatus()** interprets *hStatusHandle*. Permitted values are: WLUA_NTFY_EVENT The *hStatusHandle* parameter contains an event handle.

WLUA_NTFY_MSG_CORRELATOR
The *hStatusHandle* parameter contains a window handle and the **lParam** of the returned window message should contain the LUA correlator and RUI.

WLUA_NTFY_MSG_SID
The *hStatusHandle* parameter contains a window handle and the **lParam** of the returned window message should contain the LUA session identifier.

bClearPrevious If TRUE, status messages are no longer sent for the session identified by *dwSid*. If *dwSid* is , status messages are no longer sent for any session. If *bClearPrevious* is TRUE, *hStatusHandle* and *dwNotifyType* are ignored.

Return Codes

WLUASYSNOTREADY
Personal Communications is not running.

WLUANTFYINVALID
The *dwNotifyType* parameter is not valid.

WLUAINVALIDHANDLE
The *hStatusHandle* parameter does not contain a valid handle.

WLUALINKINACTIVE
The link to the host is not yet active.

WLUAPUINACTIVE
The link to the host is active, but no **ACTPU** has yet been received.

WLUAPUACTIVE
An **ACTPU** has been received.

WLUAPUREACTIVATED
The PU has been reactivated.

WLUAUINACTIVE
The link to the host is active, and an **ACTPU** has been received, but no **ACTLU** has been received.

WLUALUACTIVE
The LU is active.

WLUAUNKNOWN
The session is in an unknown status. (This is an internal error.)

WLUASIDINVALID
The SID specified does not match any known by the **RUI**.

WLUASIDZERO
The *hStatusHandle* parameter is NULL and *bClearPrevious* is FALSE, but *dwSid* is .

WLUAGLOBALHANDLER
The *dwSid* parameter is , and messages from all sessions will be notified. (This is a normal return code, not an error.)

Remarks

This extension is intended to be used either with a window handle or an event handle to enable asynchronous notification of status changes, but it can also be used on its own to find out the current status of a session.

To use this extension with a window handle, you can implement it in one of two ways as follows:

WinRUIGetLastInitStatus(Sid,Handle,WLUA_NTFY_MSG_CORRELATOR,FALSE);

or

WinRUIGetLastInitStatus(Sid,Handle,WLUA_NTFY_MSG_SID<FALSE);

With this implementation, changes in status are reported by a window message sent to the window handle specified. If WLUA_NTFY_MSG_CORRELATOR is specified, the **lParam** field in the window message contains the **lua_correlator** field for the session. If WLUA_NFTY_MSG_SID is specified, the **lParam** field in the window message contains the LUA session identifier for the session.

When the extension has been used with a window handle, use the following command to cancel the reporting of status:

WinRUIGetLastInitStatus(Sid,NULL, ,TRUE);

For this implementation, note that if *Sid* is nonzero, status is only reported for that session. If *Sid* is , status is reported for all sessions.

To use this extension with an event handle, implement it as follows:

WinRUIGetLastInitStatus(Sid,Handle,WLUA_NOTIFY_EVENT,FALSE);

The event whose handle is given will be signaled when a change in state occurs. Because no information is returned when an event is signaled, the following call must be issued to find out the status:

Statu = WinRUIGetLastInitStatus(Sid,NULL, , ,FALSE);

In this case, a *Sid* must be specified.

When the extension has been used with an event handle, use the following command to cancel the reporting of status:

WinRUIGetLastInitStatus(Sid,NULL, ,TRUE);

To use this extension to query the current status of a session, it is not necessary to use an event or window handle. Instead, use the following command:

Status = WinRUIGetLastInitStatus(Sid,NULL, , ,FALSE);

WinRUIStartup()

Enables an application to specify the required version of the RUI API and to retrieve details of the API.

Syntax

```
int WINAPI WinRUIStartup (WORD wVersionRequired,
                          luadata );
```

Parameter	Description
wVersionRequired	Specifies the version of RUI API support required. The high-order byte specifies the minor version (revision) number; the low-order byte specifies the major version number.
luadata	Returns version of RUI implementation.

Returned Values

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

WLUAVERNOTSUPPORTED
: The version of RUI API support requested is not provided by this particular RUI API.

WLUAINVALID
: The version requested could not be determined.

Usage Notes

This call is intended to aid in compatibility with future versions of the API. The current version is 1.0. **See also:** “WinRUICleanup()” on page 185.

RUI()

Provides event notification for all **RUI** verbs.

Syntax

void WINAPI RUI (LUA_VERB_RECORD);

Returned Values

The value returned in *lua_prim_rc* indicates whether asynchronous notification will occur. If the field is set to LUA_IN_PROGRESS, asynchronous notification will occur through event signaling. If the flag is not LUA_IN_PROGRESS, the request completed synchronously. Examine the primary return code and secondary return code for any errors.

Usage Notes

The application must provide a handle to an event in the *lua_post_handle* parameter of the verb control block. The event must be in the not-signaled state.

When the asynchronous operation is complete, the application is notified by the signaling of the event. Upon signaling of the event, examine the primary return code and secondary return code for any error conditions. **See also:** “WinRUI” on page 183.

GetLuaReturnCode()

Converts the primary and secondary return codes in the VCB to a printable string. This function provides a standard set of error strings for use by LUA applications.

Syntax

```
int WINAPI GetLuaReturnCode (lua_common  vcb,
                             UINT  buffer_length,
                             unsigned char  buffer_addr);
```

Parameter	Description
vcb	Supplied parameter; specifies the address of the verb control block.
buffer_length	Supplied parameter; specifies the length of the buffer pointed to by buffer_addr. The recommended length is 256.
buffer_addr	Supplied/returned parameter; specifies the address of the buffer that will hold the formatted, null-terminated string; the length of the string in the specified buffer.

Returned Values

x2 1
The parameters are not valid; the function could not read from the specified verb control block or could not write to the specified buffer.

x2 2
The specified buffer is too small.

Usage Notes

The descriptive error string returned in *buffer_addr* does not terminate with a new line character (/**n**).

Call Examples

The following examples show how to call **WINRUI32.DLL**. The header file for this DLL is **WINLUA.H**. This example calls the **RUI DLL** to issue an **RUI** verb from a program:

```
#include "WINLUA.H"                                  / LUA C include file for
                                                       the LUA Application. /

. . .

. . .

example()
{
LUA_VERB_RECORD      VerbRecord;                      / Declare VerbRecord as a verb
                                                      control block using the
                                                      TYPEDEF in WINLUA.H /

. . .

WINRUI((LUA_VERB_RECORD  ) &VerbRecord);             / Call the RUI API /
. . .
}
```

Using the DLL Interface

The interface supplies the following types of dynamic linking to the DLLs:

Load-time linking
Run-time linking

Load-Time Linking

To link to LUA at load time, you must link in the **WINRUI32.LIB**. The **WINRUI32.LIB** library contains entry-point linkage information for all LUA API entry points.

Run-Time Linking

Run-time linking permits your application program to link conditionally to a DLL. With run-time linking, your LUA application program issues **LoadLibrary** and **GetProcAddress** to obtain linkage to the LUA procedure entry point. To do this, your LUA application program uses the DLL name and the procedure entry-point name.

FreeLibrary can be used by the application after using the RUI interface.

Multiple Processes

An LUA application program is restricted to a single process. Different instances of the same LUA application program can start in different processes, but each application program must use a different LUA LU.

In addition, a single process can comprise multiple LUA application programs, each with its own LUA LU.

Multiple Threads

A single LUA application program can use multiple threads to issue verbs. This lets you issue multiple verbs simultaneously from a single LUA application program. Different instances of the same LUA application program can start in different threads, but each application program can use a different LUA LU.

Note: After an LUA application program issues a verb, it should not change any part of the verb control block until the verb is complete. The RUI uses only the application copy of the verb control block. See “LUA Verb Postings” for additional information.

LUA Verb Postings

LUA verbs complete synchronously or asynchronously. Synchronous verb completion means that when the RUI returns to the LUA application program after a call to LUA, all processing for that verb is finished and the asynchronous posting method is not used. Because of timing conditions, a verb can complete asynchronously, but all processing is completed by the time LUA returns to the LUA application program. Asynchronous verb completion means that LUA uses the posting method to notify the application program when processing completes, either successfully or unsuccessfully.

An LUA application program can be notified in one of the following ways when a verb completes asynchronously:

The LUA application program uses the **lua_flag2_async** and **lua_prim_rc** parameters to determine the verb processing state.

The application specifies a Windows 95 event handle in the **lua_post_handle** parameter. This is set when the verb is complete.

Converting to EBCDIC from ASCII

Typically, all messages sent to the host are in EBCDIC, and the PLU assumes that the messages are in EBCDIC. For example, a PLU name that appears in a **BIND** should be an EBCDIC string. An LUA application program that stores strings in ASCII should convert the strings to EBCDIC before the strings are sent in SNA messages.

Whether an LUA application program needs to convert application data depends on a private agreement between the partner application programs. If your LUA application program communicates with a node that normally uses EBCDIC, you should convert your ASCII data to EBCDIC where appropriate.

Conversion of ASCII to EBCDIC (or vice versa) can be done by the convert verbs described in Chapter 7, “Procedure Entry Points and Common Services Verbs (CSV).”

Chapter 11. RUI Verbs

This chapter contains the following information:

Details of the LUA common control block structure
A description of each LUA verb for all LUA verbs and the LUA verb records

The following information is provided for each LUA verb:

The purpose of the verb.

Parameters supplied to and returned by LUA. The description of each parameter includes information on the valid values for that parameter, and any additional information necessary.

Interactions with other verbs.

Additional information describing the use of the verb.

Note: Parameters marked as *reserved* should always be set to zero.

LUA Verb Control Block Format

The verb control block consists of:

lua_common, used for all verbs, and described in "Common Verb Header" on page 196.

specific, used only for the **RUI_BID** verb (described in "RUI_BID Data Structure" on page 200).

The structure is defined as follows:

```
typedef struct lua_verb_record
{
 LUA_COMMON              common;                         / The common verb header        /
 union
 {
    unsigned char     lua_peek_data[12];          / field specific to RUI_BID /
   specified
 }
} LUA_VERB_RECORD;
```

© Copyright IBM Corp. 1994, 1996

Common Verb Header

The Personal Communications LUA uses a generic *common verb header* to transport all incoming and outgoing data. The fields in the verb control block are defined as follows:

```
typedef struct lua_common
{
 unsigned short      lua_verb;                 / LUA_VERB_RUI                    /
 unsigned short      lua_verb_length;          / VCB length                      /
 unsigned short      lua_prim_rc;              / primary return code             /
 unsigned long       lua_sec_rc;               / secondary return code           /
 unsigned short      lua_opcode;               / verb opcode                     /
 unsigned long       lua_correlator;           / verb correlator                 /
 unsigned char       lua_luname[8];            / local LU name                   /
 unsigned short      lua_extension_list_offset;
                                               / reserved                        /
 unsigned short      lua_cobol_offset;         / reserved                        /
 unsigned long       lua_sid;                  / session ID                      /
 unsigned short      lua_max_length;           / max buffer length               /
 unsigned short      lua_data_length;          / actual data length              /
 unsigned char       lua_data_ptr;             / data pointer                    /
 unsigned long       lua_post_handle;          / post handle                     /
 LUA_TH              lua_th;                   / TH structure                    /
 unsigned char       lua_rh;                   / message RH                      /
 unsigned char       lua_flag1;                / application message flag        /
 unsigned char       lua_message_type;         / SNA message type                /
 unsigned char       lua_flag2;                / LUA message flag                /
 unsigned char       lua_resv56[7];            / reserved                        /
 unsigned char       lua_encr_decr_option;/ cryptography                         /
} LUA_COMMON;

typedef struct lua_th
{
 unsigned char       reserv1;                  / reserved                        /
 unsigned char       daf;                      / DAF                             /
 unsigned char       oaf;                      / OAF                             /
 unsigned char       snf[2];                   / SNF                             /
} LUA_TH;
```

lua_verb Identifies this as an **LUA** verb: always set to **LUA_VERB_RUI**.

lua_verb_length Length of the verb control block.

lua_prim_rc Primary return code set by LUA.

lua_sec_rc Secondary return code set by LUA.

lua_opcode Verb operation code, which identifies the **LUA** verb being issued.

lua_correlator A 4-byte correlator, which you can use to correlate this verb with other processing in your application. LUA does not use this parameter.

lua_luname The local LU name used by the LUA session (in ASCII). This can be an LU name or an LU pool name; see "RUI_INIT" on page 206 for more information.

lua_sid The session ID of the LUA session on which this verb is issued.

lua_max_length The length of the buffer used to receive data.

lua_data_length The length of the data to be sent, or the actual length of data received.

lua_data_ptr A pointer to the data to be sent, or the data buffer to receive data.

lua_th.flags Specifies the flags set in the transmission header. (Refer to *Systems Network Architecture: Formats* for more information.) It can be one or more of the following values ORed together:

LUA_FID	Format Identification type 2
LUA_MPF	Segmenting mapping field
LUA_BBIU	Begin BIU
LUA_EBIU	End BIU
LUA_ODAI	OAF-DAF Assignor Indicator
LUA_EFI	Expedited Flow Indicator

lua_th.daf DAF (Destination address field).

lua_th.oaf OAF (Originating address field).

lua_th.snf Sequence Number Field.

lua_rh Specifies the RH (request/response header) of the message sent or received. (Refer to *Systems Network Architecture: Formats* for more information.) This can be one or more of the following values ORed together:

LUA_RRI	Request-Response Indicator
LUA_RH_FMD	RU category: FMI data segment
LUA_RH_NC	RU category: Network control
LUA_RH_DFC	RU category: Data flow control
LUA_RH_SC	RU category: Session control
LUA_FI	Format Indicator
LUA_SDI	Sense Data Included Indicator
LUA_BCI	Begin Chain Indicator
LUA_ECI	End Chain Indicator
LUA_DR1I	Definite Response 1 Indicator
LUA_DR2I	Definite Response 2 Indicator
LUA_RI	Exception Response Indicator (for a request), or Response Type Indicator (for a response)
LUA_QRI	Queued Response Indicator
LUA_PI	Pacing Indicator

LUA_BBI	Begin Bracket Indicator
LUA_EBI	End Bracket Indicator
LUA_CDI	Change Direction Indicator
LUA_CSI	Code Selection Indicator
LUA_EDI	Enciphered Data Indicator
LUA_PDI	Padded Data Indicator

lua_flag1 Specifies flags for messages supplied by the application. (Refer to *Systems Network Architecture: Formats* for more information.) The flags can be one or more of the following values ORed together:

LUA_BID_ENABLE	Bid Enable Indicator
LUA_NOWAIT	No Wait For Data flag
LUA_SSCP_EXP	SSCP expedited flow
LUA_SSCP_NORM	SSCP normal flow
LUA_LU_EXP	LU expedited flow
LUA_LU_NORM	LU normal flow
LUA_CLOSE_ABEND	
LUA_RESERVE1	

lua_message_type The type of SNA message received by an **RUI_READ** verb (or indicated to an **RUI_BID** verb). This can be one the following values:

LUA_MESSAGE_TYPE_LU_DATA
LUA_MESSAGE_TYPE_SSCP_DATA
LUA_MESSAGE_TYPE_RSP
LUA_MESSAGE_TYPE_BID
LUA_MESSAGE_TYPE_BIND
LUA_MESSAGE_TYPE_BIS
LUA_MESSAGE_TYPE_CANCEL
LUA_MESSAGE_TYPE_CHASE
LUA_MESSAGE_TYPE_CLEAR
LUA_MESSAGE_TYPE_CRV
LUA_MESSAGE_TYPE_LUSTAT_LU
LUA_MESSAGE_TYPE_LUSTAT_SSCP
LUA_MESSAGE_TYPE_QC
LUA_MESSAGE_TYPE_QEC
LUA_MESSAGE_TYPE_RELQ
LUA_MESSAGE_TYPE_RQR
LUA_MESSAGE_TYPE_RTR
LUA_MESSAGE_TYPE_SBI
LUA_MESSAGE_TYPE_SHUTD
LUA_MESSAGE_TYPE_SIGNAL
LUA_MESSAGE_TYPE_SDT
LUA_MESSAGE_TYPE_STSN
LUA_MESSAGE_TYPE_UNBIND

lua_flag2 Specifies flags for messages returned by LUA. (Refer to *Systems Network Architecture: Formats* for more information.) The flags can be one or more of the following values ORed together:

LUA_BID_ENABLE	Bid Enable Indicator
LUA_ASYNC	Asynchronous verb completion flag
LUA_SSCP_EXP	SSCP expedited flow
LUA_SSCP_NORM	SSCP normal flow
LUA_LU_EXP	LU expedited flow
LUA_LU_NORM	LU normal flow

lua_encr_decr_option Cryptography option.

RUI_BID Data Structure

The following parameter is specific to and only supplied on the **RUI_BID** verb:

lua_peek_data Up to 12 bytes of data waiting to be read.

RUI_BID

The **RUI_BID** verb is used by the application to indicate when a received message is waiting to be read. This enables the application to determine what data, if any, is available before issuing the **RUI_READ** verb. When a message is available, the **RUI_BID** verb returns with details of the message flow on which it was received, the message type, the TH and RH of the message, and up to 12 bytes of message data. The main difference between **RUI_BID** and **RUI_READ** is that **RUI_BID** enables the application to check the data without removing it from the incoming message queue, so it can be left and accessed at a later stage. **RUI_READ** removes the message from the queue, so once the application has read the data it must process it.

Supplied Parameters

The application supplies the following parameters:

lua_verb LUA_VERB_RUI

lua_verb_length The length in bytes of the LUA verb record. Set this to

sizeof(struct LUA_COMMON) + 12.

lua_opcode LUA_OPCODE_RUI_BID

lua_correlator Optional. A 4-byte value, which you can use to correlate this verb with other processing within your application. LUA does not use or change this information.

lua_luname The name in ASCII of the local LU used by the session. This must match the LU name of an active LUA session.

This parameter is required only if the **lua_sid** parameter is zero. If a session ID is supplied in **lua_sid**, LUA does not use this parameter.

This parameter must be 8 bytes long; pad on the right with spaces, 0x20, if the name is shorter than 8 characters.

lua_sid The session ID of the session. This must match a session ID returned on a previous **RUI_INIT** verb. This parameter is optional; if you do not specify the session ID, you must specify the LU name for the session in the **lua_luname** parameter.

lua_post_handle This is a 4-byte handle that is used to post the completion of asynchronous verbs. It is a Windows 95 event handle.

Returned Parameters

The following parameter will always be returned:

lua_flag2 This is only set to LUA_ASYNC if the verb completed asynchronously.

Other returned parameters depend on whether the verb completed successfully; see the following sections.

Successful Execution

If the verb completed successfully, the following parameters are returned:

lua_prim_rc LUA_OK

lua_sid If the application specified the **lua_luname** parameter when issuing this verb, rather than specifying the session ID, LUA supplies the session ID.

lua_max_length The number of bytes of data in the received message.

lua_data_length The number of bytes of data returned in the **lua_peek_data** parameter; from 0 to 12.

lua_th Information from the transmission header (TH) of the received message.

lua_rh Information from the request/response header (RH) of the received message.

lua_message_type Message type of the received message, which will be one of the following values:

LUA_MESSAGE_TYPE_LU_DATA
LUA_MESSAGE_TYPE_SSCP_DATA
LUA_MESSAGE_TYPE_RSP
LUA_MESSAGE_TYPE_BID
LUA_MESSAGE_TYPE_BIND
LUA_MESSAGE_TYPE_BIS
LUA_MESSAGE_TYPE_CANCEL
LUA_MESSAGE_TYPE_CHASE
LUA_MESSAGE_TYPE_CLEAR
LUA_MESSAGE_TYPE_CRV
LUA_MESSAGE_TYPE_LUSTAT_LU
LUA_MESSAGE_TYPE_LUSTAT_SSCP
LUA_MESSAGE_TYPE_QC
LUA_MESSAGE_TYPE_QEC
LUA_MESSAGE_TYPE_RELQ
LUA_MESSAGE_TYPE_RTR
LUA_MESSAGE_TYPE_SBI
LUA_MESSAGE_TYPE_SHUTD
LUA_MESSAGE_TYPE_SIGNAL
LUA_MESSAGE_TYPE_SDT
LUA_MESSAGE_TYPE_STSN
LUA_MESSAGE_TYPE_UNBIND

lua_flag2 One of the following flags will be set to indicate which message flow the data was received on:

LUA_SSCP_EXP
SSCP expedited flow

LUA_LU_EXP
LU expedited flow

LUA_SSCP_NORM
SSCP normal flow

LUA_LU_NORM
LU normal flow

lua_peek_data The first 12 bytes of the message data (or all of the message data if it is shorter than 12 bytes).

Verb Canceled

The following return codes indicate that the verb did not complete successfully because it was canceled by another verb:

lua_prim_rc LUA_CANCELLED

lua_sec_rc LUA_TERMINATED
An **RUI_TERM** verb was issued while this verb was pending.

Parameter Check

The following return codes indicate that the verb did not complete successfully because a supplied parameter was in error:

lua_prim_rc LUA_PARAMETER_CHECK

lua_sec_rc Possible values:

LUA_BID_ALREADY_ENABLED
The **RUI_BID** verb was rejected because a previous **RUI_BID** verb was already outstanding. Only one **RUI_BID** can be outstanding at a time.

LUA_RESERVED_FIELD_NOT_ZERO
A reserved field in the verb record, or a parameter that is not used by this verb, was set to a nonzero value.

LUA_VERB_LENGTH_INVALID
The value of the **lua_verb_length** parameter was less than the length of the verb record required for this verb.

State Check

The following return codes indicate that the verb was issued in a session state in which it was not valid:

lua_prim_rc LUA_STATE_CHECK

lua_sec_rc LUA_NO_RUI_SESSION
An **RUI_INIT** verb has not yet completed successfully for this session, or a session outage has occurred.

Unsuccessful Execution

The following return codes indicate that the verb record supplied was valid, but the verb did not complete successfully:

lua_prim_rc LUA_UNSUCCESSFUL

lua_sec_rc LUA_INVALID_PROCESS
The application instance that issued this verb was not the same as the one that issued the **RUI_INIT** verb for this session.

Negative Response Sent to Host

The following return code indicates that Personal Communications detected an error in the data received from the host. Instead of passing the received message to the application on an **RUI_READ** verb, Personal Communications discards the message (and the rest of the chain if it is in a chain), and sends a negative response to the host. LUA informs the application on a subsequent **RUI_READ** or **RUI_BID** verb that a negative response was sent.

lua_prim_rc LUA_NEGATIVE_RSP

lua_sec_rc The secondary return code contains the sense code sent to the host on the negative response. See “SNA Layers” on page 166 for information on interpreting the sense code values that can be returned.

A zero secondary return code indicates that, following a previous **RUI_WRITE** of a negative response to a message in the middle of a chain, Personal Communications has now received and discarded all messages from this chain.

Other Conditions

The following primary and secondary return codes indicate that the verb did not complete successfully for other reasons:

lua_prim_rc LUA_SESSION_FAILURE
The session has been brought down.

lua_sec_rc Possible values:

LUA_LU_COMPONENT_DISCONNECTED
The LUA session has failed because of a problem with the communications link or with the host LU.

LUA_RUI_LOGIC_ERROR
This return code indicates one of the following things:

The host system has violated SNA protocols.
An internal error was detected within LUA.

Attempt to reproduce the problem with tracing active, and check that the host is sending correct data.

lua_prim_rc LUA_INVALID_VERB
Either the **lua_verb** parameter or the **lua_opcode** parameter was not valid. The verb did not execute.

lua_prim_rc LUA_UNEXPECTED_DOS_ERROR
An operating system error occurred, such as resource shortage.

lua_sec_rc This value is the operating system return code. Check your operating system documentation for the meaning of this return code.

Interaction with Other Verbs

The **RUI_INIT** verb must complete successfully before this verb can be issued.

Only one **RUI_BID** can be outstanding at any one time. After the **RUI_BID** verb has completed successfully, it can be reissued by setting the **lua_flag1** to LUA_BID_ENABLE on a subsequent **RUI_READ** verb. If the verb is to be reissued in this way, the application program must not free or modify the storage associated with the **RUI_BID** verb record.

If a message arrives from the host when an **RUI_READ** and an **RUI_BID** are both outstanding, the **RUI_READ** completes and the **RUI_BID** is left in progress.

Usage and Restrictions

Each message that arrives will only be bid once. Once an **RUI_BID** verb has indicated that data is waiting on a particular session flow, the application should issue the **RUI_READ** verb to receive the data. Any subsequent **RUI_BID** will not report data arriving on that session flow until the message that was bid has been accepted by issuing an **RUI_READ** verb.

In general, the **lua_data_length** parameter returned on this verb indicates only the length of data in **lua_peek_data**, not the total length of data on the waiting message (except when a value of less than 12 is returned). The **lua_max_length** parameter returns the number of bytes in the received message. The application should ensure that the data length on the **RUI_READ** verb that accepts the data is sufficient to contain the message.

RUI_INIT

The **RUI_INIT** verb establishes the SSCP-LU session for a given LUA LU.

Supplied Parameters

The application supplies the following parameters:

lua_verb LUA_VERB_RUI

lua_verb_length The length in bytes of the LUA verb record. Set this to:

Sizeof(struct LUA_COMMON) when using the standard format of **RUI_INIT** in which the LU name is specified

Sizeof(LUA_VERB_RECORD) when using the extended format of **RUI_INIT** in which either the LU name or the PU name and LU number are specified

lua_opcode LUA_OPCODE_RUI_INIT

lua_correlator Optional. A 4-byte value, which you can use to correlate this verb with other processing within your application. LUA does not use or change this information.

lua_luname The name in ASCII of the local LU for which you want to start the session. This must match an LUA LU name configured for Personal Communications. The name is used as follows:

If the name is not zero pool, Personal Communications attempts to start the session using this LU.

This field is an 8-byte ASCII string, padded with trailing space (0x20) characters if necessary. **RUI_INIT**, set this field to all zeros,

lua_post_handle This is a 4-byte handle that is used to post the completion of asynchronous verbs. It is a Windows 95 event handle.

lua_flag1 The application should set this to LUA_ASYNC_STATUS, to receive RUI_INIT_STATUS indications from Personal Communications when processing the **RUI_INIT** verb. (RUI_INIT_STATUS messages are described in “RUI_INIT_STATUS” on page 213.)

lua_encr_decr_option

Session-level cryptography option. Personal Communications accepts the following two values:

0 Session-level cryptography is not used.
128 Encryption and decryption are performed by the application program.

Any other value will result in the return code LUA_ENCR_DECR_LOAD_ERROR. (Values in the range 1 to 127, indicating user-defined encryption and decryption routines, are supported by OS/2 Communications Manager/2's LUA implementation but not by Personal Communications.)

Returned Parameters

The following parameter will always be returned:

lua_flag2 This is only set to LUA_ASYNC if the verb completed asynchronously.

Note: **RUI_INIT** will always complete asynchronously, unless it returns an error such as LUA_PARAMETER_CHECK.

Other returned parameters depend on whether the verb completed successfully; see the following sections.

Successful Execution

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

lua_prim_rc LUA_OK

lua_sid A session ID for the new session. This can be used by subsequent verbs to identify this session.

lua_luname The name of the local LU used by the session. This is required if the application specified an LU pool and needs to know which LU in the pool has been used.

Verb Canceled

The following return codes indicate that the verb did not complete successfully because it was canceled by another verb:

lua_prim_rc LUA_CANCELLED

lua_sec_rc LUA_TERMINATED
An **RUI_TERM** verb was issued before the **RUI_INIT** had completed.

Parameter Check

The following return codes indicate that the verb did not complete successfully because a supplied parameter was in error:

lua_prim_rc LUA_PARAMETER_CHECK

lua_sec_rc Possible values:

LUA_INVALID_LUNAME
The **lua_luname** parameter could not be found. Check that the LU name or LU pool name was defined in the *Personal Communications System Management Programming* API.

LUA_RESERVED_FIELD_NOT_ZERO
A reserved field in the verb record, or a parameter that is not used by this verb, was set to a nonzero value.

LUA_VERB_LENGTH_INVALID
The value of the **lua_verb_length** parameter was less than the length of the verb record required for this verb.

State Check

The following return codes indicate that the verb was issued in a session state in which it was not valid:

lua_prim_rc LUA_STATE_CHECK

lua_sec_rc LUA_DUPLICATE_RUI_INIT
The **lua_luname** parameter specified an LU name or LU pool name that is already in use by this application (or for which this application already has an **RUI_INIT** verb in progress).

Unsuccessful Execution

The following return codes indicate that the verb record supplied was valid, but the verb did not complete successfully:

lua_prim_rc LUA_UNSUCCESSFUL

lua_sec_rc Possible values:

LUA_COMMAND_COUNT_ERROR
The verb specified the name of an LU pool, or the name of an LU in a pool, but all LUs in the pool are in use.

LUA_ENCR_DECR_LOAD_ERROR
The verb specified a value for **lua_encr_decr_option** other than 0 or 128.

LUA_INVALID_PROCESS
The LU specified by the **lua_luname** parameter is in use by another process.

LUA_LINK_NOT_STARTED
The link to the host has not been started.

The following values for **lua_sec_rc** are Personal Communications sense codes, and can be returned if **lua_prim_rc** is LUA_UNSUCCESSFUL (these values reflect the state of the LU):

X1 2 **ACTPU** has not been received. **RUI_INIT** will not activate the PU.

X1 1 **ACTPU** has not been received. **RUI_INIT** will activate the PU.

X1 11 **ACTPU** has been received. **ACTLU** has not been received. SSCP does not support self-defining dependent LU (SSDLU). **RUI_INIT** will activate the LU.

X1 12 **ACTPU** has been received. **ACTLU** has not been received. SSCP does support SSDLU. **RUI_INIT** will activate the LU.

Other Conditions

The following primary and secondary return codes indicate that the verb did not complete successfully for other reasons:

lua_prim_rc LUA_SESSION_FAILURE
The session has been brought down.

lua_sec_rc	LUA_LU_COMPONENT_DISCONNECTED The LUA session has failed because of a problem with the communications link or with the host LU.
lua_prim_rc	LUA_INVALID_VERB Either the **lua_verb** parameter or the **lua_opcode** parameter was not valid. The verb did not execute.
lua_prim_rc	LUA_UNEXPECTED_DOS_ERROR An operating system error occurred, such as resource shortage.
lua_sec_rc	This value is the operating system return code. Check your operating system documentation for the meaning of this return code.

Interaction with Other Verbs

This verb must be the first LUA verb issued for the session. Until this verb has completed successfully, the only other LUA verb that can be issued for this session is **RUI_TERM** (which will terminate a pending **RUI_INIT**). All other verbs issued on this session must identify the session using one of the following parameters from this verb.

The session ID is returned to the application in the **lua_sid** parameter.
The LU name is supplied by the application in the **lua_luname** parameter.

Usage and Restrictions

The **RUI_INIT** verb completes after an **ACTLU** is received from the host. If necessary, the verb waits indefinitely. If an **ACTLU** has already been received prior to the **RUI_INIT** verb, LUA sends a **NOTIFY** to the host to inform it that the LU is ready for use.

Note: Neither the **ACTLU** nor **NOTIFY** is visible to the LUA application.

Once the **RUI_INIT** verb has completed successfully, this session uses the LU for which the session was started. No other LUA session (from this or any other application) can use the LU until the **RUI_TERM** verb is issued.

RUI_PURGE

The **RUI_PURGE** verb cancels a previous **RUI_READ**. An **RUI_READ** can wait indefinitely if it is sent without setting **lua_flag1** to LUA_NO WAIT (the immediate return option), and no data is available on the specified flow; **RUI_PURGE** forces the waiting verb to return (with the primary return code CANCELLED).

Supplied Parameters

The application supplies the following parameters:

lua_verb LUA_VERB_RUI

lua_verb_length The length in bytes of the LUA verb record. Set this to sizeof(struct LUA_COMMON).

lua_opcode LUA_OPCODE_RUI_PURGE

lua_correlator Optional. A 4-byte value, which you can use to correlate this verb with other processing within your application. LUA does not use or change this information.

lua_luname The name in ASCII of the local LU used by the session. This must match the LU name of an active LUA session.

This parameter is required only if the **lua_sid** parameter is zero. If a session ID is supplied in **lua_sid**, LUA does not use this parameter.

This parameter must be 8 bytes long; pad on the right with spaces, 0x20, if the name is shorter than 8 characters.

lua_sid The session ID of the session. This must match a session ID returned on a previous **RUI_INIT** verb.

This parameter is optional; if you do not specify the session ID, you must specify the LU name for the session in the **lua_luname** parameter.

lua_data_ptr A pointer to the **RUI_READ** LUA_VERB_RECORD that is to be purged.

lua_post_handle This is a 4-byte handle that is used to post the completion of asynchronous verbs. It is a Windows 95 event handle.

Returned Parameters

The following parameter will always be returned:

lua_flag2 This is only set to LUA_ASYNC if the verb completed asynchronously.

Other returned parameters depend on whether the verb completed successfully; see the following sections.

Successful Execution

If the verb completed successfully, the following parameters are returned:

lua_prim_rc LUA_OK

lua_sid If the application specified the **lua_luname** parameter when issuing this verb, rather than specifying the session ID, LUA supplies the session ID.

Verb Canceled

The following return codes indicate that the verb did not complete successfully because it was canceled by another verb:

lua_prim_rc LUA_CANCELLED

lua_sec_rc LUA_TERMINATED
An **RUI_TERM** verb was issued while this verb was pending.

Parameter Check

The following return codes indicate that the verb did not complete successfully because a supplied parameter was in error:

lua_prim_rc LUA_PARAMETER_CHECK

lua_sec_rc Possible values:

LUA_BAD_DATA_PTR
The **lua_data_ptr** parameter was set to zero.

LUA_RESERVED_FIELD_NOT_ZERO
A reserved field in the verb record, or a parameter that is not used by this verb, was set to a nonzero value.

LUA_VERB_LENGTH_INVALID
The value of the **lua_verb_length** parameter was less than the length of the verb record required for this verb.

State Check

The following return codes indicate that the verb was issued in a session state in which it was not valid:

lua_prim_rc LUA_STATE_CHECK

lua_sec_rc Possible values:

LUA_SEC_RC_OK
A previous **RUI_PURGE** verb is still in progress on this session.

LUA_NO_RUI_SESSION
An **RUI_INIT** verb has not yet completed successfully for this session, or a session outage has occurred.

Unsuccessful Execution

The following return codes indicate that the verb record supplied was valid, but the verb did not complete successfully:

lua_prim_rc LUA_UNSUCCESSFUL

lua_sec_rc Possible values:

LUA_INVALID_PROCESS
The application instance that issued this verb was not the same as the one that issued the **RUI_INIT** verb for this session.

LUA_NO_READ_TO_PURGE
Either the **lua_data_ptr** parameter did not contain a pointer to an **RUI_READ** LUA_VERB_RECORD or the

RUI_READ verb completed before the **RUI_PURGE** verb was issued.

Other Conditions

The following primary and secondary return codes indicate that the verb did not complete successfully for other reasons:

lua_prim_rc LUA_SESSION_FAILURE
The session has been brought down.

lua_sec_rc Possible values:

LUA_LU_COMPONENT_DISCONNECTED
The LUA session has failed because of a problem with the communications link or with the host LU.

LUA_RUI_LOGIC_ERROR
This return code indicates one of the following things:

The host system has violated SNA protocols.
An internal error was detected within LUA.

Attempt to reproduce the problem with tracing active, and check that the host is sending correct data.

lua_prim_rc LUA_INVALID_VERB
Either the **lua_verb** parameter or the **lua_opcode** parameter was not valid. The verb did not execute.

lua_prim_rc LUA_UNEXPECTED_DOS_ERROR
An operating system error occurred, such as resource shortage.

lua_sec_rc This value is the operating system return code. Check your operating system documentation for the meaning of this return code.

Interaction with Other Verbs

This verb can only be used when an **RUI_READ** has been issued and is pending completion (that is, the primary return code is IN_PROGRESS). This verb should not be issued while another **RUI_PURGE** is in progress on this session.

RUI_INIT_STATUS

The application cannot issue **RUI_INIT_STATUS** indications. Personal Communications sends this indication to the application during **RUI_INIT** processing to provide information about the state of the LU-SSCP session.
RUI_INIT_STATUS indications are only sent if the application requests status information when issuing **RUI_INIT** (see “RUI_INIT” on page 206).

Supplied Parameters

The following parameters are set on the **RUI_INIT_STATUS** indication:

lua_verb LUA_VERB_RUI

lua_verb_length The length in bytes of the LUA verb record (set by Personal Communications to sizeof(LUA_COMMON)).

lua_opcode LUA_OPCODE_RUI_INIT_STATUS

lua_primary_rc Contains the information about the state of the LU-SSCP session. Possible values are:

LUA_LINK_INACTIVE
The link to the Host is not yet active.

LUA_PU_INACTIVE
ACTPU has not yet been received, or **DACTPU** has been received.

LUA_PU_ACTIVE
ACTPU has been received from the SSCP.

LUA_PU_REACTIVATED
ACTPU(COLD) has been received while the PU was active.

LUA_LU_INACTIVE
ACTLU has been rejected, or **DACTLU** has been received.

LUA_UNKNOWN
The LU-SSCP session is not active due to a Data Link Control link error.

lua_correlator Personal Communications uses the value specified on the **RUI_INIT** verb for this parameter.

lua_post_handle This is a 4-byte handle that is used to post the completion of asynchronous verbs. It is a Windows 95 event handle. Personal Communications uses the value specified on the **RUI_INIT** verb for this parameter.

RUI_READ

The **RUI_READ** verb receives data or status information sent from the host to the application's LU. You can specify a particular message flow (LU normal, LU expedited, SSCP normal, or SSCP expedited) from which to read data, or you can specify more than one message flow. You can have multiple **RUI_READ** verbs outstanding, provided that no two of them specify the same flow.

Supplied Parameters

The application supplies the following parameters:

lua_verb LUA_VERB_RUI

lua_verb_length The length in bytes of the LUA verb record. Set this to sizeof(struct LUA_COMMON).

lua_opcode LUA_OPCODE_RUI_READ

lua_correlator Optional. A 4-byte value, which you can use to correlate this verb with other processing within your application. LUA does not use or change this information.

lua_luname The name in ASCII of the local LU used by the session. This must match the LU name of an active LUA session.

This parameter is required only if the **lua_sid** parameter is zero. If a session ID is supplied in **lua_sid**, LUA does not use this parameter.

This parameter must be 8 bytes long; pad on the right with spaces, 0x20, if the name is shorter than 8 characters.

lua_sid The session ID of the session. This must match a session ID returned on a previous **RUI_INIT** verb.

This parameter is optional; if you do not specify the session ID, you must specify the LU name for the session in the **lua_luname** parameter.

lua_max_length The length of the buffer supplied to receive the data (see **lua_data_ptr**).

lua_data_ptr A pointer to the buffer supplied to receive the data.

lua_post_handle This is a 4-byte handle that is used to post the completion of asynchronous verbs. It is a Windows 95 event handle.

lua_flag1 The flags can be one or more of the following values ORed together:

Set LUA_NOWAIT if you want the **RUI_READ** verb to return immediately whether or not data is available to be read, or do not set it if you want the verb to wait for data before returning.

Set LUA_BID_ENABLE to reenable the most recent **RUI_BID** verb (equivalent to issuing **RUI_BID** again with exactly the same parameters as before), or do not set it if you do not want to reenable **RUI_BID**.

Note: Re-enabling the previous **RUI_BID** reuses the LUA_VERB_RECORD originally allocated and

does not permit the LUA_VERB_RECORD to be freed or modified.

Set one or more of the following flags to indicate which message flow to read data from:

LUA_SSCP_EXP
SSCP expedited flow

LUA_LU_EXP
LU expedited flow

LUA_SSCP_NORM
SSCP normal flow

LUA_LU_NORM
LU normal flow

If more than one flag is set, the highest-priority data available will be returned. The order of priorities (highest to lowest) is as follows:

1. SSCP expedited
2. LU expedited
3. SSCP normal
4. LU normal

The equivalent flag will be set in **lua_flag2** to indicate which flow the data was read from (see "Returned Parameters").

Returned Parameters

The following parameters will always be returned:

lua_flag2 LUA_ASYNC is set if the verb completes asynchronously (and not set if the verb completes synchronously).

LUA_BID_ENABLE is set if an **RUI_BID** was successfully reenabled (and not set if it was not reenabled).

Other returned parameters depend on whether the verb completed successfully; see the following sections.

Successful Execution or Truncated Data

If the verb executes successfully, LUA also returns the following parameters:

lua_prim_rc LUA_OK

The following parameters are returned if the verb completes successfully. They are also returned if the verb returns with truncated data because the **lua_data_length** parameter supplied was too small (see "Unsuccessful Execution" on page 218).

lua_sid If the application specified the **lua_luname** parameter when issuing this verb, rather than specifying the session ID, LUA supplies the session ID.

lua_data_length The length of the data received. LUA places the data in the buffer specified by **lua_data_ptr**.

lua_th Information from the transmission header (TH) of the received message.

lua_rh Information from the request/response header (RH) of the received message.

lua_message_type Message type of the received message, which will be one of the following values:

LUA_MESSAGE_TYPE_LU_DATA
LUA_MESSAGE_TYPE_SSCP_DATA
LUA_MESSAGE_TYPE_RSP
LUA_MESSAGE_TYPE_BID
LUA_MESSAGE_TYPE_BIND
LUA_MESSAGE_TYPE_BIS
LUA_MESSAGE_TYPE_CANCEL
LUA_MESSAGE_TYPE_CHASE
LUA_MESSAGE_TYPE_CLEAR
LUA_MESSAGE_TYPE_CRV
LUA_MESSAGE_TYPE_LUSTAT_LU
LUA_MESSAGE_TYPE_LUSTAT_SSCP
LUA_MESSAGE_TYPE_QC
LUA_MESSAGE_TYPE_QEC
LUA_MESSAGE_TYPE_RELQ
LUA_MESSAGE_TYPE_RTR
LUA_MESSAGE_TYPE_SBI
LUA_MESSAGE_TYPE_SHUTD
LUA_MESSAGE_TYPE_SIGNAL
LUA_MESSAGE_TYPE_SDT
LUA_MESSAGE_TYPE_STSN
LUA_MESSAGE_TYPE_UNBIND

lua_flag2 parameters This will be set to one of the following values, to indicate which message flow the data was received on:

LUA_SSCP_EXP
SSCP expedited flow

LUA_LU_EXP
LU expedited flow

LUA_SSCP_NORM
SSCP normal flow

LUA_LU_NORM
LU normal flow

Verb Canceled

The following return codes indicate that the verb did not complete successfully because it was canceled by another verb or by an internal error:

lua_prim_rc LUA_CANCELLED

lua_sec_rc Possible values:

LUA_PURGED
This **RUI_READ** verb has been canceled by an **RUI_PURGE** verb.

LUA_TERMINATED
An **RUI_TERM** verb was issued while this verb was pending.

Parameter Check

The following return codes indicate that the verb did not complete successfully because a supplied parameter was in error:

lua_prim_rc LUA_PARAMETER_CHECK

lua_sec_rc Possible values:

LUA_BAD_DATA_PTR
The **lua_data_ptr** parameter contained an incorrect value.

LUA_BID_ALREADY_ENABLED
The **lua_flag1** was set to LUA_BID_ENABLE to reenable an **RUI_BID** verb, but the previous **RUI_BID** verb was still in progress.

LUA_DUPLICATE_READ_FLOW
The flow flags on **lua_flag1** specified one or more session flows for which an **RUI_READ** verb was already outstanding. Only one **RUI_READ** at a time can be waiting on each session flow.

LUA_INVALID_FLOW
None of the **lua_flag1** flow flags was set. At least one of these flags must be set to indicate which flow or flows to read from.

LUA_NO_PREVIOUS_BID_ENABLED
The **lua_flag1** was set to LUA_BID_ENABLE, to reenable an **RUI_BID** verb, but there was no previous **RUI_BID** verb that could be enabled. (See “Interaction with Other Verbs” on page 219 for more information.)

LUA_RESERVED_FIELD_NOT_ZERO
A reserved field in the verb record, or a parameter that is not used by this verb, was set to a nonzero value.

LUA_VERB_LENGTH_INVALID
The value of the **lua_verb_length** parameter was less than the length of the verb record required for this verb.

State Check

The following return codes indicate that the verb was issued in a session state in which it was not valid:

lua_prim_rc LUA_STATE_CHECK

lua_sec_rc LUA_NO_RUI_SESSION
An **RUI_INIT** verb has not yet completed successfully for this session, or a session outage has occurred.

Negative Response Sent to Host

The following primary return code indicates one of the following two cases, which can be distinguished by the secondary return code:

Personal Communications detected an error in the data received from the host. Instead of passing the received message to the application on an **RUI_READ** verb, Personal Communications discards the message (and the rest of the chain if it is in a chain), and sends a negative response to the host. LUA

informs the application on a subsequent **RUI_READ** or **RUI_BID** verb that a negative response was sent.

The LUA application previously sent a negative response to a message in the middle of a chain. Personal Communications has purged subsequent messages in this chain, and is now reporting to the application that all messages from the chain have been received and purged.

lua_prim_rc LUA_NEGATIVE_RSP

lua_sec_rc A nonzero secondary return code contains the sense code sent to the host on the negative response. This indicates that Personal Communications detected an error in the host data, and sent a negative response to the host. See "SNA Layers" on page 166 for information on interpreting the sense code values that can be returned.

A zero secondary return code indicates that, following a previous **RUI_WRITE** of a negative response to a message in the middle of a chain, Personal Communications has now received and discarded all messages from this chain.

Unsuccessful Execution

The following return codes indicate that the verb record supplied was valid, but the verb did not complete successfully:

lua_prim_rc LUA_UNSUCCESSFUL

lua_sec_rc Possible values:

LUA_DATA_TRUNCATED
The **lua_data_length** parameter was smaller than the actual length of data received on the message. Only **lua_data_length** bytes of data were returned to the verb; the remaining data was discarded. Additional parameters are also returned if this secondary return code is obtained; see "Successful Execution or Truncated Data" on page 215.

LUA_NO_DATA
The **lua_flag1** was set to LUA_NOWAIT to indicate immediate return without waiting for data, and no data was currently available on the specified session flow or flows.

LUA_INVALID_PROCESS
The application instance that issued this verb was not the same as the one that issued the **RUI_INIT** verb for this session.

Other Conditions

The following primary and secondary return codes indicate that the verb did not complete successfully for other reasons.

lua_prim_rc LUA_SESSION_FAILURE
The session has been brought down.

lua_sec_rc Possible values:

LUA_LU_COMPONENT_DISCONNECTED
The LUA session has failed because of a problem with the communications link or with the host LU.

LUA_RUI_LOGIC_ERROR
This return code indicates one of the following things:

The host system has violated SNA protocols.
An internal error was detected within LUA.

Attempt to reproduce the problem with tracing active, and check that the host is sending correct data.

lua_prim_rc LUA_INVALID_VERB
Either the **lua_verb** parameter or the **lua_opcode** parameter was not valid. The verb did not execute.

lua_prim_rc LUA_UNEXPECTED_DOS_ERROR
An operating system error occurred, such as resource shortage.

lua_sec_rc This value is the operating system return code. Check your operating system documentation for the meaning of this return code.

Interaction with Other Verbs

The **RUI_INIT** verb must have completed successfully before this verb can be issued. While an existing **RUI_READ** is pending, you can issue another **RUI_READ** only if it specifies a different session flow or flows from pending **RUI_READs**; that is, you cannot have more than one **RUI_READ** outstanding for the same session flow.

The **lua_flag1** can only be set to LUA_BID_ENABLE if all of the following things are true:

An **RUI_BID** has already been issued successfully and has completed.
The storage allocated for the **RUI_BID** verb has not been freed or modified.
No other **RUI_BID** is pending.

Usage and Restrictions

If the data received is longer than the **lua_max_length** parameter, it will be truncated; only **lua_max_length** bytes of data will be returned. The primary and secondary return codes LUA_UNSUCCESSFUL and LUA_DATA_TRUNCATED will also be returned.

Once a message has been read using the **RUI_READ** verb, it is removed from the incoming message queue, and cannot be accessed again.

Note: The **RUI_BID** verb can be used as a nondestructive read; that is, the application can use it to check the type of data available, but the data remains on the incoming queue and need not be used immediately.

Pacing can be used on the primary-to-secondary half-session (this is specified in the host configuration) in order to protect the Personal Communications node from being flooded with messages. If the LUA application is slow to read messages, Personal Communications delays the sending of pacing responses to the host in order to slow it down.

RUI_TERM

The **RUI_TERM** verb ends both the LU-LU session and the LU-SSCP session for a given LUA LU.

Supplied Parameters

The application supplies the following parameters:

lua_verb LUA_VERB_RUI

lua_verb_length The length in bytes of the LUA verb record. Set this to size of (struct LUA_COMMON).

lua_opcode LUA_OPCODE_RUI_TERM

lua_correlator Optional. A 4-byte value, which you can use to correlate this verb with other processing within your application. LUA does not use or change this information.

lua_luname The name in ASCII of the local LU used by the session. This must match the LU name of an active LUA session (or the LU name specified on an outstanding **RUI_INIT** verb).

This parameter is required only if the **lua_sid** parameter is zero. If a session ID is supplied in **lua_sid**, LUA does not use this parameter.

This parameter must be 8 bytes long; pad on the right with spaces, 0x20, if the name is shorter than 8 characters.

lua_sid The session ID of the session. This must match a session ID returned on a previous **RUI_INIT** verb.

This parameter is optional; if you do not specify the session ID, you must specify the LU name for the session in the **lua_luname** parameter.

lua_post_handle This is a 4-byte handle that is used to post the completion of asynchronous verbs. It is a Windows 95 event handle.

Returned Parameters

The following parameter will always be returned:

lua_flag2 This is only set to LUA_ASYNC if the verb completed asynchronously.

Other returned parameters depend on whether the verb completed successfully; see the following sections.

Successful Execution

If the verb executes successfully, LUA also returns the following parameters:

lua_prim_rc LUA_OK

Parameter Check

The following return codes indicate that the verb did not complete successfully because a supplied parameter was in error:

lua_prim_rc LUA_PARAMETER_CHECK

lua_sec_rc Possible values:

LUA_RESERVED_FIELD_NOT_ZERO
A reserved field in the verb record, or a parameter that is not used by this verb, was set to a nonzero value.

LUA_VERB_LENGTH_INVALID
The value of the **lua_verb_length** parameter was less than the length of the verb record required for this verb.

State Check

The following return codes indicate that the verb was issued in a session state in which it was not valid:

lua_prim_rc LUA_STATE_CHECK

lua_sec_rc LUA_NO_RUI_SESSION
An **RUI_INIT** verb has not yet completed successfully for this session, or a session outage has occurred.

Unsuccessful Execution

The following return codes indicate that the verb record supplied was valid, but the verb did not complete successfully:

lua_prim_rc LUA_UNSUCCESSFUL

lua_sec_rc Possible values:

LUA_COMMAND_COUNT_ERROR
An **RUI_TERM** was already pending when the verb was issued.

LUA_INVALID_PROCESS
The application instance that issued this verb was not the same as the one that issued the **RUI_INIT** verb for this session.

Other Conditions

The following primary and secondary return codes indicate that the verb did not complete successfully for other reasons.

lua_prim_rc LUA_SESSION_FAILURE
The session has been brought down.

lua_sec_rc Possible values:

LUA_LU_COMPONENT_DISCONNECTED
The LUA session has failed because of a problem with the communications link or with the host LU.

LUA_RUI_LOGIC_ERROR
This return code indicates one of the following things:

The host system has violated SNA protocols.
An internal error was detected within LUA.

Attempt to reproduce the problem with tracing active, and check that the host is sending correct data.

lua_prim_rc LUA_INVALID_VERB
Either the **lua_verb** parameter or the **lua_opcode** parameter was not valid. The verb did not execute.

lua_prim_rc	LUA_UNEXPECTED_DOS_ERROR An operating system error occurred, such as resource shortage.
lua_sec_rc	This value is the operating system return code. Check your operating system documentation for the meaning of this return code.

Interaction with Other Verbs

This verb can be issued at any time after the **RUI_INIT** verb has been issued (whether or not it has completed). If any other LUA verb is pending when **RUI_TERM** is issued, no further processing on the pending verb will take place, and it will return with a primary return code of LUA_CANCELLED.

After this verb has completed, no other LUA verb can be issued for this session.

RUI_WRITE

The **RUI_WRITE** verb sends an SNA request or response unit from the LUA application to the host, over either the LU-LU session or the LU-SSCP session.

Supplied Parameters

The application supplies the following parameters:

lua_verb LUA_VERB_RUI

lua_verb_length The length in bytes of the LUA verb record. Set this to sizeof(struct LUA_COMMON).

lua_opcode LUA_OPCODE_RUI_WRITE

lua_correlator Optional. A 4-byte value, which you can use to correlate this verb with other processing within your application. LUA does not use or change this information.

lua_luname The name in ASCII of the local LU used by the session. This must match the LU name of an active LUA session.

This parameter is required only if the **lua_sid** parameter is zero. If a session ID is supplied in **lua_sid**, LUA does not use this parameter.

This parameter must be 8 bytes long; pad on the right with spaces, 0x20, if the name is shorter than 8 characters.

lua_sid The session ID of the session. This must match a session ID returned on a previous **RUI_INIT** verb.

This parameter is optional; if you do not specify the session ID, you must specify the LU name for the session in the **lua_luname** parameter.

lua_data_length The length of the supplied data (see **lua_data_ptr**). When sending data on the LU normal flow, the maximum length is as specified in the **BIND** received from the host; for all other flows the maximum length is 256 bytes.

When sending a positive response, this parameter is normally set to zero. LUA will complete the response based on the supplied sequence number (see **lua_th.snf**). In the case of a positive response to a **BIND** or **STSN**, an extended response is permitted, so a nonzero value can be used.

When sending a negative response, set this parameter to the length of the SNA sense code (4 bytes), which is supplied in the data buffer (see **lua_data_ptr**).

lua_data_ptr A pointer to the buffer containing the supplied data.

For a request, or a positive response that requires data, the buffer should contain the entire RU. The length of the RU must be specified in **data_length**.

For a negative response, the buffer should contain the SNA sense code.

lua_post_handle This is a 4-byte handle that is used to post the completion of asynchronous verbs. It is a Windows 95 event handle.

lua_th.snf Required only when sending a response. The sequence number of the request to which this is the response.

lua_rh When sending a request, most of the **lua_rh** flags must be set to correspond to the RH (request header) of the message to be sent. Do not set LUA_PI and LUA_QRI; these will be set by LUA.

When sending a response, only the following two **lua_rh** flags are set:

LUA_RRI
Is set to indicate a response.

LUA_RI
Is not set for a positive response, or set for a negative response.

lua_flag1 Set one of the following flags to indicate which message flow the data is to be sent on:

LUA_LU_EXP
LU expedited flow

LUA_SSCP_NORM
SSCP normal flow

LUA_LU_NORM
LU normal flow

One and only one of the flags must be set.

Note: Personal Communications does not permit applications to send data on the SSCP expedited flow (LUA_SSCP_EXP).

Returned Parameters

The following parameter will always be returned:

lua_flag2 This is only set to LUA_ASYNC if the verb completed asynchronously.

Other returned parameters depend on whether the verb completed successfully; see the following sections.

Successful Execution

If the verb executes successfully, LUA also returns the following parameters:

lua_prim_rc LUA_OK

lua_sid If the application specified the **lua_luname** parameter when issuing this verb, rather than specifying the session ID, LUA supplies the session ID.

lua_th The completed TH of the message written, including the fields filled in by LUA. You might need to save the value of **lua_th.snf** (the sequence number) for correlation with responses from the host.

lua_rh The completed RH of the message written, including the fields filled in by LUA.

lua_flag2 This will be set to one of the following values to indicate which message flow the data was received on:

LUA_SSCP_EXP
SSCP expedited flow

LUA_LU_EXP
LU expedited flow

LUA_SSCP_NORM
SSCP normal flow

LUA_LU_NORM
LU normal flow

Verb Canceled

The following return codes indicate that the verb did not complete successfully because it was canceled by another verb:

lua_prim_rc LUA_CANCELLED

lua_sec_rc LUA_TERMINATED
The verb was canceled because an **RUI_TERM** verb was issued for this session.

Parameter Check

The following return codes indicate that the verb did not complete successfully because a supplied parameter was in error:

lua_prim_rc LUA_PARAMETER_CHECK

lua_sec_rc Possible values:

LUA_BAD_DATA_PTR
The **lua_data_ptr** parameter contained an incorrect value.

LUA_DUPLICATE_WRITE_FLOW
An **RUI_WRITE** was already outstanding for the session flow specified on this verb (the session flow is specified by setting one of the **lua_flag1** flow flags). Only one **RUI_WRITE** at a time can be outstanding on each session flow.

LUA_INVALID_FLOW
lua_flag1 was set to LUA_SSCP_EXP, indicating that the message should be sent on the SSCP expedited flow. Personal Communications does not permit applications to send data on this flow.

LUA_MULTIPLE_WRITE_FLOWS
More than one of the **lua_flag1** flow flags was set. One and only one of these flags must be set to indicate which session flow the data is to be sent on.

LUA_REQUIRED_FIELD_MISSING
This return code indicates one of the following cases:

None of the **lua_flag1** flow flags was set. One and only one of these flags must be set.

The **RUI_WRITE** verb was used to send a response, and the response required more data than was supplied.

LUA_RESERVED_FIELD_NOT_ZERO
A reserved field in the verb record, or a parameter that is not used by this verb, was set to a nonzero value.

LUA_VERB_LENGTH_INVALID
The value of the **lua_verb_length** parameter was less than the length of the verb record required for this verb.

State Check

The following return codes indicate that the verb was issued in a session state in which it was not valid:

lua_prim_rc LUA_STATE_CHECK

lua_sec_rc Possible values:

LUA_MODE_INCONSISTENCY
The SNA message sent on the **RUI_WRITE** was not valid at this time. This is caused by trying to send data on the LU-LU session before the session is bound. Check the sequence of SNA messages sent.

LUA_NO_RUI_SESSION
An **RUI_INIT** verb has not yet completed successfully for this session, or a session outage has occurred.

Unsuccessful Execution

The following return codes indicate that the verb record supplied was valid, but the verb did not complete successfully:

lua_prim_rc LUA_UNSUCCESSFUL

lua_sec_rc Possible values:

LUA_FUNCTION_NOT_SUPPORTED
This return code indicates one of the following cases:

lua_rh was set to LUA_FI (Format Indicator), but the first byte of the supplied RU was not a recognized request code.

lua_rh was set to LUA_RH_NC (RU category specified the Network Control (NC) category); Personal Communications does not permit applications to send requests in this category.

LUA_INVALID_PROCESS
The application instance that issued this verb was not the same as the one that issued the **RUI_INIT** verb for this session.

LUA_INVALID_SESSION_PARAMETERS
The application used **RUI_WRITE** to send a positive response to a **BIND** message received from the host. However, the Personal Communications node cannot accept the **BIND** parameters as specified, and has sent a negative response to the host. See “SNA Layers” on

page 166 for more information on the **BIND** profiles accepted by Personal Communications.

LUA_RSP_CORRELATION_ERROR
When using **RUI_WRITE** to send a response, the **lua_th.snf** parameter (which indicates the sequence number of the received message being responded to) did not contain a valid value.

LUA_RU_LENGTH_ERROR
The **lua_data_length** parameter contained an incorrect value. When sending data on the LU normal flow, the maximum length is as specified in the **BIND** received from the host; for all other flows the maximum length is 256 bytes.

(any other value)
Any other secondary return code here is an SNA sense code indicating that the supplied SNA data was not valid or could not be sent. See “SNA Layers” on page 166 for information on interpreting the SNA sense codes that can be returned.

Other Conditions

The following primary and secondary return codes indicate that the verb did not complete successfully for other reasons:

lua_prim_rc LUA_SESSION_FAILURE
The session has been brought down.

lua_sec_rc Possible values:

LUA_LU_COMPONENT_DISCONNECTED
The LUA session has failed because of a problem with the communications link or with the host LU.

LUA_RUI_LOGIC_ERROR
This return code indicates one of the following things: The host system has violated SNA protocols. An internal error was detected within LUA.

Attempt to reproduce the problem with tracing active, and check that the host is sending correct data.

lua_prim_rc LUA_INVALID_VERB
Either the **lua_verb** parameter or the **lua_opcode** parameter was not valid. The verb did not execute.

lua_prim_rc LUA_UNEXPECTED_DOS_ERROR
An operating system error occurred, such as resource shortage.

lua_sec_rc This value is the operating system return code. Check your operating system documentation for the meaning of this return code.

Interaction with Other Verbs

The **RUI_INIT** verb must be issued successfully before this verb can be issued. While an existing **RUI_WRITE** is pending, you can issue a second **RUI_WRITE** only if it specifies a different session flow from the pending **RUI_WRITE**; that is, you cannot have more than one **RUI_WRITE** outstanding for the same session flow.

The **RUI_WRITE** verb can be issued on the SSCP normal flow at any time after a successful **RUI_INIT** verb. **RUI_WRITE** verbs on the LU expedited or LU normal flows are permitted only after a **BIND** has been received, and must abide by the protocols specified on the **BIND**.

Usage and Restrictions

The successful completion of **RUI_WRITE** indicates that the message was queued successfully to the data link; it does not necessarily indicate that the message was sent successfully, or that the host accepted it. Pacing can be used on the secondary-to-primary half-session (this is specified on the **BIND**) in order to prevent the LUA application from sending more data than the local or remote LU can handle. If this is the case, an **RUI_WRITE** on the LU normal flow can be delayed by LUA and can take some time to complete.

Note: Personal Communications does not permit applications to send data on the SSCP expedited flow (LUA_SSCP_EXP).

Appendix A. APPC Common Return Codes

This appendix describes the primary (and, if applicable, secondary) return codes that are common to several APPC verbs.

Verb-specific return codes are described in the documentation for the individual verbs.

AP_ALLOCATION_ERROR
Personal Communications has failed to allocate a conversation. The conversation state is set to RESET. This code can be returned through a verb issued after **ALLOCATE** or **MC_ALLOCATE**. The associated secondary return codes are as follows:

AP_ALLOCATION_FAILURE_NO_RETRY
The conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. To determine the error, the system administrator should examine the error log file. Do not attempt to retry the allocation until the error has been corrected.

AP_ALLOCATION_FAILURE_RETRY
The conversation could not be allocated because of a temporary condition, such as a link failure. The reason for the failure is logged in the system error log. Retry the allocation, preferably after a timeout to permit the condition to clear.

AP_SECURITY_NOT_VALID
The user ID or password specified in the allocation request was not accepted by the partner LU.

AP_TRANS_PGM_NOT_AVAIL_RETRY
The remote LU rejected the allocation request because it was unable to start the requested partner transaction program. The requested transaction program (TP) is not available because of a transient condition, such as a timeout. The reason for the error might be logged on the remote node. The condition might clear itself without operator intervention. The transaction program should retry the conversation, preferably after a timeout to permit the condition to clear.

AP_TRANS_PGM_NOT_AVAIL_NO_RTRY
The remote LU rejected the allocation request because it was unable to start the requested partner transaction program. The requested transaction program (TP) is not available because of a permanent or semi-permanent condition. The reason for the error might be logged on the remote node. The condition will not clear itself without operator intervention. The transaction program should not retry the conversation until the error condition has been cleared.

AP_TP_NAME_NOT_RECOGNIZED
The transaction program name specified in the allocation request is not recognized by the partner LU.

AP_PIP_NOT_ALLOWED
The requested transaction program cannot receive program initialization parameters (PIP). This indicates a mismatch between the local and partner transaction programs.

© Copyright IBM Corp. 1994, 1996

AP_PIP_NOT_SPECIFIED_CORRECTLY
The requested transaction program can receive program initialization parameters (PIP), but detected an error in the supplied PIP. This indicates a mismatch between the local and partner transaction programs.

AP_CONVERSATION_TYPE_MISMATCH
The requested transaction program cannot support conversations of the type (basic or mapped) specified in the allocation request. This indicates a mismatch between the local and partner transaction programs.

AP_SYNC_LEVEL_NOT_SUPPORTED
The requested transaction program cannot support conversations with the **sync_level** (AP_NONE, AP_CONFIRM_SYNC_LEVEL or AP_SYNCPT) specified in the allocation request. This indicates a mismatch between the local and partner transaction programs.

AP_CANCELLED
The verb returned because the conversation was canceled (the transaction program issued a **CANCEL_CONVERSATION** verb).

AP_CONV_FAILURE_NO_RETRY
The conversation was terminated because of a permanent condition, such as a session protocol error. The system administrator should examine the system error log to determine the cause of the error. Do not retry the conversation until the error has been corrected.

AP_CONV_FAILURE_RETRY
The conversation was terminated because of a temporary error. Restart the transaction program to see if the problem occurs again. If it does, the system administrator should examine the error log to determine the cause of the error.

AP_CONVERSATION_TYPE_MIXED
The transaction program has attempted to mix conversation verbs for different conversation types on the same conversation. For example, the transaction program issued an **MC_ALLOCATE** verb followed by a **CONFIRM** verb.

AP_DEALLOC_ABEND
The conversation has been deallocated for one of the following reasons.

The partner transaction program has issued the **MC_DEALLOCATE** verb with **dealloc_type** set to AP_ABEND.

The partner transaction program has ended abnormally, causing the partner LU to send an **MC_DEALLOCATE** request.

AP_DEALLOC_ABEND_PROG
The conversation has been deallocated for one of the following reasons.

The partner transaction program has issued the **DEALLOCATE** verb with **dealloc_type** set to AP_ABEND_PROG.

The partner transaction program has ended abnormally, causing the partner LU to send a **DEALLOCATE** request.

AP_DEALLOC_ABEND_SVC
The conversation has been deallocated because the partner transaction program issued the **DEALLOCATE** verb with **dealloc_type** set to AP_ABEND_SVC.

AP_DEALLOC_ABEND_TIMER
The conversation has been deallocated because the partner transaction program has issued the **DEALLOCATE** verb with **dealloc_type** set to AP_ABEND_TIMER.

AP_DEALLOC_NORMAL
This return code does not indicate an error. The partner transaction program issued the **DEALLOCATE** or **MC_DEALLOCATE** verb with **dealloc_type** set to one of the following values.

AP_FLUSH

AP_SYNC_LEVEL with the synchronization level of the conversation specified as AP_NONE

AP_PROG_ERROR_NO_TRUNC
The partner transaction program has issued one of the following verbs while the conversation was in SEND state.

SEND_ERROR with **err_type** set to AP_PROG
MC_SEND_ERROR

Data was not truncated.

AP_PROG_ERROR_PURGING
The partner transaction program issued one of the following verbs while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state.

SEND_ERROR with **err_type** set to AP_PROG.
MC_SEND_ERROR

Data sent, but not yet received, is purged.

AP_PROG_ERROR_TRUNC
In SEND state, after sending an incomplete logical record, the partner transaction program issued a **SEND_ERROR** verb with **err_type** set to AP_PROG. The local transaction program might have received the first part of the logical record through a **RECEIVE** verb.

AP_SVC_ERROR_NO_TRUNC
While in SEND state, the partner transaction program (or partner LU) issued a **SEND_ERROR** verb with **err_type** set to AP_SVC. Data was not truncated.

AP_SVC_ERROR_PURGING
The partner transaction program (or partner LU) issued a **SEND_ERROR** verb with **err_type** set to AP_SVC while in RECEIVE, PENDING_POST, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE state. Data sent to the partner transaction program might have been purged.

AP_SVC_ERROR_TRUNC
In SEND state, after sending an incomplete logical record, the partner transaction program (or partner LU) issued a **SEND_ERROR** verb. The local transaction program might have received the first part of the logical record.

AP_TP_BUSY
The local transaction program has issued a blocking verb to Personal Communications while Personal Communications was processing another verb for the same conversation.

AP_UNEXPECTED_SYSTEM_ERROR

Personal Communications has encountered an unexpected system error, and cannot complete the verb. Usually these errors arise from a shortage of system resources (for example, memory), and are usually transient. Check the system log for more details.

Appendix B. APPC Conversation State Transitions

The following table shows the conversation states in which each APPC verb can be issued, and the state change that occurs on completion of the verb. In some cases, the state change depends on the **primary_rc** parameter returned to the verb; where this applies, the applicable **primary_rc** values are listed in the Return codes column.

Where no return codes are shown, the state changes are the same for all return codes (except as described in Notes 2 and 3 following the table).

The possible conversation states are shown as column headings. Against each verb, the following information is given under each heading to indicate the results of issuing the verb in this state:

X if the verb cannot be issued in this state.

S, **SP**, **R**, **C**, **CS**, **CD**, or **P** to indicate the state of the conversation after the verb has completed: **R**eset, **S**end, **S**end **P**ending, **R**eceive, **C**onfirm, **C**onfirm **S**end, **C**onfirm **D**eallocate, or **P**ending Post.

/ if it is not applicable to consider the previous state. This applies to the **[MC_]ALLOCATE** and **RECEIVE_ALLOCATE** verbs; these verbs always start a new conversation as though they were in Reset state, with no effect on the conversation (if any) in which they were issued.

Blank if the return code shown cannot occur in this state.

© Copyright IBM Corp. 1994, 1996

Table 4. APPC Conversation State Transitions

Verb Return Codes	Reset (T)	Send (S)	Send Pending (SP)	Receive (R)	Confirm (C)	Confirm Send (CS)	Confirm Deall (CD)	Pend-Post (PS)
[MC_]ALLOCATE AP_OK (other)	S T	/	/	/	/	/	/	/
CANCEL_CONVERSATION	X	T	T	T	T	T	T	T
[MC_]CONFIRM AP_OK AP_ERROR	X	S R	S R	X	X	X	X	X
[MC_]CONFIRMED	X	X	X	X	R	S	T	X
[MC_]DEALLOCATE (Abend)	X	T	T	T	T	T	T	T
[MC_]DEALLOCATE (Other) AP_ERROR (other)	X	R T	R T	X	X	X	X	X
[MC_]FLUSH	X	S	S	X	X	X	X	X
[MC_]GET_ATTRIBUTES	X	S	SP	R	C	CS	CD	P
GET_STATE	X	S	SP	R	C	CS	CD	P
GET_TYPE	X	S	SP	R	C	CS	CD	P
[MC_]PREPARE_TO_ RECEIVE	X	R	R	X	X	X	X	X
RECEIVE_ALLOCATE AP_OK (other)	R T	/	/	/	/	/	/	/
[MC_]RECEIVE_AND_POST (Note 4)	X	P	P	P	X	X	X	X
[MC_]RECEIVE_AND_WAIT	X	Note 5	Note 5	Note 5	X	X	X	X
[MC_]RECEIVE_IMMEDIATE	X	X	X	Note 5	X	X	X	X
[MC_]REQUEST_TO_SEND	X	X	X	R	C	X	X	P
[MC_]SEND_DATA AP_OK AP_ERROR	X	S R	S	X	X	X	X	X
[MC_]SEND_ERROR AP_OK AP_ERROR	X	S R	S	S	S	S	S	S
[MC_]TEST_RTS	X	S	S	R	C	C	C	P

Notes:

1. In the Return codes column of the table, the abbreviation AP_ERROR is used for the following:

   AP_PROG_ERROR_TRUNC
   AP_PROG_ERROR_NO_TRUNC
   AP_PROG_ERROR_PURGING
   AP_SVC_ERROR_TRUNC
   AP_SVC_ERROR_NO_TRUNC
   AP_SVC_ERROR_PURGING.

2. The conversation will always enter Reset state if any of the following return codes are received:

   AP_ALLOCATION_ERROR
   AP_COMM_SUBSYSTEM_ABENDED
   AP_COMM_SUBSYSTEM_NOT_LOADED
   AP_CONV_FAILURE_RETRY
   AP_CONV_FAILURE_NO_RETRY
   AP_DEALLOC_ABEND
   AP_DEALLOC_ABEND_PROG
   AP_DEALLOC_ABEND_SVC
   AP_DEALLOC_ABEND_TIMER
   AP_DEALLOC_NORMAL

3. The following non-OK return codes do not cause any state change. The conversation always remains in the state in which the verb was issued:

   AP_CONVERSATION_TYPE_MIXED
   AP_PARAMETER_CHECK
   AP_STATE_CHECK
   AP_TP_BUSY
   AP_UNEXPECTED_SYSTEM_ERROR
   AP_UNSUCCESSFUL

4. After **[MC_]RECEIVE_AND_POST** has been issued and received the initial **primary_rc** of AP_OK, the conversation changes to Pending Post state. Once the supplied callback routine has been called to indicate that the verb has completed, the new conversation state depends on the **primary_rc** and **what_rcvd** parameters as in Note 5.

5. The state change after one of the **RECEIVE** verbs depends on both the **primary_rc** and **what_rcvd** parameters.

   If the **primary_rc** parameter is AP_PROG_ERROR*, AP_SVC_ERROR*, or ([MC_]RECEIVE_IMMEDIATE only) AP_UNSUCCESSFUL, the new state is RECEIVE.

   If the **primary_rc** parameter is AP_DEALLOC*, the new state is RESET.

   If the **primary_rc** parameter is AP_OK, the new state depends on the value of the **what_rcvd** parameter:

   Receive state
   AP_DATA, AP_DATA_COMPLETE, AP_DATA_INCOMPLETE

   Send state
   AP_SEND

Send Pending state
AP_DATA_SEND, AP_DATA_COMPLETE_SEND

Confirm state
AP_CONFIRM_WHAT_RECEIVED, AP_DATA_CONFIRM, AP_DATA_COMPLETE_CONFIRM

Confirm Send state
AP_CONFIRM_SEND, AP_DATA_CONFIRM_SEND, AP_DATA_COMPLETE_CONFIRM_SEND

Confirm Deallocate state
AP_CONFIRM_DEALLOCATE, AP_DATA_CONFIRM_DEALLOCATE, AP_DATA_COMPLETE_CONFIRM_DEALL

Glossary

This glossary includes terms and definitions from:

The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.

The ANSI/EIA Standard—440-A, *Fiber Optic Terminology*. Copies may be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue, N.W., Washington, DC 20006. Definitions are identified by the symbol (E) after the definition.

The *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

The *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

Internet Request for Comments: 1208, *Glossary of Networking Terms*.

Internet Request for Comments: 1392, *Internet Users' Glossary*.

The *Object-Oriented Interface Design: IBM Common User Access Guidelines*, Carmel, Indiana: Que, 1992.

The following cross-references are used in this glossary:

Contrast with: This refers to a term that has an opposed or substantively different meaning.

Synonym for: This indicates that the term has the same meaning as a preferred term, which is defined in its proper place in the glossary.

Synonymous with: This is a backward reference from a defined term to all other terms that have the same meaning.

See: This refers the reader to multiple-word terms that have the same last word.

See also: This refers the reader to terms that have a related, but not synonymous, meaning.

Deprecated term for: This indicates that the term should not be used. It refers to a preferred term, which is defined in its proper place in the glossary.

A

abend. (1) Abnormal end of task; the termination of a task before its completion because of an error condition that cannot be resolved by recovery facilities while the task is executing. (2) Synonym for *abnormal termination*.

abnormal termination. (1) The cessation of processing prior to planned termination. (T) (2) A system failure or operator action that causes a job to end unsuccessfully. (3) Synonymous with *abend*.

ACB name. (1) The label of an ACB macroinstruction used to name the ACB. (2) A name specified either on the VTAM APPL definition statement or on the VTAM application program's ACB macroinstruction. Contrast with *network name*.

accept. (1) In a VTAM application program, to establish a session with a logical unit (LU) in response to a CINIT request from a system services control point (SSCP). The session-initiation request may begin when a terminal user logs on, a VTAM application program issues a macroinstruction, or a VTAM operator issues a command. See also *acquire*. (2) An SMP process that moves distributed code and MVS-type programs to the distribution libraries.

ACCESS. In the Simple Network Management Protocol (SNMP), the clause in a Management Information Base (MIB) module that defines the minimum level of support that a managed node provides for an object.

access method. (1) A technique, implemented in software, that controls the flow of information through a network. (2) A technique for moving data between main storage and input/output devices.

ACF/VTAM. Advanced Communications Function for the Virtual Telecommunications Access Method. Synonym for *VTAM*.

acknowledgment. (1) The transmission, by a receiver, of acknowledge characters as an affirmative response to a sender. (T) (2) An indication that an item sent was received.

© Copyright IBM Corp. 1994, 1996

acquire. (1) In VTAM, to take over resources that were formerly controlled by an access method in another domain or to resume control of resources that were controlled by that domain but released. Contrast with *release*. See also *resource takeover*. (2) In a VTAM application program, to initiate and establish a session with another logical unit (LU). The acquire process begins when the application program issues a macroinstruction. See also *accept*.

action. (1) An operation on a managed object, the semantics of which are defined as part of the managed object class definition. (2) In the AIX operating system, a defined task that an application performs. An action modifies the properties of an object or manipulates the object in some way.

activate. To make a resource ready to perform its function. Contrast with *deactivate*.

active. (1) Operational. (2) Pertaining to a file, page, or program that is in main storage or memory, as opposed to a file, page, or program that must be retrieved from auxiliary storage; for example, an active page in an IBM personal computer. (3) Pertaining to a node or device that is connected or is available for connection to another node or device. (4) The state of a resource when it has been activated and is operational. (5) In the AIX operating system, pertaining to the window pane in which the text cursor is currently positioned. (6) In VTAM, pertaining to a major or minor node that has been activated by VTAM. Most resources are activated as part of VTAM start processing or as the result of a VARY ACT command. (7) Contrast with *inactive* and *inoperative*. (8) See also *pending active session*.

ACTLU. Activate logical unit. In SNA, a command used to start a session on a logical unit.

ACTPU. Activate physical unit. In SNA, a command used to start a session on a physical unit.

adapter. A part that electrically or physically connects a device to a computer or to another device.

adaptive pacing. Synonym for *adaptive session-level pacing*.

adaptive session-level pacing. A form of session-level pacing in which session components exchange pacing windows that may vary in size during the course of a session. This allows transmission within a network to adapt dynamically to variations in availability and demand of buffers on a session-by-session basis. Session-level pacing occurs within independent stages along the session path according to local congestion at the intermediate and endpoint nodes. Synonymous with *adaptive pacing* and *adaptive session pacing*. Contrast with *fixed session-level pacing*.

adaptive session pacing. Synonym for *adaptive session-level pacing*.

address. In data communication, the unique code assigned to each device, workstation, or user connected to a network.

Administrative Domain. A collection of hosts and routers, and the interconnecting networks, managed by a single administrative authority.

Advanced Peer-to-Peer Networking (APPN). An extension to SNA featuring (a) greater distributed network control that avoids critical hierarchical dependencies, thereby isolating the effects of single points of failure; (b) dynamic exchange of network topology information to foster ease of connection, reconfiguration, and adaptive route selection; (c) dynamic definition of network resources; and (d) automated resource registration and directory lookup. APPN extends the LU 6.2 peer orientation for end-user services to network control and supports multiple LU types, including LU 2, LU 3, and LU 6.2.

Advanced Peer-to-Peer Networking (APPN) end node. A node that provides a broad range of end-user services and supports sessions between its local control point (CP) and the CP in an adjacent network node. It uses these sessions to dynamically register its resources with the adjacent CP (its network node server), to send and receive directory search requests, and to obtain management services. An APPN end node can also attach to other end nodes.

Advanced Peer-to-Peer Networking (APPN) network. A collection of interconnected network nodes and their client end nodes.

Advanced Peer-to-Peer Networking (APPN) network node. A node that offers a broad range of end-user services and that can provide the following:

- Distributed directory services, including registration of its domain resources to a central directory server
- Topology database exchanges with other APPN network nodes, enabling network nodes throughout the network to select optimal routes for LU-LU sessions based on requested classes of service
- Session services for its local LUs and client end nodes
- Intermediate routing services within an APPN network

Advanced Peer-to-Peer Networking (APPN) node. An APPN network node or an APPN end node.

advanced program-to-program communication (APPC). (1) The general facility characterizing the LU 6.2 architecture and its various implementations in products. (2) Sometimes used to refer to the LU 6.2 architecture and its product implementations as a whole, or to an LU 6.2 product feature in particular, such as an APPC application programming interface.

AID. Attention identifier.

AIX NetView Service Point. See *NetView for AIX Service Point*.

all points addressable (APA). In computer graphics, pertaining to the ability to address and display or not display each picture element (pel) on a display surface.

allocate. (1) To assign a resource, such as a disk or diskette file, to perform a task. (2) An LU 6.2 application programming interface (API) verb used to assign a session to a conversation for the conversation's use. (3) Contrast with *deallocate*.

American National Standards Institute (ANSI). An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States. (A)

AND operation. Synonym for *conjunction*.

ANSI. American National Standards Institute.

AP. Alternate printer.

APA. All points addressable.

API. Application programming interface.

APPC. Advanced program-to-program communication.

application. A collection of software components used to perform specific types of user-oriented work on a computer.

application program. (1) A program written for or by a user that applies to the user's work, such as a program that does inventory control or payroll. (2) A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities.

application programming interface (API). (1) The set of programming language constructs or statements that can be coded in an application program to obtain the specific functions and services provided by an underlying operating system or service program. (2) In VTAM, the language structure used in control blocks so that application programs can reference them and be identified to VTAM.

apply. An SMP process that moves distributed code and MVS-type programs to the system libraries.

Apply. A push button that carries out the selected choices in a window without closing the window.

APPN. Advanced Peer-to-Peer Networking.

APPN end node. See *Advanced Peer-to-Peer Networking (APPN) end node*.

area. In Internet and DECnet routing protocols, a subset of a network or gateway grouped together by definition of the network administrator. Each area is self-contained; knowledge of an area's topology remains hidden from other areas.

argument. (1) An independent variable. (I) (A) (2) Any value of an independent variable; for example, a search key; a number identifying the location of an item in a table. (I) (A) (3) A parameter passed between a calling program and a called program.

ASCII (American National Standard Code for Information Interchange). The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (A)

ASYNC. Asynchronous.

asynchronous (ASYNC). (1) Pertaining to two or more processes that do not depend upon the occurrence of specific events such as common timing signals. (T) (2) Without regular time relationship; unexpected or unpredictable with respect to the execution of program instructions.

asynchronous operation. Simultaneous operations of software or hardware. In software, an operation, such as a request for session establishment or data transfer, in which the application program is allowed to continue execution while the operation is performed. The access method informs the application program after the operation is completed. Contrast with *synchronous operation*.

asynchronous request. In VTAM, a request for an asynchronous operation. Contrast with *synchronous request*.

attach. (1) In programming, to create a task that can be executed asynchronously with the execution of the mainline code. (2) To connect a device logically to a ring network.

attention identifier (AID). A character in a data stream indicating that the user has pressed a key, such as the Enter key, that requests an action by the system.

attribute. (1) A characteristic that identifies and describes a managed object. The characteristic can be determined, and possibly changed, through operations on the managed object. (2) Information within a managed object that is visible at the object boundary. An attribute has a type, which indicates the range of information given by the attribute, and a value, which is within that range. (3) Variable data that is logically a part of an object and that represents a property of the object. For example, a serial number is an attribute of an equipment object.

authorization. (1) In computer security, the right granted to a user to communicate with or make use of a computer system. (T) (2) An access right. (3) The process of granting a user either complete or restricted access to an object, resource, or function.

automatic activation. In VTAM, the activation of links and link stations in adjacent subarea nodes as a result of channel device name or RNAME specifications related to an activation command that names a subarea node. See also *direct activation*.

autotask. (1) An unattended NetView operator station task that does not require a terminal or a logged-on user. Autotasks can run independently of VTAM and are typically used for automated console operations. (2) Contrast with *logged-on operator*.

available. In VTAM, pertaining to a logical unit that is active, connected, enabled, and not at its session limit.

B

background process. (1) A process that does not require operator intervention but can be run by the computer while the workstation is used to do other work. (2) In the AIX operating system, a mode of program execution in which the shell does not wait for program completion before prompting the user for another command. (3) Contrast with *foreground process*.

backout. In IMS/VS, the process of removing all the database updates performed by an application program that has terminated abnormally.

base set. (1) The set of functions, including verbs, parameters, return codes, and what-received indications, that is supported by all products that implement a particular architecture. (2) Contrast with *option set*.

basic conversation. An LU 6.2 conversation type specified by the allocating transaction program. Transaction programs using basic conversation have available to them a wider variety of LU 6.2 functions, but they are responsible for more of their own error recovery and must manage details of the data stream used on the conversation. Contrast with *mapped conversation*.

basic information unit (BIU). In SNA, the unit of data and control information passed between half-sessions. It consists of a request/response header (RH) followed by a request/response unit (RU).

batch. (1) An accumulation of data to be processed. (2) A group of records or data processing jobs brought together for processing or transmission. (3) Pertaining to activity involving little or no user action. Contrast with *interactive*.

batch file. A file that contains a series of commands to be processed sequentially.

begin bracket. In SNA, the value (binary 1) of the begin-bracket indicator in the request header (RH) of the first request in the first chain of a bracket; the value denotes the start of a bracket. Contrast with *end bracket*. See also *bracket*.

behavior. (1) Ideally, a collection of assertions that describe the allowed states that a managed object can assume. An assertion can be a precondition, a postcondition, or an invariant. In practice, the behavior is often an informal description of the semantics of attributes, operations, and notifications. (2) The way in which managed objects, name bindings, attributes, notifications, and operations interact with the actual resources that they model and with each other.

Berkeley Internet Name Domain (BIND). The Berkeley implementation of the Domain Name System (DNS).

bid. In the contention form of invitation or selection, an attempt by the computer or by a station to gain control of a line in order to transmit data.

bidder. See *bidder session*.

bidder session. The half-session defined at session activation as having to request and receive permission from the other half-session to begin a bracket. Contrast with *first-speaker session*. Synonym for *contention-loser session*.

binary synchronous communication (BSC). (1) A form of telecommunication line control that uses a standard set of transmission control characters and control character sequences, for binary synchronous transmission of binary-coded data between stations. (2) Contrast with *Synchronous Data Link Control (SDLC)*.

bind. To relate an identifier to another object in a program; for example, to relate an identifier to a value, an address or another identifier, or to associate formal parameters and actual parameters. (T)

BIND. (1) In SNA, a request to activate a session between two logical units (LUs). See also *session activation request*. Contrast with *UNBIND*. (2) Berkeley Internet Name Domain.

bit. Either of the digits 0 or 1 when used in the binary numeration system. (T)

BIU. Basic information unit.

BIU segment. In SNA, the portion of a basic information unit (BIU) that is contained within a path information unit (PIU). It consists of either (a) a request/response header (RH) followed by all or a part of a request/response unit (RU) or (b) a part of an RU. Synonymous with *segment*.

block. A string of data elements recorded or transmitted as a unit. The elements may be characters, words, or physical records. (T)

blocking mode. (1) A way of requesting a service over an interface so that if the request cannot be completed immediately, the requesting process is suspended until the request is completed. (2) Contrast with *nonblocking mode*.

bracket. In SNA, one or more chains of request units and their responses that are exchanged between two session partners and that represent a transaction between them. A bracket must be completed before another bracket can be started. Examples of brackets are database inquiries/replies, update transactions, and remote job entry output sequences to workstations.

bracket protocol. In SNA, a data flow control protocol in which exchanges between two session partners are achieved through the use of brackets, with one partner designated at session activation as the first speaker and the other as the bidder. The bracket protocol involves bracket initiation and termination rules.

bridge. (1) A functional unit that interconnects two local area networks that use the same logical link control protocol but may use different medium access control protocols. (T) (2) A functional unit that interconnects multiple LANs (locally or remotely) that use the same logical link control protocol but that can use different medium access control protocols. A bridge forwards a frame to another bridge based on the medium access control (MAC) address. (3) In the connection of local loops, channels, or rings, the equipment and techniques used to match circuits and to facilitate accurate data transmission. (4) Contrast with *gateway* and *router*.

browse. (1) To look at records in a file. (2) In the NetView Graphic Monitor Facility, to open a view that cannot receive status changes from the NetView program. Contrast with *monitor*.

BSC. Binary synchronous communication.

buffer. (1) A routine or storage used to compensate for a difference in rate of flow of data, or time of occurrence of events, when transferring data from one device to another. (A) (2) To allocate and schedule the use of buffers. (A) (3) A portion of storage used to hold input or output data temporarily.

bus. (1) A facility for transferring data between several devices located between two end points, only one device being able to transmit at a given moment. (T) (2) A computer configuration in which processors are interconnected in series.

bus master. A device or subsystem that controls data transfers between itself and a subordinate.

byte. (1) A string that consists of a number of bits, treated as a unit, and representing a character. (T) (2) A binary character operated upon as a unit and usually shorter than a computer word. (A) (3) A group of 8 adjacent binary digits that represent one EBCDIC character.

C

C language. A language used to develop software applications in compact, efficient code that can be run on different types of computers with minimal change.

call. (1) The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (I) (A) (2) In data communication, the actions necessary to make a connection between two stations on a switched line. (3) In communications, a conversation between two users. (4) To transfer control to a procedure, program, routine, or subroutine. (5) To attempt to contact a user, regardless of whether the attempt is successful.

callback. In the AIX operating system, a procedure that is called if and when certain specified conditions are met.

calling. (1) The process of transmitting selection signals in order to establish a connection between data stations. (I) (A) (2) In X.25 communications, pertaining to the location or user that makes a call.

Cancel. A push button that removes a window without applying any changes made in that window.

case-sensitive. Pertaining to the ability to distinguish between uppercase and lowercase letters.

CD. Compact disc.

chain. (1) A group of logically linked user data records processed by LU 6.2. (2) A group of request units delimited by begin-chain and end-chain. Responses are always single-unit chains. See *RU chain*.

channel. (1) A path along which signals can be sent, for example, data channel, output channel. (A) (2) In data communication, a means of one-way transmission. (3) A functional unit, controlled by the processor, that handles the transfer of data between processor storage and local peripheral equipment.

channel-attached. (1) Pertaining to the attachment of devices directly by input/output channels to a host processor. (2) Pertaining to devices attached to a controlling unit by cables, rather than by telecommunication lines. (3) Contrast with *link-attached*. (4) Synonymous with *local*.

child. Pertaining to a secured resource, either a file or library, that uses the user list of a parent resource. A child resource can have only one parent resource. A child is a process, started by a parent process, that shares the resources of the parent process. Contrast with *parent*.

child process. In the AIX and OS/2 operating systems, a process, started by a parent process, that shares the resources of the parent process. See also *fork*.

CICS. Customer Information Control System.

circuit. (1) One or more conductors through which an electric current can flow. See *physical circuit* and *virtual circuit*. (2) A logic device.

circuit switching. (1) A process that, on demand, connects two or more data terminal equipment (DTEs) and permits the exclusive use of a data circuit between them until the connection is released. (I) (A) (2) Synonymous with *line switching*. (3) See also *message switching* and *packet switching*.

class. (1) In object-oriented design or programming, a group of objects that share a common definition and that therefore share common properties, operations, and behavior. Members of the group are called instances of the class. (2) In the AIX operating system, pertaining to the I/O characteristics of a device. System devices are classified as block or character devices.

cleanup. In SNA products, a network services request, sent by a system services control point (SSCP) to a logical unit (LU), that causes a particular LU-LU session with that LU to be ended immediately without requiring the participation of either the other LU or its SSCP.

clear to send (CTS). In data communication, a signal raised by data circuit-terminating equipment (DCE) when it is ready to accept data, usually in response to request to send (RTS) being raised.

client. (1) A functional unit that receives shared services from a server. (T) (2) A user. (3) In an AIX distributed file system environment, a system that is dependent on a server to provide it with programs or access to programs. (4) Synonymous with *requester*.

client/server. In communications, the model of interaction in distributed data processing in which a program at one site sends a request to a program at another site and awaits a response. The requesting program is called a client; the answering program is called a server.

Close. A choice that removes a window and all of the windows associated with it from the workplace. For example, if a user is performing a task in a window and a message appears, or the user asks for help, both the message and the help windows disappear when the user closes the original window.

CNOS. Change number of sessions.

code page. (1) An assignment of graphic characters and control function meanings to all code points; for example, assignment of characters and meanings to 256 code points for an 8-bit code, assignment of characters and meanings to 128 code points for a 7-bit code. (2) In the Print Management Facility, a font library member that associates code points and character identifiers. A code page also identifies invalid code points. (3) A particular assignment of hexadecimal identifiers to graphic characters. (4) In AFP support, a font file that associates code points and graphic character identifiers.

command. (1) A request from a terminal for the performance of an operation or the execution of a particular program. (2) In SNA, any field set in the transmission header (TH), request header (RH), and sometimes portions of a request unit (RU), that initiates an action or that begins a protocol; for example: (a) Bind Session (session-control request unit), a command that activates an LU-LU session, (b) the change-direction indicator in the RH of the last RU of a chain, (c) the virtual route reset window indicator in an FID4 transmission header. (3) See also *VTAM operator command*.

command area. In Basic CUA architecture, the area on a panel that contains the command entry field.

command line. (1) On a display screen, a display line usually at the bottom of the screen, in which only commands can be entered. (2) In CUA architecture, deprecated term for *command area*.

command prompt. A displayed character or string of characters that indicates that a user may enter a command to be processed.

Common Programming Interface for Communications (CPI-C). An evolving application programming interface (API), embracing functions to meet the growing demands from different application environments and to achieve openness as an industry standard for communications programming. CPI-C provides access to interprogram services such as (a) sending and receiving data, (b) synchronizing processing between programs, and (c) notifying a partner of errors in the communication.

Common Services API. In Communications Manager/2, application programming interface (API) verbs used to access services provided by Communications Manager/2 for user-written programs.

common transport semantics (CTS). The layer of the Networking Blueprint above the transport layer that makes the services of transport providers available to the transport user. See also *transport-layer protocol boundary (TLPB)*.

Communications Manager/2. An IBM licensed program that supports the development and use of OS/2 applications involving two or more connected systems or workstations. Communications Manager/2 provides multiple concurrent connectivities using different protocols for IBM 3270 and 5250 emulation sessions, printer sessions, and file transfers. It supports a range of application programming interfaces (APIs), which may be called concurrently and are designed for a variety of applications. Communications Manager/2 includes the necessary interfaces for network management.

compact disc (CD). (1) A disc, usually 4.75 inches in diameter, from which data is read optically by means of a laser. (2) A disc with information stored in the form of pits along a spiral track. The information is decoded by a compact-disc player and interpreted as digital audio data, which most computers can process.

compile. (1) To translate all or part of a program expressed in a high-level language into a computer program expressed in an intermediate language, an assembly language, or a machine language. (T) (2) To prepare a machine language program from a computer program written in another programming language by making use of the overall logic structure of the program, or generating more than one computer instruction for each symbolic statement, or both, as well as performing the function of an assembler. (A) (3) To translate a source program into an executable program (an object program). (4) To translate a program written in a high-level programming language into a machine language program.

compiler. (1) A program that translates a source program into an executable program (an object program). (2) A program that decodes instructions written as pseudo codes and produces a machine language program to be executed at a later time.

component. (1) Hardware or software that is part of a functional unit. (2) A part of a structured type or value, such as an array element or a record field.

computer. A functional unit that can perform substantial computations, including numerous arithmetic operations and logic operations without human intervention during a run. In information processing, the term computer usually describes a digital computer. A computer may consist of a stand-alone unit or may consist of several interconnected units. (T)

configuration. (1) The manner in which the hardware and software of an information processing system are organized and interconnected. (T) (2) The devices and programs that make up a system, subsystem, or network. (3) In CCP, the arrangement of controllers, lines, and terminals attached to an IBM 3710 Network Controller. Also, the collective set of item definitions that describe such a configuration.

configuration file. A file that specifies the characteristics of a system device or network.

configuration management. The control of information necessary to identify both physical and logical information system resources and their relationship to one another.

configuration services. One of the types of network services in a control point (SSCP, NNCP, or ENCP). Configuration services activates, deactivates, and records the status of physical units, links, and link stations.

configure. To describe to a system the devices, optional features, and programs installed on the system.

congestion. See *network congestion*.

conjunction. The Boolean operation whose result has the Boolean value 1 if and only if each operand has the Boolean value 1. (I) (A) Synonymous with *AND operation*.

connected. In VTAM, the state of a physical unit (PU) or a logical unit (LU) that has an active physical path to the host processor containing the system services

control point (SSCP) that controls the respective PU or LU.

connection. (1) In data communication, an association established between functional units for conveying information. (I) (A) (2) In Open Systems Interconnection architecture, an association established by a given layer between two or more entities of the next higher layer for the purpose of data transfer. (T) (3) In SNA, the network path that links together two logical units (LUs) in different nodes to enable them to establish communications. (4) In X.25 communication, a virtual circuit between two data terminal equipments (DTEs). A switched virtual circuit (SVC) connection lasts for the duration of a call; a permanent virtual circuit (PVC) is a permanent connection between the DTEs. (5) In TCP/IP, the path between two protocol applications that provides reliable data stream delivery service. In the Internet, a connection extends from a TCP application on one system to a TCP application on another system. (6) In system communications, a line over which data can be passed between two systems or between a system and a device. (7) Synonym for *physical connection*.

connection point manager. In SNA, a component of the transmission control layer that (a) performs session-level pacing of normal-flow requests, (b) checks sequence numbers of received request units, (c) verifies that request units do not exceed the maximum permissible size, (d) routes incoming request units to their destinations in the half-session, and (e) enciphers and deciphers FMD request units when cryptography is selected. The connection point manager coordinates the normal and expedited flows for one half-session.

Note: The sending connection point manager in a half-session builds the request/response header (RH) for outgoing request/response units (RUs), and the receiving connection point manager interprets the request/response headers that precede incoming request/response units.

connectivity. (1) The capability of a system or device to be attached to other systems or devices without modification. (T) (2) The capability to attach a variety of functional units without modifying them.

constraints. In NetDA/2, the set of essential requirements specified with the node, connection, or application definitions. A change in a constraint value changes the input to the network design. Contrast with *parameters*.

contention. In a session, a situation in which both NAUs attempt to initiate the same action at the same time, such as when both attempt to send data in a half-duplex protocol (half-duplex contention), or both attempt to start a bracket (bracket contention). At session initiation, one NAU is defined to be the contention winner; its action will take precedence when contention occurs. The contention loser must get explicit or implicit permission from the contention winner to begin its action.

contention-loser session. (1) To an NAU, a session for which it was defined during session initiation to be the contention loser. (2) Synonymous with *bidder session*.

contention-winner session. (1) To an NAU, a session for which it was defined during session initiation to be the contention winner. (2) Synonymous with *first-speaker session*.

control block. (1) A storage area used by a computer program to hold control information. (I) (2) In the IBM Token-Ring Network, a specifically formatted block of information provided from the application program to the Adapter Support Interface to request an operation.

control character. A character whose occurrence in a particular context specifies a control function. (T)

control point (CP). (1) A component of an APPN or LEN node that manages the resources of that node. In an APPN node, the CP is capable of engaging in CP-CP sessions with other APPN nodes. In an APPN network node, the CP also provides services to adjacent end nodes in the APPN network. (2) A component of a node that manages resources of that node and optionally provides services to other nodes in the network. Examples are a system services control point (SSCP) in a type 5 subarea node, a network node control point (NNCP) in an APPN network node, and an end node control point (ENCP) in an APPN or LEN end node. An SSCP and an NNCP can provide services to other nodes.

Control Program (CP). In VM/ESA, a component that manages the resources of a single computer so multiple computing systems appear to exist. Each of these apparent systems, or virtual machines, is the functional equivalent of an IBM System/370, 370-XA, or ESA computer.

conversation. A logical connection between two transaction programs using an LU 6.2 session. Conversations are delimited by brackets to gain exclusive use of a session.

conversation group ID. An identifier of a particular session between two specific LUs or CPs. Specified in a conversation allocation command, the conversation group ID allows the requested conversation to be allocated to the identified session. This then permits a pair of transaction programs (one in each LU) to serially share use of the designated session with a group of related pairs of transaction programs that use the same conversation group ID.

coordinated universal time (UTC). The time scale, based on the Système International (SI) second, as defined and recommended by the Comité Consultatif International de la Radio (CCIR) and maintained (using an atomic clock) by the Bureau International des Poids et Mesures (BIPM).

Note: The Système International is based on three fundamental units of measure—the meter, the kilogram, and the second—and is sometimes called the "MKS system" because of these units.

For most practical purposes, coordinated universal time (UTC) is equivalent to the mean solar time at the prime meridian (0 degrees longitude) of Greenwich, England, known as *Greenwich mean time (GMT)*. Synonymous with *Z time* and *Zulu time*.

Copy. A choice that places a copy of a selected object onto the clipboard.

correlator. Information that identifies a relation among things. An example is a variable field of a response that identifies the corresponding request.

CP. (1) Control point. (2) In VM, Control Program.

CRV. Cryptography verification.

cryptography verification (CRV) request. A request unit sent by the primary logical unit (PLU) to the secondary logical unit (SLU) as part of cryptographic session establishment, to allow the SLU to verify that the PLU is using the correct session cryptography key and initialization vector (IV).

CS. Current state.

CTS. (1) Common transport semantics. (2) Clear to send.

Customer Information Control System (CICS). An IBM licensed program that enables transactions entered at remote terminals to be processed concurrently by user-written application programs. It includes facilities for building, using, and maintaining databases.

D

DACTLU. Deactivate logical unit.

DACTPU. Deactivate physical unit.

DAF. Destination address field.

DAF'. Destination address field prime.

DASD. Direct access storage device.

data. (1) A re-interpretable representation of information in a formalized manner suitable for communication, interpretation, or processing. Operations can be performed upon data by humans or by automatic means. (T) (2) Any representations such as characters or analog quantities to which meaning is or might be assigned. (A) (3) A representation of facts or instructions in a form suitable for communication, interpretation, or processing by human or automatic means. Data include constants, variables, arrays, and character strings.

Note: Programmers make a distinction between instructions and the data they operate on; however, in the usual sense of the word, data includes programs and program instructions.

data circuit. (1) A pair of associated transmit and receive channels that provide a means of two-way data communication. (I) (2) In SNA, synonym for *link connection*. (3) See also *physical circuit* and *virtual circuit*.

Notes:

1. Between data switching exchanges, the data circuit may include data circuit-terminating equipment (DCE), depending on the type of interface used at the data switching exchange.
2. Between a data station and a data switching exchange or data concentrator, the data circuit includes the data circuit-terminating equipment at the data station end, and may include equipment similar to a DCE at the data switching exchange or data concentrator location.

data flow control (DFC). In SNA, a request/response unit (RU) category used for requests and responses exchanged between the data flow control layer in one half-session and the data flow control layer in the session partner.

data flow control (DFC) layer. In SNA, the layer within a half-session that controls whether the half-session can send, receive, or concurrently send and receive, request units (RUs); groups related RUs into RU chains; delimits transactions via the bracket protocol; controls the interlocking of requests and responses in accordance with control modes specified at session activation; generates sequence numbers; and correlates requests and responses.

data flow control (DFC) protocol. In SNA, the sequencing rules for requests and responses by which network addressable units (NAUs) in a session coordinate and control data transfer and other operations; for example, bracket protocol.

data link. In SNA, synonym for *link*.

data link control (DLC). A set of rules used by nodes on a data link (such as an SDLC link or a token ring) to accomplish an orderly exchange of information.

data link control (DLC) layer. In SNA, the layer that consists of the link stations that schedule data transfer over a link between two nodes and perform error control for the link. Examples of data link control are SDLC for serial-by-bit link connection and data link control for the System/370 channel.

Note: The DLC layer is usually independent of the physical transport mechanism and ensures the integrity of data that reaches the higher layers.

data set. (1) Synonym for *file*. (2) Deprecated term for *modem*.

data stream. (1) All information (data and control commands) sent over a data link usually in a single read or write operation. (2) A continuous stream of data elements being transmitted, or intended for transmission, in character or binary-digit form, using a defined format.

data types. In the NetView program, a description of the organization of panels. Data types are alerts, events, and statistics. Data types are combined with resource types and display types to describe the NetView program's display organization. See also *display types* and *resource types*.

DBCS. Double-byte character set.

deactivate. To take a resource of a node out of service, rendering it inoperable, or to place it in a state in which it cannot perform the functions for which it was designed. Contrast with *activate*.

deallocate. An LU 6.2 application programming interface (API) verb that terminates a conversation, thereby freeing the session for a future conversation. Contrast with *allocate*.

decryption. In computer security, transforming encoded text or ciphertext into plaintext.

default. Pertaining to an attribute, condition, value, or option that is assumed when none is explicitly specified. (I)

definite response (DR). In SNA, a protocol requested in the form-of-response-requested field of the request header that directs the receiver of the request to return a response unconditionally, whether positive or negative, to that request chain. Contrast with *exception response* and *no response*.

definition statement. (1) In VTAM, the statement that describes an element of the network. (2) In NCP, a type of instruction that defines a resource to the NCP. (3) See also *macroinstruction*. (4) See Figure 6, Figure 7, and Figure 8.

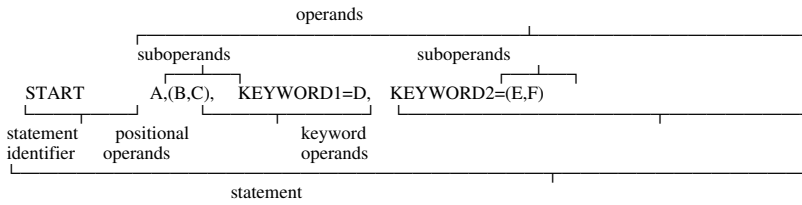

Figure 6. Example of a Language Statement

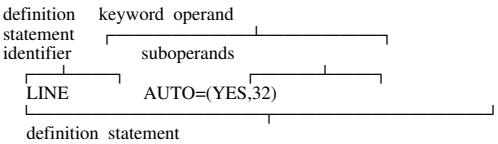

Figure 7. Example of an NCP Definition Statement

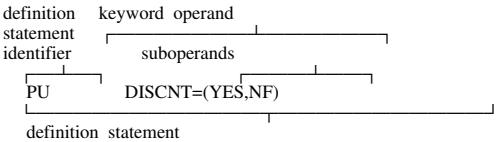

Figure 8. Example of a VTAM Definition Statement

Delete. A choice that removes a selected object. The space it occupied is usually filled by the remaining object or objects in the window.

dependent LU. See *SSCP-dependent LU*.

dependent LU requester (DLUR). An APPN end node or an APPN network node that owns dependent LUs, but requests that a dependent LU server provide the SSCP services for those dependent LUs.

destination. (1) Any point or location, such as a node, station, or a particular terminal, to which information is to be sent. (2) In ACF/TCAM, the place to which a message being handled by a message handler is to be sent. (3) An external logical unit (LU) or application program to which messages or other data are directed.

destination address. A code that identifies the location to which information is to be sent.

destination address field (DAF). In SNA, a field in a FID0 or FID1 transmission header that contains the network address of the destination. Contrast with *origin address field (OAF)*.

device. (1) A mechanical, electrical, or electronic contrivance with a specific purpose. (2) In the AIX operating system, a valuator, button, or the keyboard. Buttons have values of 0 or 1 (up or down); valuators return values in a range, and the keyboard returns ASCII values.

DFC. Data flow control.

dialog. (1) The interaction between a user and a computer. (2) In an interactive system, a series of related inquiries and responses similar to a conversation between two people. (3) In the AIXwindows Toolkit, a two-way text interface between an application and its user. The interface takes the form of a collection of widgets and gadgets, including a DialogShell widget, a BulletinBoard widget (or a subclass of a BulletinBoard widget or some other container widget), plus various children, including Label, PushButton, and Text widgets.

direct access storage. A storage device that provides direct access to data. (I) (A)

direct access storage device (DASD). A device in which access time is effectively independent of the location of the data.

direct activation. In VTAM, the activation of a resource as a result of an activation command specifically naming the resource. See *automatic activation*. Contrast with *indirect activation*.

directory. (1) A table of identifiers and references to the corresponding items of data. (I) (A) (2) A database in an APPN node that lists names of resources (in particular, logical units) and records the CP name of the node where each resource is located. See *distributed directory database* and *local directory database*. (3) A type of file containing the names and controlling information for other files or other directories. (4) A listing of the files stored on a disk or diskette. (5) In VM, a Control Program (CP) disk file that defines each virtual machine's typical configuration: the user ID, password, regular and maximum allowable virtual storage, CP command privilege class or classes allowed, dispatching priority, logical editing symbols to be used, account number, and CP options desired.

disable. To make nonfunctional.

disabled. (1) Pertaining to a state of a processing unit that prevents the occurrence of certain types of interruptions. (2) Pertaining to the state in which a transmission control unit or audio response unit cannot accept incoming calls on a line. (3) In VTAM, pertaining to a logical unit (LU) that has indicated to its system services control point (SSCP) that it is temporarily not ready to establish LU-LU sessions. An initiate request for a session with a disabled logical unit (LU) can specify that the session be queued by the SSCP until the LU becomes enabled. The LU can separately indicate whether this applies to its ability to act as a primary logical unit (PLU) or a secondary logical unit (SLU). See also *enabled* and *inhibited*.

display. (1) A visual presentation of data. (I) (A) (2) To present data visually. (I) (A) (3) Deprecated term for *panel*.

display levels. Synonym for *display types*.

display types. In the NetView program, a concept to describe the organization of panels. Display types are defined as total, most recent, user action, and detail. Display types are combined with resource types and data types to describe NetView's panel organization. See also *data types* and *resource types*. Synonymous with *display levels*.

distributed directory database. The complete listing of all the resources in the network as maintained in the individual directories scattered throughout an APPN network. Each node has a piece of the complete directory, but it is not necessary for any one node to have the entire list. Entries are created, modified, and deleted through system definition, operator action, automatic registration, and ongoing network search procedures. Synonymous with *distributed network directory* and *network directory database*.

distributed network directory. Synonym for *distributed directory database*.

distributed processing. Processing that takes place across two or more linked systems.

DLC. Data link control.

DLL. Dynamic link library.

domain. (1) That part of a computer network in which the data processing resources are under common control. (T) (2) In SNA, see *end node domain*, *network node domain*, and *system services control point (SSCP) domain*. (3) In Open Systems Interconnection (OSI), a part of a distributed system or a set of managed objects to which a common policy applies. (4) In a database, all the possible values of an attribute or a data element. (5) See *Administrative Domain* and *domain name*.

domain name. In the Internet suite of protocols, a name of a host system. A domain name consists of a sequence of subnames separated by a delimiter character. For example, if the fully qualified domain name (FQDN) of a host system is ralvm7.vnet.ibm.com, each of the following is a domain name:

ralvm7.vnet.ibm.com
vnet.ibm.com
ibm.com

domain operator. In a multiple-domain network, the person or program that controls operation of resources controlled by one system services control point (SSCP). See also *network operator*.

DOS. Disk Operating System. See *IBM Disk Operating System*.

double-byte character set (DBCS). A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, the typing, display, and printing of DBCS characters requires hardware and programs that support DBCS. Contrast with *single-byte character set (SBCS)*.

drain. To honor pending allocation requests before deactivating sessions with a partner logical unit. This applies to LU 6.2 only.

drive. A peripheral device, especially one that has addressed storage media.

duplex. (1) Pertaining to communication in which data can be sent and received at the same time. Synonymous with *full-duplex*. Contrast with *half-duplex*. (2) Pertaining to printing on both sides of a sheet of paper such that the bottom of the printed image on one side of the paper is juxtaposed against the bottom of the printed image on the other side of the paper. Contrast with *simplex* and *tumble*.

dynamic. (1) In programming languages, pertaining to properties that can only be established during the execution of a program; for example, the length of a variable-length data object is dynamic. (I) (2) Pertaining to an operation that occurs at the time it is needed rather than at a predetermined or fixed time. (3) Contrast with *static*.

dynamic link library (DLL). A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a dynamic link library can be shared by several applications simultaneously.

dynamic linking. In the OS/2 operating system, the delayed connection of a program to a routine until load time or run time.

E

EBCDIC. Extended binary-coded decimal interchange code. A coded character set of 256 8-bit characters.

EC. Engineering change.

Edit. A choice on a menu bar that provides access to other choices that enable a user to modify data.

element. (1) A field in the network address. (2) In SNA, the particular resource within a subarea that is identified by an element address. See also *subarea*.

enable. To make functional.

enabled. (1) Pertaining to a state of the processing unit that allows the occurrence of certain types of interruptions. (2) Pertaining to the state in which a transmission control unit or an audio response unit can accept incoming calls on a line. (3) In VTAM, pertaining to a logical unit (LU) that has indicated to its system services control point (SSCP) that it is ready to establish LU-LU sessions. The LU can separately indicate whether this prevents it from acting as a primary logical unit (PLU) or a secondary logical unit (SLU). See also *disabled* and *inhibited*.

enciphered data (ED). Data whose meaning is concealed from unauthorized users or observers.

encryption. In computer security, the process of transforming data into an unintelligible form in such a way that the original data either cannot be obtained or can be obtained only by using a decryption process.

end bracket. In SNA, the value (binary 1) of the end bracket indicator in the request header (RH) of the first request of the last chain of a bracket; the value denotes the end of the bracket. Contrast with *begin bracket*. See also *bracket*.

end node (EN). (1) See *Advanced Peer-to-Peer Networking (APPN) end node* and *low-entry networking (LEN) end node*. (2) In communications, a node that is frequently attached to a single data link and cannot perform intermediate routing functions.

end node domain. An end node control point, its attached links, and its local LUs.

entry point (EP). (1) The address or label of the first instruction executed on entering a computer program, routine, or subroutine. A computer program, routine, or subroutine may have a number of different entry points, each perhaps corresponding to a different function or purpose. (I) (A) (2) In SNA, a type 2.0, type 2.1, type 4, or type 5 node that provides distributed network management support. It sends network management data about itself and the resources it controls to a focal point for centralized processing, and it receives and executes focal-point initiated commands to manage and control its resources.

environment variable. A variable that specifies (a) how an operating system or another program will run or (b) the devices that the operating system will recognize.

ER. (1) Explicit route. (2) Exception response.

error. A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. (I) (A)

error log. (1) A data set or file in a product or system where error information is stored for later access. (2) A

form in a maintenance library that is used to record error information about a product or system. (3) A record of machine checks, device errors, and volume statistical data.

Ethernet. A 10-Mbps baseband local area network that allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by using collision detection and delayed retransmission. Ethernet uses carrier sense multiple access with collision detection (CSMA/CD).

event. (1) An occurrence of significance to a task; for example, an SNMP trap, the opening of a window or a submap, or the completion of an asynchronous operation. (2) In the NetView and NETCENTER programs, a record indicating irregularities of operation in physical elements of a network. (3) See also *event report*.

event report. The unsolicited report that an event has occurred. When a managed object emits a notification, the agent uses one or more event forwarding discriminators (EFDs) to find the destinations to which the report is sent.

exception. An abnormal condition such as an I/O error encountered in processing a data set or a file.

exception request (EXR). In SNA, a request that replaces another message unit in which an error has been detected and that carries sense data identifying the error.

exception response (ER). In SNA, a protocol requested in the form-of-response-requested field of a request header that directs the receiver to return a response only if the request is unacceptable as received or cannot be processed; that is, a negative response, but not a positive response, can be returned. Contrast with *definite response* and *no response*.

execute. To perform the actions specified by a program or a portion of a program. (T)

exit. (1) To execute an instruction within a portion of a computer program in order to terminate the execution of that portion. Such portions of computer programs include loops, subroutines, modules, and so on. (T) (2) See *installation exit* and *user exit*.

expedited flow. In SNA, a data flow designated in the transmission header (TH) that is used to carry network control, session control, and various data flow control request/response units (RUs); the expedited flow is separate from the normal flow (which carries primarily end-user data) and can be used for commands that affect the normal flow. Contrast with *normal flow*.

Note: The normal and expedited flows move in both the primary-to-secondary and secondary-to-primary directions. Requests and responses on a given flow, whether normal or expedited, usually are processed sequentially within the path, but the expedited flow traffic may be moved ahead of the normal-flow traffic within the path at queuing points in the half-sessions and for half-session support in boundary functions.

explicit route (ER). In SNA, a series of one or more transmission groups that connect two subarea nodes. An explicit route is identified by an origin subarea address, a destination subarea address, an explicit route number, and a reverse explicit route number. Contrast with *virtual route (VR)*.

EXR. Exception request.

EXT. External trace file.

F

fault. An accidental condition that causes a functional unit to fail to perform its required function. (I) (A)

FDX. Full-duplex.

feature. A part of an IBM product that may be ordered separately by the customer.

FID. Format identification.

field. (1) An identifiable area in a window. Examples of fields are: an entry field, into which a user can type or place text, and a field of radio button choices, from which a user can select one choice. (2) In NetView for AIX, the building block of which objects are composed. A field is characterized by a field name, a data type (integer, Boolean, character string, or enumerated value), and a set of flags that describe how the field is treated by NetView for AIX. A field can contain data only when it is associated with an object.

file. A named set of records stored or processed as a unit. (T) Synonymous with *data set*.

file specification. In Communications Manager/2, the full identifier for a file, which includes its drive, path, file name, and file extension. The format is dependent on the storage medium of a file; for example, C:\PATH\FILENAME.EXT.

file transfer. The transfer of one or more files from one system to another over a data link.

first-in-chain (FIC). A request unit (RU) whose request header (RH) begin chain indicator is on and

whose RH end chain indicator is off. See also *RU chain*.

first speaker. See *first-speaker session*.

first-speaker session. The half-session defined at session activation as (a) able to begin a bracket without requesting permission from the other half-session to do so and (b) winning contention if both half-sessions attempt to begin a bracket simultaneously. Synonym for *contention-winner session*. Contrast with *bidder session*.

fixed pacing. Synonym for *fixed session-level pacing*.

fixed session-level pacing. A form of session-level pacing in which the data transfer rate is controlled using fixed pacing-window sizes, which are initialized at session-activation time. Synonymous with *fixed pacing*. Contrast with *adaptive session-level pacing*.

flag. (1) To mark an information item for selection for further processing. (T) (2) A character that signals the occurrence of some condition, such as the end of a word. (A)

flow. In NetDA/2, the amount of traffic that can pass through a node, connection, or route in both directions during a given period of time.

flow control. (1) In data communication, control of the data transfer rate. (I) (2) In SNA, the process of managing the rate at which data traffic passes between components of the network. The purpose of flow control is to optimize the rate of flow of message units with minimum congestion in the network; that is, to neither overflow the buffers at the receiver or at intermediate routing nodes, nor leave the receiver waiting for more message units. (3) See also *pacing*.

flushing. In logical unit (LU) 6.2, the process of sending through the network all remaining buffered data generated by a transaction program. The transaction program issues the flush verb to begin the process. It also occurs if the network operator issues the command.

FMD. Function management data.

foreground process. (1) In the AIX operating system, a process that must run to completion before another command is issued to the shell. The foreground process is in the foreground process group, which is the group that receives the signals generated by a terminal. (2) Contrast with *background process*.

foreign host. Synonym for *remote host*.

fork. In the AIX operating system, to create and start a child process.

format identification (FID) field. In SNA, a field in each transmission header (TH) that indicates the format of the TH; that is, the presence or absence of certain fields. TH formats differ in accordance with the types of nodes between which they pass. Following are the six FID types:

FID0, used for traffic involving non-SNA devices between adjacent subarea nodes when either or both nodes do not support explicit route and virtual route protocols

FID1, used for traffic involving SNA devices between adjacent subarea nodes when either or both nodes do not support explicit route and virtual route protocols

FID2, used for traffic between a subarea node and an adjacent type 2 peripheral node

FID3, used for traffic between a subarea node and an adjacent type 1 peripheral node

FID4, used for traffic between adjacent subarea nodes when both nodes support explicit route and virtual route protocols

FIDF, used for certain commands (for example, for transmission group control) sent between adjacent subarea nodes when both nodes support explicit route and virtual route protocols.

FQDN. Fully qualified domain name.

frame. (1) In Open Systems Interconnection architecture, a data structure pertaining to a particular area of knowledge and consisting of slots that can accept the values of specific attributes and from which inferences can be drawn by appropriate procedural attachments. (T) (2) The unit of transmission in some local area networks, including the IBM Token-Ring Network. It includes delimiters, control characters, information, and checking characters. (3) In SDLC, the vehicle for every command, every response, and all information that is transmitted using SDLC procedures.

frequency. The rate of signal oscillation, expressed in hertz.

full-duplex (FDX). Synonym for *duplex*.

fully qualified domain name (FQDN). In the Internet suite of protocols, the name of a host system that includes all of the subnames of the domain name. An example of a fully qualified domain name is ralvm7.vnet.ibm.com. See also *host name*.

fully qualified name. (1) In SNA, synonym for *network-qualified name*. (2) In the Internet suite of protocols, see *fully qualified domain name (FQDN)*.

function call. An expression that moves the path of execution from the current function to a specified function and evaluates to the return value provided by

the called function. A function call contains the name of the function to which control moves and a parenthesized list of values.

function management data (FMD). An RU category used for end-user data exchanged between logical units (LUs) and for requests and responses exchanged between network services components of LUs, PUs, and control points.

G

gateway. (1) A functional unit that interconnects two computer networks with different network architectures. A gateway connects networks or systems of different architectures. A bridge interconnects networks or systems with the same or similar architectures. (T) (2) A functional unit that connects two networks or subnetworks having different characteristics, such as different protocols or different policies concerning security or transmission priority. (3) The combination of machines and programs that provide address translation, name translation, and system services control point (SSCP) rerouting between independent SNA networks to allow those networks to communicate. A gateway consists of one gateway NCP and at least one gateway VTAM. (4) In the IBM Token-Ring Network, a device and its associated software that connect a local area network to another local area network or a host that uses different logical link protocols. (5) In the AIX operating system, an entity that operates above the link layer and translates, when required, the interface and protocol used by one network into those used by another distinct network. (6) In TCP/IP, synonym for *router*.

GDS. General data stream.

general data stream (GDS). The data stream used for conversations in LU 6.2 sessions.

general data stream (GDS) variable. A type of RU substructure that is preceded by an identifier and a length field and includes either application data, user control data, or SNA-defined control data.

generic unbind. Synonym for *session deactivation request*.

GMT. Greenwich mean time.

graph. A set of vertices and the set of arcs that link pairs of those vertices.

Greenwich mean time (GMT). The mean solar time at the prime meridian (0 degrees longitude) of Greenwich, England. Greenwich mean time is sometimes called *Z time* or *Zulu time*.

Note: Although *Greenwich mean time (GMT)* and *coordinated universal time (UTC)* are sometimes used interchangeably, they are not synonyms. Greenwich mean time is an approximate time. Because the second is no longer defined in terms of astronomical phenomena, the preferred name for this time scale is *coordinated universal time (UTC)*.

group. In the NetView/PC program, to identify a set of application programs that are to run concurrently.

group ID (GID). (1) In RACF, a string of one to eight characters that identifies a group. The first character must be A through Z, #, $, or @. The rest can be A through Z, #, $, @, or 0 through 9. (2) In the AIX operating system, a number that corresponds to a specific group name. The group ID can often be substituted in commands that take a group name as a value.

H

half-duplex (HD, HDX). In data communication, pertaining to transmission in only one direction at a time. Contrast with *duplex*.

half-session. A session-layer component consisting of the combination of data flow control and transmission control components comprising one end of a session. See also *session connector*.

handle. (1) In the Advanced DOS and OS/2 operating systems, a binary value created by the system that identifies a drive, directory, and file so that the file can be found and opened. (2) In the AIX operating system, a data structure that is a temporary local identifier for an object. Allocating a handle creates it. Binding a handle makes it identify an object at a specific location. (3) In OS/400 application programming interfaces, a variable that represents an object.

HD. Half-duplex.

header. (1) System-defined control information that precedes user data. (2) The portion of a message that contains control information for the message such as one or more destination fields, name of the originating station, input sequence number, character string indicating the type of message, and priority level for the message.

header file. Synonym for *include file*.

Help. A choice that gives a user access to helpful information about objects, choices, tasks, and products. A Help choice can appear on a menu bar or as a push button.

help panel. Information displayed by a system in response to a help request from a user.

hexadecimal. (1) Pertaining to a selection, choice, or condition that has 16 possible different values or states. (I) (2) Pertaining to a fixed-radix numeration system, with radix of 16. (I) (3) Pertaining to a system of numbers to the base 16; hexadecimal digits range from 0 through 9 and A through F, where A represents 10 and F represents 15.

highlighting. Emphasizing a display element or segment by modifying its visual attributes. (I) (A)

host. (1) In the Internet suite of protocols, an end system. The end system can be any workstation; it does not have to be a mainframe. (2) See *host processor*.

host LU. In SNA, a logical unit in a host processor; for example, a VTAM application program.

host name. In the Internet suite of protocols, the name given to a machine. Sometimes, “host name” is used to mean *fully qualified domain name (FQDN)*; other times, it is used to mean the most specific subname of a fully qualified domain name. For example, if ralvm7.vnet.ibm.com is the fully qualified domain name, either of the following may be considered the host name:

ralvm7.vnet.ibm.com
ralvm7

host processor. (1) A processor that controls all or part of a user application network. (T) (2) In a network, the processing unit in which the data communication access method resides.

I

I/O. Input/output.

IBM Disk Operating System (DOS). A disk operating system based on MS-DOS that operates with all IBM personal computers.

IBM Operating System/2 (OS/2). An IBM licensed program that can be used as the operating system for personal computers. The OS/2 licensed program can perform multiple tasks at the same time.

icon. (1) A graphic symbol, displayed on a screen, that a user can point to with a device such as a mouse in order to select a particular function or software application. (T) (2) A graphical representation of an object, consisting of an image, image background, and a label.

ID. (1) Identifier. (2) Identification.

IDLC. Integrated data link control.

inactive. (1) Not operational. (2) Pertaining to a node or device not connected or not available for connection to another node or device. (3) In the AIX operating system, pertaining to a window that does not have an input focus. (4) In VTAM, the state of a resource or a major or minor node that has not been activated or for which the VARY INACT command has been issued. (5) Contrast with *active*. (6) See also *inoperative*.

inbound. In communications, data that is received from the network.

include file. A text file that contains declarations used by a group of functions, programs, or users. Synonymous with *header file*.

indirect activation. In VTAM, the activation of a lower-level resource of the resource hierarchy as a result of SCOPE or ISTATUS specifications related to an activation command naming a higher-level resource. Contrast with *direct activation*.

information (I) format. A format used for information transfer.

information (I) frame. A frame in I format used for numbered information transfer.

inhibited. In VTAM, pertaining to a logical unit (LU) that has indicated to its system services control point (SSCP) that it is temporarily not ready to establish LU-LU sessions. An initiate request for a session with an inhibited LU will be rejected by the SSCP. The LU can separately indicate whether this applies to its ability to act as a primary logical unit (PLU) or a secondary logical unit (SLU). See also *disabled* and *enabled*.

INITIATE. A network services request sent from a logical unit (LU) to a system services control point (SSCP) requesting that an LU-LU session be established.

inoperative. (1) The condition of a resource that has been active but is not currently active. A resource may be inoperative for reasons such as the following: (a) it may have failed, (b) it may have received an INOP request, or (c) it may be suspended while a reactivate command is being processed. (2) See also *inactive*.

input/output (I/O). (1) Pertaining to input, output, or both. (A) (2) Pertaining to a device, process, or channel involved in data input, data output, or both.

installation exit. The means specifically described in an IBM software product's documentation by which an IBM software product may be modified by a customer's system programmers to change or extend the functions of the IBM software product. Such modifications consist of exit routines written to replace one or more existing

modules of an IBM software product, or to add one or more modules or subroutines to an IBM software product, for the purpose of modifying or extending the functions of the IBM software product. Synonymous with *installation-wide exit*. See *user exit*.

installation-wide exit. Synonym for *installation exit*.

instance. In the AIX operating system, a concrete realization of an abstract object class. An instance of a widget or a gadget is a specific data structure that contains detailed appearance and behavioral information that is used to generate a specific graphical object on-screen at run time.

INT. Internal trace table.

integrated data link control (IDLC). A full-duplex high-level data link control (HDLC) protocol. It is IBM's implementation of the CCITT Q.922 standard or link access procedure extended (LAPE). IDLC can support point-to-point workstation connections over a full-duplex wide area network (WAN).

interactive. (1) Pertaining to a program or system that alternately accepts input and then responds. An interactive system is conversational, that is, a continuous dialog exists between user and system. Contrast with *batch*. (2) Pertaining to the exchange of information between a user and a computer.

interface. (1) A shared boundary between two functional units, defined by functional characteristics, signal characteristics, or other characteristics, as appropriate. The concept includes the specification of the connection of two devices having different functions. (T) (2) Hardware, software, or both, that links systems, programs, or devices.

intermediate session routing (ISR). A type of routing function within an APPN network node that provides session-level flow control and outage reporting for all sessions that pass through the node but whose end points are elsewhere.

IPR. Isolated pacing response.

K

keyword. (1) In programming languages, a lexical unit that, in certain contexts, characterizes some language construct; for example, in some contexts, IF characterizes an if-statement. A keyword normally has the form of an identifier. (I) (2) One of the predefined words of an artificial language. (A) (3) A significant and informative word in a title or document that describes the content of that document. (4) A name or symbol that identifies a parameter. (5) The part of a command operand that consists of a specific character string (such as DSNAME=). (6) See also *keyword operand*. (7) Contrast with *positional operand*.

keyword operand. (1) An operand that consists of a keyword followed by one or more values (such as DSNAME=HELLO). (2) Contrast with *positional operand*. (3) See also *definition statement*.

keyword parameter. A parameter that consists of a keyword followed by one or more values.

L

LAN. Local area network.

layer. (1) In network architecture, a group of services that is complete from a conceptual point of view, that is one out of a set of hierarchically arranged groups, and that extends across all systems that conform to the network architecture. (T) (2) In the Open Systems Interconnection reference model, one of seven conceptually complete, hierarchically arranged groups of services, functions, and protocols, that extend across all open systems. (T) (3) In SNA, a grouping of related functions that are logically separate from the functions in other groups. Implementation of the functions in one layer can be changed without affecting functions in other layers.

LEN. Low-entry networking.

line. (1) The portion of a data circuit external to data circuit-terminating equipment (DCE), that connects the DCE to a data switching exchange (DSE), that connects a DCE to one or more other DCEs, or that connects a DSE to another DSE. (I) (2) Synonymous with *channel* and *circuit*.

line control discipline. Synonym for *link protocol* and *protocol*.

line discipline. Synonym for *link protocol* and *protocol*.

line switching. Synonym for *circuit switching*.

link. (1) The combination of the link connection (the transmission medium) and two link stations, one at each end of the link connection. A link connection can be shared among multiple links in a multipoint or token-ring configuration. (2) To interconnect items of data or portions of one or more computer programs: for example, the linking of object programs by a linkage editor, linking of data items by pointers. (T) (3) In SNA, synonymous with *data link*.

link-attached. (1) Pertaining to devices that are connected to a controlling unit by a data link. (2) Contrast with *channel-attached*. (3) Synonymous with *remote*.

link connection. (1) The physical equipment providing two-way communication between one link station and one or more other link stations; for example, a telecommunication line and data circuit-terminating equipment (DCE). (2) In SNA, synonymous with *data circuit*.

link connection segment. A portion of the configuration that is located between two resources listed consecutively in the service point command service (SPCS) query link configuration request list.

link protocol. (1) The rules for sending and receiving data at the link level. (2) Synonymous with *line control discipline* and *line discipline*.

link status (LS). Information maintained by local and remote modems.

load. (1) To bring all or part of a computer program into memory from auxiliary storage so that the computer can run the program. (2) To place a diskette into a diskette drive.

local. (1) Pertaining to a device accessed directly without use of a telecommunication line. (2) Contrast with *remote*. (3) Synonym for *channel-attached*.

local area network (LAN). (1) A computer network located on a user's premises within a limited geographical area. Communication within a local area network is not subject to external regulations; however, communication across the LAN boundary may be subject to some form of regulation. (T) (2) A network in which a set of devices are connected to one another for communication and that can be connected to a larger network. (3) See also *Ethernet* and *token ring*. (4) Contrast with *metropolitan area network (MAN)* and *wide area network (WAN)*.

local directory database. That set of resources (LUs) in the network known at a particular node. The resources included are all those in the node's domain as well as any cache entries.

local session identification (LSID). In SNA, a field in an FID3 (format identification type 3) transmission header that contains an indication of the type of session (SSCP-PU, SSCP-LU, or LU-LU) and the local address of the peripheral logical unit (LU) or physical unit (PU).

logged-on operator. (1) A NetView operator station task that requires a terminal and a logged-on user. (2) Contrast with *autotask*.

logic error. In VTAM, an error condition that results from an invalid request (a program logic error).

logical link. (1) A pair of link stations, one in each of two adjacent nodes, and their underlying link connection, providing a single link-layer connection between the two nodes. Multiple logical links can be distinguished while they share the use of the same physical media connecting two nodes. Examples are 802.2 logical links used on local area network (LAN) facilities and LAP E logical links on the same point-to-point physical link between two nodes. The term logical link also includes the multiple X.25 logical channels that share the use of the access link from a DTE to an X.25 network. (2) In APPNTAM, the unidirectional representation in a node of a link.

logical record. (1) A set of related data or words considered to be a record from a logical viewpoint. (T) (2) In VSAM, a unit of information normally pertaining to a single subject; a logical record is the user record requested of or given to the data management function.

logical unit (LU). A type of network accessible unit that enables users to gain access to network resources and communicate with each other.

logical unit of work (LUW). The changes to protected resources that are committed or backed out as a unit as a result of a sync point operation. The protected resources may be distributed among different LUs joined together by conversations.

logon. (1) The procedure by which a user begins a terminal session. (2) In VTAM, an unformatted session-initiation request for a session between two logical units.

logon mode. In VTAM, a subset of session parameters specified in a logon mode table for communication with a logical unit. See also *session parameters*.

low-entry networking (LEN). A capability of nodes to attach directly to one another using basic peer-to-peer protocols to support multiple and parallel sessions between logical units.

low-entry networking (LEN) end node. A LEN node receiving network services from an adjacent APPN network node.

low-entry networking (LEN) node. A node that provides a range of end-user services, attaches directly to other nodes using peer protocols, and derives network services implicitly from an adjacent APPN network node, that is, without the direct use of CP-CP sessions.

LS. Link status.

LSID. Local session identification.

LU. Logical unit.

LU-LU session. A logical connection between two logical units (LUs) in an SNA network that typically provides communication between two users.

LU type. The classification of an LU in terms of the specific subset of SNA protocols and options it supports for a given session, namely:

The mandatory and optional values allowed in the session activation request

The usage of data stream controls, function management headers (FMHs), request unit parameters, and sense data values

Presentation services protocols such as those associated with FMH usage

LU types 0, 1, 2, 3, 4, 6.1, 6.2, and 7 are defined.

LU 6.2. A type of logical unit that supports general communication between programs in a distributed processing environment. LU 6.2 is characterized by (a) a peer relationship between session partners, (b) efficient utilization of a session for multiple transactions, (c) comprehensive end-to-end error processing, and (d) a generic application programming interface (API) consisting of structured verbs that are mapped into a product implementation.

LU 6.2 verb. A syntactical unit in the LU 6.2 application programming interface representing an operation.

LUS. Logical unit services.

LUW. Logical unit of work.

M

macroinstruction. (1) An instruction in a source language that is to be replaced by a defined sequence of instructions in the same source language and that may also specify values for parameters in the replaced instructions. (T) (2) In assembler programming, an assembler language statement that causes the assembler to process a predefined set of statements called a macro definition. The statements normally produced from the macro definition replace the macroinstruction in the program. (3) See also *definition statement*.

maintenance services. In SNA, one of the types of network services in system services control points (SSCPs) and physical units (PUs). Maintenance services provide facilities for testing links and nodes and for collecting and recording error information.

major node. In VTAM, a set of resources that can be activated and deactivated as a group. See also *minor node*.

MAN. Metropolitan area network.

management services (MS). (1) One of the types of network services in control points (CPs) and physical units (PUs). Management services are the services provided to assist in the management of SNA networks, such as problem management, performance and accounting management, configuration management, and change management. (2) Services that assist in the management of systems and networks in areas such as problem management, performance management, business management, operations management, configuration management, and change management.

manager. (1) In systems management, a user that, for a particular interaction, has assumed a manager role. (2) An entity that monitors or controls one or more managed objects by (a) receiving notifications regarding the objects and (b) requesting management operations to modify or query the objects. (3) A system that assumes a manager role.

mapped conversation. An LU 6.2 conversation type specified by the allocating transaction program. Transaction programs using a mapped conversation can exchange messages of arbitrary format regardless of the underlying data stream. System-defined or user-defined mappers can perform data transformation for the transaction programs. See also *conversation*. Contrast with *basic conversation*.

mapping. The process of converting data that is transmitted in one format by the sender into the data format that can be accepted by the receiver.

matching. In MPTN architecture, pertaining to the relationship between peer transport users or peer transport providers that use the same user protocols or the same transport protocols.

memory. All of the addressable storage space in a processing unit and other internal storages that is used to execute instructions. (T)

menu. (1) A list of options displayed to the user by a data processing system, from which the user can select an action to be initiated. (T) (2) In text processing, a list of choices displayed to the user by a text processor from which the user can select an action to be initiated. (T) (3) A list of choices that can be applied to an object. A menu can contain choices that are not available for selection in certain contexts. Those choices are indicated by reduced contrast.

message. (1) An assembly of characters and sometimes control codes that is transferred as an entity from an originator to one or more recipients. A message consists of two parts: envelope and content. (T) (2) A communication sent from a person

or program to another person or program. (3) In VTAM, the amount of function management data (FMD) transferred to VTAM by the application program with one SEND request.

message processing facility (MPF). An MVS facility that controls message display and message processing.

message queue. In IMS/VS, the data set on which messages are queued before being processed by an application program or sent to a terminal.

message switching. The process of receiving a message, storing it, and forwarding it to its destination unaltered. (T)

method. In the NetView program, the code that runs within the Resource Object Data Manager (RODM) address space. Methods are used to implement behavior specified by an operation.

metric. In Internet communications, a value, associated with a route, which is used to discriminate between multiple exit or entry points to the same autonomous system. The route with the lowest metric is preferred.

metropolitan area network (MAN). A network formed by the interconnection of two or more networks which may operate at higher speed than those networks, may cross administrative boundaries, and may use multiple access methods. (T) Contrast with *local area network (LAN)* and *wide area network (WAN)*.

MID. Machine identifier.

migration. The installation of a new version or release of a program to replace an earlier version or release.

minor node. In VTAM, a uniquely defined resource within a major node.

mode. See *mode name*.

mode name. The name used by the initiator of a session to designate the characteristics desired for the session, such as traffic pacing values, message-length limits, sync point and cryptography options, and the class of service within the transport network.

modem (modulator/demodulator). (1) A functional unit that modulates and demodulates signals. One of the functions of a modem is to enable digital data to be transmitted over analog transmission facilities. (T) (A) (2) A device that converts digital data from a computer to an analog signal that can be transmitted on a telecommunication line, and converts the analog signal received to data for the computer.

module. A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to or output from an assembler, compiler, linkage editor, or executive routine. (A)

monitor. (1) A device that observes and records selected activities within a data processing system for analysis. Possible uses are to indicate significant departure from the norm, or to determine levels of utilization of particular functional units. (T) (2) Software or hardware that observes, supervises, controls, or verifies operations of a system. (A) (3) The function required to initiate the transmission of a token on the ring and to provide soft-error recovery in case of lost tokens, circulating frames, or other difficulties. The capability is present in all ring stations. (4) In the NetView Graphic Monitor Facility, to open a view that can receive status changes from the NetView program. Problem determination and correction can be performed directly from the view. Contrast with *browse*.

mouse. A commonly used pointing device, containing one or more buttons, with which a user can interact with a product or the operating environment.

MPF. Message processing facility.

MS. Management services.

MSG. Console messages.

N

NAU. (1) Network accessible unit. (2) Network addressable unit.

NC. Network control.

NCP/EP definition facility (NDF). A program that is part of System Support Programs (SSP) and that is used to generate a load module for a partitioned emulation program (PEP), a Network Control Program (NCP), or an Emulation Program (EP).

NDF. NCP/EP definition facility.

negative response (NR). In SNA, a response indicating that a request did not arrive successfully or was not processed successfully by the receiver. Contrast with *positive response*.

negotiable BIND. In SNA, a capability that allows two half-sessions to negotiate the parameters of a session when the session is being activated.

negotiation. The process of deciding what packet size to transmit between a network and a 3710 Network Controller.

NETID. See *network identifier*.

NetView for AIX Service Point. (1) Formerly known as the *AIX NetView Service Point*. (2) An IBM licensed program that operates in the AIX and UNIX environments. It functions as a gateway in an unattended environment.

NetView-NetView task (NNT). The task under which a cross-domain NetView operator session runs. See *operator station task*.

network. (1) An arrangement of nodes and connecting branches. (T) (2) A configuration of data processing devices and software connected for information interchange. (3) A group of nodes and the links interconnecting them.

network accessible unit (NAU). A logical unit (LU), physical unit (PU), control point (CP), or system services control point (SSCP). It is the origin or the destination of information transmitted by the path control network. Synonymous with *network addressable unit*.

network address. (1) According to ISO 7498-3, a name, unambiguous within the OSI environment, that identifies a set of network service access points. (2) An identifier for a node, station, or unit of equipment in a network. (3) In a subarea network, an address, consisting of subarea and element fields, that identifies a link, link station, physical unit, logical unit, or system services control point. Subarea nodes use network addresses; peripheral nodes use local addresses or local-form session identifiers (LFSIDs). The boundary function in the subarea node to which a peripheral node is attached transforms local addresses or LFSIDs to network addresses and vice versa. Contrast with *network name*.

network addressable unit (NAU). Synonym for *network accessible unit*.

network architecture. The logical structure and operating principles of a computer network. (T)

Note: The operating principles of a network include those of services, functions, and protocols.

network congestion. An undesirable overload condition caused by traffic in excess of what a network can handle.

network control (NC). In SNA, a request/response unit (RU) category used for requests and responses exchanged between physical units (PUs) for such purposes as activating and deactivating explicit and virtual routes and sending load modules to adjust peripheral nodes. See also *data flow control*, *function management data*, and *session control*.

network directory database. Synonym for *distributed directory database*.

network identifier. (1) In TCP/IP, that part of the IP address that defines a network. The length of the network ID depends on the type of network class (A, B, or C). (2) A 1- to 8-byte customer-selected name or an 8-byte IBM-registered name that uniquely identifies a specific subnetwork. (3) In MPTN architecture, the address qualifier of a transport provider address that identifies a group of nodes according to the network in which they reside.

network name. (1) The symbolic identifier by which users refer to a network accessible unit, a link, or a link station within a given subnetwork. In APPN networks, network names are also used for routing purposes. Contrast with *network address*. (2) In a multiple-domain network, the name of the APPL statement defining a VTAM application program. The network name must be unique across domains. Contrast with *ACB name*. See *uninterpreted name*.

network node domain. An APPN network-node control point, its attached links, the network resources for which it answers directory search requests (namely, its local LUs and adjacent LEN end nodes), the adjacent APPN end nodes with which it exchanges directory search requests and replies, and other resources (such as a local storage device) associated with its own node or an adjacent end node for which it provides management services.

network operator. (1) A person who controls the operation of all or part of a network. (2) In a multiple-domain network, a person or program responsible for controlling all domains. (3) See also *domain operator*.

network-qualified name. In SNA, a name that uniquely identifies a specific resource (such as an LU or a CP) within a specific network. It consists of a network identifier and a resource name, each of which is a 1- to 8-byte symbol string. Synonymous with *fully qualified name*.

network services. (1) The services within network accessible units that control network operation through SSCP-SSCP, SSCP-PU, SSCP-LU, and CP-CP sessions. (2) The session services (directory and route-selection functions) and management services provided by an APPN network-node control point to its domain.

no response. In SNA, a protocol requested in the form-of-response-requested field of the request header that directs the receiver of the request not to return any response, regardless of whether or not the request is received and processed successfully. Contrast with *definite response* and *exception response*.

node. (1) In a network, a point at which one or more functional units connect channels or data circuits. (I)

(2) In network topology, the point at an end of a branch. (T) (3) The representation of a state or an event by means of a point on a diagram. (A) (4) In a tree structure, a point at which subordinate items of data originate. (A) (5) Any device, attached to a network, that transmits and receives data. (6) An endpoint of a link or a junction common to two or more links in a network. Nodes can be processors, communication controllers, cluster controllers, or terminals. Nodes can vary in routing and other functional capabilities. (7) In VTAM, a point in a network defined by a symbolic name. See *major node* and *minor node*.

node name. In VTAM, the symbolic name assigned to a specific major or minor node during network definition.

nonblocking mode. (1) A way of requesting a service over an interface so that if the request cannot be completed immediately, the requesting process is able to continue and is not suspended. (2) Contrast with *blocking mode*.

normal flow. In SNA, a data flow designated in the transmission header (TH) that is used primarily to carry end-user data. The rate at which requests flow on the normal flow can be regulated by session-level pacing. Normal and expedited flows move in both the primary-to-secondary and secondary-to-primary directions. Contrast with *expedited flow*.

notification. (1) An unscheduled, spontaneously generated report of an event that has occurred. (2) In systems management, information emitted by a managed object relating to an event that has occurred within the managed object, such as a threshold violation or a change in configuration status.

NOTIFY. A network services request that is sent by a system services control point (SSCP) to a logical unit (LU) to inform the LU of the status of a procedure requested by the LU.

O

OAF. Origin address field.

OAF'. Origin address field prime.

object. (1) In object-oriented design or programming, an abstraction consisting of data and the operations associated with that data. See also *class*. (2) An item that a user can manipulate as a single unit to perform a task. An object can appear as text, an icon, or both.

ODAI. Origin-Destination Assignor indicator, a bit in a FID2 transmission header used to divide the address space so that an address space manager (ASM) in one node may use all possible combinations of OAF′, DAF′ with the ODAI having one setting and the ASM in the adjacent node may use all possible combinations of OAF′, DAF′ with the ODAI having the complementary setting.

Off. A choice that appears in the cascaded menu from the Refresh choice. It sets the refresh function to off.

offline. (1) Pertaining to the operation of a functional unit that takes place either independently of, or in parallel with, the main operation of a computer. (T) (2) Neither controlled by, nor communicating with, a computer. Contrast with *online*.

offset. The number of measuring units from an arbitrary starting point in a record, area, or control block, to some other point.

OIA. Operator information area.

OK. A push button that accepts the information in a window and closes it. If the window contains changed information, those changes are applied before the window is closed.

On. A choice that appears in a cascaded menu from the Refresh choice. It immediately refreshes the view in a window.

one-way bracket. A bracket in which data is sent from one NAU to another in a single chain with begin bracket, conditional end bracket, and exception response requested. When one-way brackets are used on CP-CP sessions, they are always sent on the contention-winner session.

online. (1) Pertaining to the operation of a functional unit when under the direct control of the computer. (T) (2) Pertaining to a user's ability to interact with a computer. (A) (3) Pertaining to a user's access to a computer via a terminal. (A) (4) Controlled by, or communicating with, a computer. (5) Contrast with *offline*.

open. (1) A break in an electrical circuit. (2) To make an adapter ready for use.

Open. A choice that leads to a window in which users can select the object they want to open.

operand. (1) An entity on which an operation is performed. (I) (2) That which is operated upon. An operand is usually identified by an address part of an instruction. (A) (3) Information entered with a command name to define the data on which a command processor operates and to control the execution of the command processor. (4) An expression to whose value an operator is applied. See also *definition statement*, *keyword*, *keyword parameter*, and *parameter*.

operating system (OS). Software that controls the execution of programs and that may provide services such as resource allocation, scheduling, input/output control, and data management. Although operating systems are predominantly software, partial hardware implementations are possible. (T)

operation. In object-oriented design or programming, a service that can be requested at the boundary of an object. Operations include modifying an object or disclosing information about an object.

operator. (1) In a language statement, the lexical entity that indicates the action to be performed on operands. See also *definition statement*. (2) A person or program responsible for managing activities controlled by a given piece of software such as MVS, the NetView program, or IMS. (3) A person who operates a device. (4) A person who keeps a system running. (5) See also *autotask*, *logged-on operator*, *network operator*, and *operator station task*.

operator information area (OIA). The area near the bottom of the display area where terminal or system status information is displayed.

operator station task (OST). The NetView task that establishes and maintains the online session with the network operator. There is one operator station task for each network operator who logs on to the NetView program. See *NetView-NetView task*.

option set. (1) A set of functions that may be supported by products that implement a particular architecture. A product may support any number of option sets or none. For each option set supported, all functions in that set are supported. (2) Contrast with *base set*.

origin. An external logical unit (LU) or application program from which a message or other data originates. See also *destination*.

origin address field (OAF). In SNA, a field in a FID0 or FID1 transmission header that contains the address of the originating network accessible unit (NAU). Contrast with *destination address field (DAF)*. See also *format identification (FID) field* and *local session identification (LSID)*.

OS. Operating system.

outbound. In communications, data that is transmitted to the network.

output. Pertaining to a device, process, or channel involved in an output process, or to the associated data or states. The word “output” may be used in place of “output data,” “output signal,” “output process,” when such a usage is clear in a given context. (T)

overlay. A collection of predefined data, such as lines, shading, text, boxes, or logos, that can be merged with variable data on a page while printing.

P

pacing. (1) A technique by which a receiving component controls the rate of transmission of a sending component to prevent overrun or congestion. (2) See *receive pacing*, *send pacing*, *session-level pacing*, and *virtual route (VR) pacing*. (3) See also *flow control*.

pacing group. Synonym for *pacing window*.

pacing response. In SNA, an indicator that signifies the readiness of a receiving component to accept another pacing group. The indicator is carried in a response header (RH) for session-level pacing and in a transmission header (TH) for virtual route pacing.

pacing window. (1) The path information units (PIUs) that can be transmitted on a virtual route before a virtual-route pacing response is received, indicating that the virtual route receiver is ready to receive more PIUs on the route. (2) The requests that can be transmitted on the normal flow in one direction on a session before a session-level pacing response is received, indicating that the receiver is ready to accept the next group of requests. (3) Synonymous with *pacing group*.

packet assembler/disassembler (PAD). A functional unit that enables data terminal equipment (DTEs) not equipped for packet switching to access a packet switched network. (T) (A)

packet mode operation. Synonym for *packet switching*.

packet switching. (1) The process of routing and transferring data by means of addressed packets so that a channel is occupied only during transmission of a packet. On completion of the transmission, the channel is made available for transfer of other packets. (I) (2) Synonymous with *packet mode operation*. See also *circuit switching*.

PAD. Packet assembler/disassembler.

page. (1) In a virtual storage system, a fixed-length block that has a virtual address and is transferred as a unit between real storage and auxiliary storage. (I) (A) (2) A printed form. (3) The information displayed at the same time on the screen of a display device. (4) In VSE, a fixed-length block of instructions, data, or both that can be located in processor storage or in the page data set on disk. (5) To replace the information displayed on the screen with prior or subsequent information from the same file.

panel. (1) See *window*. (2) A formatted display of information that appears on a display screen. See *help panel* and *task panel*. (3) In computer graphics, a display image that defines the locations and characteristics of display fields on a display surface.

parallel. (1) Pertaining to a process in which all events occur within the same interval of time, each handled by a separate but similar functional unit; for example, the parallel transmission of the bits of a computer word along the lines of an internal bus. (T) (2) Pertaining to concurrent or simultaneous operation of two or more devices or to concurrent performance of two or more activities in a single device. (A) (3) Pertaining to concurrent or simultaneous occurrence of two or more related activities in multiple devices or channels. (A) (4) Pertaining to the simultaneity of two or more processes. (A) (5) Pertaining to the simultaneous processing of the individual parts of a whole, such as the bits of a character and the characters of a word, using separate facilities for the various parts. (A) (6) Contrast with *serial*.

parallel sessions. Two or more concurrently active sessions between the same two network accessible units (NAUs) using different pairs of network addresses or local-form session identifiers. Each session can have independent session parameters.

parameter. (1) A variable that is given a constant value for a specified application and that may denote the application. (I) (A) (2) In Basic CUA architecture, a variable used in conjunction with a command to affect its result. (3) An item in a menu for which the user specifies a value or for which the system provides a value when the menu is interpreted. (4) Data passed to a program or procedure by a user or another program, namely as an operand in a language statement, as an item in a menu, or as a shared data structure. (5) See also *keyword*, *keyword parameter*, and *operand*.

parameters. In NetDA/2, the set of restrictions that affect only the output of a network design. A change in a parameter value does not change the input to the network design. Contrast with *constraints*.

parent. A process that has spawned a child process using the fork primitive. Contrast with *child*.

parent process. In the AIX and OS/2 operating systems, a process that creates other processes. Contrast with *child process*.

password. (1) A value used in authentication or a value used to establish membership in a set of people having specific privileges. (2) A unique string of characters known to a computer system and to a user, who must specify the character string to gain access to a system and to the information stored within it. (3) In computer security, a string of characters known to the computer system and a user, who must specify it to gain full or limited access to a system and to the data stored within it.

path. (1) In a network, any route between any two nodes. A path may include more than one branch. (T) (2) The route used to locate files; the storage location of a file. A fully qualified path lists the drive identifier, directory name, subdirectory name (if any), and file name with the associated extension. (3) The series of transport network components (path control and data link control) that are traversed by the information exchanged between two network accessible units. See also *explicit route (ER)*, *route extension (REX)*, and *virtual route (VR)*. (4) In VTAM when defining a switched major node, a potential dial-out port that can be used to reach that node. (5) In the NetView/PC program, a complete line in a configuration that contains all of the resources in the service point command service (SPCS) query link configuration request list.

path control (PC). The function that routes message units between network accessible units in the network and provides the paths between them. It converts the basic information units (BIUs) from transmission control (possibly segmenting them) into path information units (PIUs) and exchanges basic transmission units containing one or more PIUs with data link control. Path control differs by node type: some nodes (APPN nodes, for example) use locally generated session identifiers for routing, and others (subarea nodes) use network addresses for routing.

path information unit (PIU). A message unit consisting of a transmission header (TH) alone, or a TH followed by a basic information unit (BIU) or a BIU segment.

PC. (1) Personal computer. (2) Path control.

peer. In network architecture, any functional unit that is in the same layer as another entity. (T)

pending active session. In VTAM, the state of an LU-LU session recorded by the system services control point (SSCP) when it finds both logical units (LUs) available and has sent a CINIT request to the primary logical unit (PLU) of the requested session.

performance error. Synonym for *temporary error*.

peripheral LU. In SNA, a logical unit in a peripheral node. Contrast with *subarea LU*.

peripheral node. A node that uses local addresses for routing and therefore is not affected by changes in network addresses. A peripheral node requires boundary-function assistance from an adjacent subarea

node. A peripheral node can be a type 1, 2.0, or 2.1 node connected to a subarea boundary node.

peripheral PU. In SNA, a physical unit in a peripheral node. Contrast with *subarea PU*.

permanent error. An error that cannot be resolved by error recovery programs. Contrast with *temporary error*.

permanent virtual circuit (PVC). (1) In X.25 and frame-relay communications, a virtual circuit that has a logical channel permanently assigned to it at each data terminal equipment (DTE). Call-establishment protocols are not required. Contrast with *switched virtual circuit (SVC)*. (2) The logical connection between two frame-relay terminating equipment stations, either directly or through one or more frame-relay frame handlers. A PVC consists of one or more PVC segments.

persistent verification. In VTAM, a security function that enables two logical units to verify the identity of each other for the initial conversation on a session and to assume that future conversations are verified for the duration of the session.

personal computer (PC). (1) A microcomputer primarily intended for stand-alone use by an individual. (T) (2) A desktop, floor-standing, or portable microcomputer that usually consists of a system unit, a display monitor, a keyboard, one or more diskette drives, internal fixed-disk storage, and an optional printer. PCs are designed primarily for stand-alone operation but may be connected to mainframes or networks.

physical circuit. A circuit established without multiplexing. See also *data circuit*. Contrast with *virtual circuit*.

physical connection. (1) A connection that establishes an electrical circuit. (2) A point-to-point or multipoint connection. (3) Synonymous with *connection*.

physical unit (PU). (1) The component that manages and monitors the resources (such as attached links and adjacent link stations) associated with a node, as requested by an SSCP via an SSCP-PU session. An SSCP activates a session with the physical unit in order to indirectly manage, through the PU, resources of the node such as attached links. This term applies to type 2.0, type 4, and type 5 nodes only. (2) See also *peripheral PU* and *subarea PU*.

physical unit (PU) services. In SNA, the components within a physical unit (PU) that provide configuration services and maintenance services for SSCP-PU sessions.

PIP. Program initialization parameters.

PLU. Primary logical unit.

pointer. (1) A data element that indicates the location of another data element. (T) (2) An identifier that indicates the location of an item of data. (A)

pop. To remove an item from the top of a pushdown list. Contrast with *push*.

POP. Post Office Protocol.

positional operand. (1) An operand in a language statement that has a fixed position. (2) Contrast with *keyword operand*. (3) See also *definition statement*.

positive response. In SNA, a response indicating that a request was received and processed. Contrast with *negative response*.

post. (1) In the AIX operating system, the action required to make a pop-up or pull-down menu appear. This action is normally a click or a button press on one of the mouse buttons. (2) To enter a unit of information on a record. (3) To note the occurrence of an event. (4) To add information to a record in order to keep the record current.

POST. Power-on self test.

Post Office Protocol (POP). A protocol used for exchanging network mail and accessing mailboxes.

Prepare. A presentation services header that flows as part of commit processing, indicating the partner has begun the first phase of the two-phase commit process.

primary half-session. In SNA, the half-session that sends the session activation request. See also *primary logical unit (PLU)*. Contrast with *secondary half-session*.

primary logical unit (PLU). In SNA, the logical unit (LU) that sends the BIND to activate a session with its partner LU. Contrast with *secondary logical unit (SLU)*.

problem determination. The process of determining the source of a problem; for example, a program component, machine failure, telecommunication facilities, user or contractor-installed programs or equipment, environmental failure such as a power loss, or user error.

procedure. (1) In a programming language, a block, with or without formal parameters, whose execution is invoked by means of a procedure call. (I) (2) The description of the course of action taken for the solution of a problem. (A)

process. (1) To perform operations on data in a process. (I) (A) (2) In data processing, the course

of events that occurs during the execution of all or part of a program. (T) (3) A course of the events defined by its purpose or by its effect, achieved under given conditions. (4) Any operation or combination of operations on data. (5) A function being performed or waiting to be performed.

process identification number (process ID). A unique number assigned to a process by the operating system. The number is used internally by processes to communicate.

processor. In a computer, a functional unit that interprets and executes instructions. A processor consists of at least an instruction control unit and an arithmetic and logic unit. (T)

profile. Data that describes the significant characteristics of a user, a group of users, or one or more computer resources.

program. (1) A sequence of instructions suitable for processing by a computer. Processing may include the use of an assembler, a compiler, an interpreter, or a translator to prepare the program for execution, as well as to execute it. (I) (2) In programming languages, a logical assembly of one or more interrelated modules. (I) (3) To design, write, and test computer programs. (I) (A)

program initialization parameters (PIP). The initial parameter values passed to a target program as input or used to set up the process environment.

protocol. (1) A set of semantic and syntactic rules that determine the behavior of functional units in achieving communication. (I) (2) In Open Systems Interconnection architecture, a set of semantic and syntactic rules that determine the behavior of entities in the same layer in performing communication functions. (T) (3) In SNA, the meanings of, and the sequencing rules for, requests and responses used for managing the network, transferring data, and synchronizing the states of network components. Synonymous with *line control discipline* and *line discipline*. See *bracket protocol* and *link protocol*.

PU. Physical unit.

push. To add an item to the top of a pushdown list. Contrast with *pop*.

pushdown list. (1) A list constructed and maintained so that the next data element to be retrieved is the most recently stored. (T) (2) Synonymous with *stack*.

PUT. Program update tape.

PVC. Permanent virtual circuit.

Q

queue. (1) A list constructed and maintained so that the next data element to be retrieved is the one stored first. (T) (2) A line or list of items waiting to be processed; for example, work to be performed or messages to be displayed. (3) To arrange in or form a queue.

quiesce. (1) To end a process by allowing operations to complete normally. (2) To request that a node stop sending synchronous-flow messages.

quiesce protocol. In VTAM, a method of communicating in one direction at a time. Either the primary logical unit (PLU) or the secondary logical unit (SLU) assumes the exclusive right to send normal-flow requests, and the other node does not send such requests. When the sender wants to receive, it releases the other node from its quiesced state.

R

RAM. Random access memory. (A)

RAM semaphore. In the OS/2 operating system, a semaphore used to serialize different threads of a single process. Contrast with *system semaphore*.

random access memory (RAM). (1) A storage device in which data can be written and read. (2) Deprecated term for *direct access storage device (DASD)*. (T) (3) A storage device into which data is entered and from which data is retrieved in a nonsequential manner. (4) See also *direct access storage*.

read-only. A type of access to data that allows data to be read but not copied, printed, or modified.

receive pacing. In SNA, the pacing of message units that a component is receiving. Contrast with *send pacing*.

record. (1) In programming languages, an aggregate that consists of data objects, possibly with different attributes, that usually have identifiers attached to them. In some programming languages, records are called structures. (I) (2) A set of data treated as a unit. (T) (3) A set of one or more related data items grouped for processing. (4) In VTAM, the unit of data transmission for record mode. A record represents whatever amount of data the transmitting node chooses to send.

reentrant. The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

Refresh. A cascading choice that gives a user access to other choices (*On* and *Off*) that control whether changes made to underlying data in a window are displayed immediately, not displayed at all, or displayed at a later time.

release. (1) A distribution of a new product or new function and APAR fixes for an existing product. Normally, programming support for the prior release is discontinued after some specified period of time following availability of a new release. The first version of a product is announced as Release 1, Modification Level 0. (2) In VTAM, to relinquish control of resources (communication controllers or physical units). See also *resource takeover*. Contrast with *acquire*.

remote. (1) Pertaining to a system, program, or device that is accessed through a telecommunication line. (2) Synonym for *link-attached*. (3) Contrast with *local*.

remote host. Any host on a network except the host at which a particular operator is working. Synonymous with *foreign host*.

request header (RH). The control information that precedes a request unit (RU). See also *request/response header (RH)*.

request to send (RTS). In data communication, a signal raised by data terminal equipment (DTE), while the data terminal is ready, to request facilities from data circuit-terminating equipment (DCE) so that data can be sent. Contrast with *clear to send (CTS)*.

request unit (RU). A message unit that contains control information, end-user data, or both.

request/response header (RH). Control information associated with a particular RU. The RH precedes the request/response unit (RU) and specifies the type of RU (request unit or response unit).

request/response unit (RU). A generic term for a request unit or a response unit. See *request unit (RU)* and *response unit (RU)*.

requester. Synonym for *client*.

reset. (1) To cause a counter to take the state corresponding to a specified initial number. (I) (A) (2) To put all or part of a data processing device back into a prescribed state. (I) (A) (3) On a virtual circuit, reinitialization of data flow control. At reset, all data in transit are eliminated.

resource. (1) Any facility of a computing system or operating system required by a job or task, and including main storage, input/output devices, the processing unit, data sets, and control or processing programs. (2) In the NetView program, any hardware or software that provides function to the network.

resource takeover. In VTAM, an action initiated by a network operator to transfer control of resources from one domain to another without breaking the connections or disrupting existing LU-LU sessions on the connection. See also *acquire* and *release*.

resource types. In the NetView program, a concept to describe the organization of panels. Resource types are defined as central processing unit, channel, control unit, and I/O device for one category; and communication controller, adapter, link, cluster controller, and terminal for another category. Resource types are combined with data types and display types to describe display organization. See also *data types* and *display types*.

response. (1) In data communication, a reply represented in the control field of a response frame. It advises the primary or combined station of the action taken by the secondary or other combined station to one or more commands. (2) See also *command*.

response header (RH). (1) A header, optionally followed by a response unit (RU), that indicates whether the response is positive or negative and that may contain a pacing response. (2) See also *negative response*, *pacing response*, and *positive response*.

response indicator. In the AS/400 system, a 1-character field passed with an input record from the system to a program to provide information about the data record or actions taken by the workstation user.

response unit (RU). A message unit that acknowledges a request unit. It may contain prefix information received in a request unit. If positive, the response unit may contain additional information (such as session parameters in response to BIND SESSION). If negative, the response unit contains sense data defining the exception condition.

resync. Recovery processing that is performed by sync point services when a failure of a session, transaction program, or LU occurs during sync point processing. The purpose of resync is to return protected resources to consistent states.

return code. (1) A code used to influence the execution of succeeding instructions. (A) (2) A value returned to a program to indicate the results of an operation requested by that program.

REX. Route extension.

RH. Request/response header.

ring. See *ring network*.

ring network. (1) A network in which every node has exactly two branches connected to it and in which there

are exactly two paths between any two nodes. (T) (2) A network configuration in which devices are connected by unidirectional transmission links to form a closed path.

route. (1) An ordered sequence of nodes and transmission groups (TGs) that represent a path from an origin node to a destination node traversed by the traffic exchanged between them. (2) The path that network traffic uses to get from source to destination.

route daemon. A program that runs under 4BSD UNIX to propagate route information among machines on a local area network. Also referred to as *routed* (pronounced "route-d").

route extension (REX). In SNA, the path control network components, including a peripheral link, that make up the portion of a path between a subarea node and a network addressable unit (NAU) in an adjacent peripheral node. See also *explicit route (ER)*, *path*, and *virtual route (VR)*.

routed. Pronounced "route-d." See *route daemon*.

router. (1) A computer that determines the path of network traffic flow. The path selection is made from several paths based on information obtained from specific protocols, algorithms that attempt to identify the shortest or best path, and other criteria such as metrics or protocol-specific destination addresses. (2) An attaching device that connects two LAN segments, which use similar or different architectures, at the reference model network layer. (3) In OSI terminology, a function that determines a path by which an entity can be reached. (4) In TCP/IP, synonymous with *gateway*. (5) Contrast with *bridge*.

routine. A program, or part of a program, that may have some general or frequent use. (T)

routing. (1) The process of determining the path to be used for transmission of a message over a network. (T) (2) The assignment of the path by which a message is to reach its destination. (3) In SNA, the forwarding of a message unit along a particular path through a network, as determined by parameters carried in the message unit, such as the destination network address in a transmission header.

RQD. Request discontact.

RTS. Request to send.

RU. Request/response unit.

RU chain. In SNA, a set of related request/response units (RUs) that are consecutively transmitted on a particular normal or expedited data flow. The request RU chain is the unit of recovery: if one of the RUs in the chain cannot be processed, the entire chain is discarded. Each RU belongs to only one chain, which has a beginning and an end indicated by means of control bits in request/response headers within the RU chain. Each RU can be designated as first-in-chain (FIC), last-in-chain (LIC), middle-in-chain (MIC), or only-in-chain (OIC). Response units and expedited-flow request units are always sent as only-in-chain.

S

SBCS. Single-byte character set.

SC. Session control.

SDLC. Synchronous Data Link Control.

SDT. Start data traffic.

secondary half-session. In SNA, the half-session that receives the session-activation request. See also *secondary logical unit (SLU)*. Contrast with *primary half-session*.

secondary logical unit (SLU). In SNA, the logical unit (LU) that contains the secondary half-session for a particular LU-LU session. An LU may contain secondary and primary half-sessions for different active LU-LU sessions. Contrast with *primary logical unit (PLU)*.

secondary logical unit (SLU) key. A key-encrypting key used to protect a session cryptography key during its transmission to the secondary half-session.

segment. (1) A portion of a computer program that may be executed without the entire computer program being resident in main storage. (T) (2) A group of display elements. (3) A section of cable between components or devices. A segment may consist of a single patch cable, several patch cables that are connected, or a combination of building cable and patch cables that are connected. (4) In Internet communications, the unit of transfer between TCP functions in different machines. Each segment contains control and data fields; the current byte-stream position and actual data bytes are identified along with a checksum to validate received data. (5) In IMS/VS, the unit of access to a database; for the database system, the smallest amount of data that can be transferred by one DL/I operation. For input terminal operations using the DC feature, a segment is defined by the particular terminal type and is obtained by the application program with one call. (6) In the Enhanced X-Windows Toolkit, one or more lines that are drawn but not necessarily connected at the endpoints. (7) In LANs or WANs, a subset of nodes in a network or subnet that are connected by a common physical medium. (8) Synonym for *BIU segment*. (9) See also *link connection segment*.

segmentation. A process by which path control (PC) divides basic information units (BIUs) into smaller units, called BIU segments, to accommodate smaller buffer sizes in adjacent nodes. Both segmentation and segment assembly are optional PC features. The support for either or both is indicated in the BIND request and response.

segmenting. In OSI, a function performed by a layer to map one protocol data unit (PDU) from the layer it supports into multiple PDUs.

select. To explicitly identify one or more objects to which a subsequent choice will apply.

selection. The process of explicitly identifying one or more objects to which a subsequent choice will apply.

semaphore. (1) An indicator used to control access to a file; for example, in a multiuser application, a flag that prevents simultaneous access to a file. (2) An entity used to control access to system resources. Processes can be locked to a resource with semaphores if the processes follow certain programming conventions.

send pacing. In SNA, the pacing of message units that a component is sending. Contrast with *receive pacing*.

sequence number. (1) In communications, a number assigned to a particular frame or packet to control the transmission flow and receipt of data. (2) A numerical value assigned by VTAM to each message exchanged between two nodes. The value (one for messages sent from the application program to the logical unit and another for messages sent from the logical unit to the application program) increases by one for each successive message transmitted unless it is reset by the application program with a set and test sequence numbers (STSN) indicator.

serial. (1) Pertaining to a process in which all events occur one after the other; for example, serial transmission of the bits of a character according to V24 CCITT protocol. (T) (2) Pertaining to the sequential or consecutive occurrence of two or more related activities in a single device or channel. (A) (3) Pertaining to the sequential processing of the individual parts of a whole, such as the bits of a character or the characters of a word, using the same facilities for successive parts. (A) (4) Contrast with *parallel*.

server. (1) A functional unit that provides shared services to workstations over a network; for example, a file server, a print server, a mail server. (T) (2) In a network, a data station that provides facilities to other stations; for example, a file server, a print server, a mail server. (A) (3) In the AIX operating system, an application program that usually runs in the background and is controlled by the system program controller. (4) In the Enhanced X-Windows Toolkit, a program that provides the basic windowing mechanism. It handles interprocess communication (IPC) connections from clients, demultiplexes graphics requests onto screens, and multiplexes input back to clients.

service point (SP). (1) An entry point that supports applications that provide network management for resources not under the direct control of itself as an entry point. Each resource is either under the direct control of another entry point or not under the direct control of any entry point. A service point accessing these resources is not required to use SNA sessions (unlike a focal point). A service point is needed when entry point support is not yet available for some network management function. (2) In NetView for AIX, see *NetView for AIX Service Point*.

session. (1) In network architecture, for the purpose of data communication between functional units, all the activities which take place during the establishment, maintenance, and release of the connection. (T) (2) A logical connection between two network accessible units (NAUs) that can be activated, tailored to provide various protocols, and deactivated, as requested. Each session is uniquely identified in a transmission header (TH) accompanying any transmissions exchanged during the session.

session activation request. In SNA, a request that activates a session between two network accessible units (NAUs) and specifies session parameters that control various protocols during session activity; for example, BIND and ACTPU. Contrast with *session deactivation request*.

session connector. A session-layer component in an APPN network node or in a subarea node boundary or gateway function that connects two stages of a session. Session connectors swap addresses from one address space to another for session-level intermediate routing, segment session message units as needed, and (except for gateway function session connectors) adaptively pace the session traffic in each direction. See also *half-session*.

session control (SC). In SNA, either of the following:

One of the components of transmission control. Session control is used to purge data flowing in a session after an unrecoverable error occurs, to resynchronize the data flow after such an error, and to perform cryptographic verification.

A request unit (RU) category used for requests and responses exchanged between the session control components of a session and for session activation and deactivation requests and responses.

session data. Data about a session, collected by the NetView program, that consists of session awareness data, session trace data, and session response time data.

session deactivation request. In SNA, a request that deactivates a session between two network accessible units (NAUs); for example, UNBIND and DACTPU. Synonymous with *generic unbind*. Contrast with *session activation request*.

session-level LU-LU verification. An LU 6.2 security service that is used to verify the identity of each logical unit when a session is established.

session-level pacing. (1) A flow control technique that permits a receiving half-session or session connector to control the data transfer rate (the rate at which it receives request units) on the normal flow. It is used to prevent overloading a receiver with unprocessed requests when the sender can generate requests faster than the receiver can process them. (2) See also *adaptive session-level pacing*, *fixed session-level pacing*, and *virtual route (VR) pacing*.

session limit. (1) The maximum number of concurrently active LU-LU sessions that a particular logical unit (LU) can support. (2) In NCP, the maximum number of concurrent line-scheduling sessions on a non-SDLC, multipoint line.

session parameters. In SNA, the parameters that specify or constrain the protocols (such as bracket protocol and pacing) for a session between two network accessible units. See also *logon mode*.

session services (SS). One of the types of network services in the control point (CP) and in the logical unit (LU). These services provide facilities for an LU or a network operator to request that a control point (an ENCP, NNCP, or SSCP) assist with initiating or terminating sessions between logical units. Assistance with session termination is needed only by SSCP-dependent LUs. See *configuration services*, *maintenance services*, and *management services*.

shared. Pertaining to the availability of a resource for more than one use at the same time.

shell. (1) A software interface between a user and the operating system of a computer. Shell programs interpret commands and user interactions on devices such as keyboards, pointing devices, and touch-sensitive screens and communicate them to the operating system. Shells simplify user interactions by eliminating the user's concern with operating system requirements. A computer may have several layers of shells for various levels of user interaction. (2) In the AIX operating system, a command interpreter that acts as an interface between the user and the operating system. A shell can contain another shell nested inside it; the outer shell is the parent shell, and the inner shell is the child.

shift-in character (SI). A code extension character used to terminate a sequence that has been introduced by the shift-out character to make effective the graphic characters of the standard character set. (I) (A)

shift-out character (SO). A code extension character that substitutes for the graphic characters of the standard character set an alternative set of graphic characters upon which an agreement has been arrived at or that has been designated using code extension procedures. (I) (A)

shutdown. The process of ending operation of a system or a subsystem, following a defined procedure.

SI. The shift-in character. (I) (A)

simplex. Pertaining to printing on only one side of a sheet of paper. Contrast with *duplex* and *tumble*.

single-byte character set (SBCS). A character set in which each character is represented by a one-byte code. Contrast with *double-byte character set (DBCS)*.

SLU. Secondary logical unit.

SNA. Systems Network Architecture.

SNA network. The part of a user-application network that conforms to the formats and protocols of Systems Network Architecture. It enables reliable transfer of data among users and provides protocols for controlling the resources of various network configurations. The SNA network consists of network accessible units (NAUs), boundary function, gateway function, and intermediate session routing function components; and the transport network.

SNAP. Subnetwork Access Protocol.

SO. The shift-out character. (I) (A)

SP. Service point.

SSCP. System services control point.

SSCP-dependent LU. An LU that requires assistance from a system services control point (SSCP) in order to initiate an LU-LU session. It requires an SSCP-LU session.

SSCP-LU session. In SNA, a session between a system services control point (SSCP) and a logical unit (LU). The session enables the LU to request the SSCP to help initiate LU-LU sessions.

SSCP-PU session. In SNA, a session between a system services control point (SSCP) and a physical unit (PU); SSCP-PU sessions allow SSCPs to send requests to and receive status information from individual nodes in order to control the network configuration.

stack. Synonym for *pushdown list*.

stage. A program that processes messages in a NetView pipeline. Stages send messages to each other serially.

statement. A language syntactic unit consisting of an operator, or other statement identifier, followed by one or more operands. See *definition statement*.

static. (1) In programming languages, pertaining to properties that can be established before execution of a program; for example, the length of a fixed length variable is static. (I) (2) Pertaining to an operation that occurs at a predetermined or fixed time. (3) Contrast with *dynamic*.

status. The condition or state of hardware or software, usually represented by a status code.

storage. (1) A functional unit into which data can be placed, in which they can be retained and from which they can be retrieved. (T) (2) The action of placing data into a storage device. (I) (A) (3) A storage device. (A)

stream. (1) To send data from one device to another. (2) See *data stream*.

STSN. Set and test sequence numbers.

stub. (1) A program module that transfers remote procedure calls (RPCs) and responses between a client and a server. Stubs perform marshalling, unmarshalling, and data format conversion. Both clients and servers have stubs. The Network Interface Definition Language (NIDL) compiler generates client and server stub code from an interface definition. (2) Hooking functions used as extensions to the protocol to generate protocol requests for the Enhanced X-Windows Toolkit. (3) A small module, link-edited into application code, that locates and transfers control to a larger body of related code.

subarea. A portion of the SNA network consisting of a subarea node, attached peripheral nodes, and associated resources. Within a subarea node, all network accessible units (NAUs), links, and adjacent link stations (in attached peripheral or subarea nodes) that are addressable within the subarea share a common subarea address and have distinct element addresses.

subarea LU. In SNA, a logical unit in a subarea node. Contrast with *peripheral LU*.

subarea network. Interconnected subareas, their directly attached peripheral nodes, and the transmission groups that connect them.

subarea node (SN). A node that uses network addresses for routing and maintains routing tables that reflect the configuration of the network. Subarea nodes can provide gateway function to connect multiple subarea networks, intermediate routing function, and boundary function support for peripheral nodes. Type 4 and type 5 nodes can be subarea nodes.

subarea PU. In SNA, a physical unit in a subarea node. Contrast with *peripheral PU*.

subdirectory. A directory contained within another directory in a file system hierarchy.

Subnetwork Access Protocol (SNAP). In LANs, a 5-byte protocol discriminator that identifies the non-IEEE standard protocol family to which a packet belongs. The SNAP value is used to differentiate between protocols that use $AA as their service access point (SAP) value.

subsystem. A secondary or subordinate system, usually capable of operating independently of, or asynchronously with, a controlling system. (T)

subsystem management. In Communications Manager/2, a group of advanced functions for diagnosis and problem resolution. These functions are typically used by system administrators and application programmers.

supervisor call (SVC). A request that serves as the interface into operating system functions, such as allocating storage. The SVC protects the operating system from inappropriate user entry. All operating system requests must be handled by SVCs.

SVC. (1) Supervisor call. (2) Switched virtual circuit.

switched virtual circuit (SVC). An X.25 circuit that is dynamically established when needed. The X.25 equivalent of a switched line. Contrast with *permanent virtual circuit (PVC)*.

symbol. In NetView for AIX, a picture or an icon on a submap that represents an object (a network resource or an application). Each symbol belongs to a class, represented by the symbol's shape, and to a subclass, represented by the design within the shape. The symbol reflects characteristics of the object it represents, such as its status; it also has characteristics of its own, such as behavior.

sync point. An intermediate or end point during processing of a transaction at which an update or modification to one or more of the transaction's protected resources is logically complete and error free. Synonymous with *synchronization point*.

synchronization point. Synonym for *sync point*.

synchronous. (1) Pertaining to two or more processes that depend upon the occurrence of specific events such as common timing signals. (T) (2) Occurring with a regular or predictable time relationship.

Synchronous Data Link Control (SDLC). (1) A discipline conforming to subsets of the Advanced Data Communication Control Procedures (ADCCP) of the American National Standards Institute (ANSI) and High-level Data Link Control (HDLC) of the International Organization for Standardization, for managing synchronous, code-transparent, serial-by-bit information transfer over a link connection. Transmission exchanges may be duplex or half-duplex over switched or nonswitched links. The configuration of the link connection may be point-to-point, multipoint, or loop. (I) (2) Contrast with *binary synchronous communication (BSC)*.

synchronous operation. In VTAM, a communication, or other operation in which VTAM, after receiving the request for the operation, does not return control to the program until the operation is completed. Contrast with *asynchronous operation*.

synchronous request. In VTAM, a request for a synchronous operation. Contrast with *asynchronous request*.

SYNTAX. In the Simple Network Management Protocol (SNMP), a clause in the MIB module that defines the abstract data structure that corresponds to a managed object.

system. In data processing, a collection of people, machines, and methods organized to accomplish a set of specific functions. (I) (A)

system semaphore. In the OS/2 operating system, a signal mechanism providing control between threads of multiple processes that can be used by processes that do not share memory. Contrast with *RAM semaphore*.

system services control point (SSCP). A component within a subarea network for managing the configuration, coordinating network operator and problem determination requests, and providing directory services and other session services for users of the network. Multiple SSCPs, cooperating as peers with one another, can divide the network into domains of control, with each SSCP having a hierarchical control relationship to the physical units and logical units within its own domain.

system services control point (SSCP) domain. The system services control point, the physical units (PUs), the logical units (LUs), the links, the link stations, and all the resources that the SSCP has the ability to control by means of activation and deactivation requests.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks. The layered structure of SNA allows the ultimate origins and destinations of information, that is, the users, to be independent of and unaffected by the specific SNA network services and facilities used for information exchange.

T

table. A repository for data that NetDA/2 uses to design a network. Each table contains information related to the network.

task. In a multiprogramming or multiprocessing environment, one or more sequences of instructions treated by a control program as an element of work to be accomplished by a computer. (I) (A)

task panel. Online display from which you communicate with the program in order to accomplish the program's function, either by selecting an option provided on the panel or by entering an explicit command. See also *help panel*.

Task-Related User Exit (TRUE). A CICS module used for invoking resource managers that are outside of CICS. A TRUE is one of the components of CICS sockets.

TC. Transmission control.

temporary error. A resource failure that can be resolved by error recovery programs. Synonymous with *performance error*. Contrast with *permanent error*.

TERMINATE. In SNA, a request unit that is sent by a logical unit (LU) to its system services control point (SSCP) to cause the SSCP to start a procedure to end one or more designated LU-LU sessions.

TH. Transmission header.

thread. In the OS/2 operating system, the smallest unit of operation to be performed within a process.

timeout. (1) An event that occurs at the end of a predetermined period of time that began at the occurrence of another specified event. (I) (2) A time

interval allotted for certain operations to occur; for example, response to polling or addressing before system operation is interrupted and must be restarted.

TLPB. Transport-layer protocol boundary.

token. (1) In a local area network, the symbol of authority passed successively from one data station to another to indicate the station temporarily in control of the transmission medium. Each data station has an opportunity to acquire and use the token to control the medium. A token is a particular message or bit pattern that signifies permission to transmit. (T) (2) In LANs, a sequence of bits passed from one device to another along the transmission medium. When the token has data appended to it, it becomes a frame.

token ring. (1) According to IEEE 802.5, network technology that controls media access by passing a token (special packet or frame) between media-attached stations. (2) A FDDI or IEEE 802.5 network with a ring topology that passes tokens from one attaching ring station (node) to another. (3) See also *local area network (LAN)*.

token-ring network. (1) A ring network that allows unidirectional data transmission between data stations, by a token passing procedure, such that the transmitted data return to the transmitting station. (T) (2) A network that uses a ring topology, in which tokens are passed in a circuit from node to node. A node that is ready to send can capture the token and insert data for transmission.

TP. (1) Transaction program. (2) Transmission priority.

transaction. In IMS/VS, a specific set of input data that triggers execution of a specific process or job. A transaction is a message destined for an application program.

transaction program (TP). (1) A program that processes transactions in an SNA network. There are two kinds of transaction programs: application transaction programs and service transaction programs. See also *conversation*. (2) In VTAM, a program that performs services related to the processing of a transaction. One or more transaction programs may operate within a VTAM application program that is using the VTAM application program interface (API). In that situation, the transaction program would request services from the application program, using protocols defined by that application program. The application program, in turn, could request services from VTAM by issuing the APPCCMD macroinstruction.

translation table. A table used to replace one or more characters with alternative characters; for example, to translate characters representing a virtual address to those representing a real address, characters representing an event to those representing a procedure call, characters of a national character set to those of another national language, or characters representing a relocated address to those representing an absolute address.

transmission control (TC) layer. The layer within a half-session or session connector that synchronizes and paces session-level data traffic, checks session sequence numbers of requests, and enciphers and deciphers end-user data. See also *half-session*.

transmission header (TH). Control information, optionally followed by a basic information unit (BIU) or a BIU segment, that is created and used by path control to route message units and to control their flow within the network. See also *path information unit*.

transport-layer protocol boundary (TLPB). In MPTN architecture, the protocol boundary that provides access in a protocol-independent fashion to multiple transport protocols.

trap. In the Simple Network Management Protocol (SNMP), a message sent by a managed node (agent function) to a management station to report an exception condition.

TRUE. Task-Related User Exit.

tumble. Pertaining to printing on both sides of a sheet of paper such that the bottom of the printed image on one side of the paper is juxtaposed against the ***top*** of the printed image on the other side of the paper. Contrast with *duplex* and *simplex*.

tutorial. Information presented in a teaching format.

U

UNBIND. In SNA, a request to deactivate a session between two logical units (LUs). See also *session deactivation request*. Contrast with *BIND*.

uninterpreted name. In SNA, a character string that a system services control point (SSCP) can convert into the network name of a logical unit (LU). Typically, an uninterpreted name is used in a logon or Initiate request from a secondary logical unit (SLU) to identify the primary logical unit (PLU) with which the session is requested.

UP. Unnumbered poll.

UPM. User Profile Management.

user. (1) Any person or any thing that may issue or receive commands and messages to or from the

information processing system. (T) (2) Anyone who requires the services of a computing system.

user exit. (1) A point in an IBM-supplied program at which a user exit routine may be given control. (2) A programming service provided by an IBM software product that may be requested during the execution of an application program for the service of transferring control back to the application program upon the later occurrence of a user-specified event.

user profile. In computer security, a description of a user that includes such information as user ID, user name, password, access authority, and other attributes obtained at logon.

User Profile Management (UPM). In the OS/2 operating system, a function that provides user ID validation and user and group management facilities. Communications Manager/2 uses UPM. Each installation of UPM is local to the particular workstation on which it is installed and validates user access to controlled data or programs that reside on that particular workstation. UPM also provides the logon-logoff mechanism for identifying and authenticating system users.

UTC. Coordinated universal time.

V

validation. The checking of data for correctness or for compliance with applicable standards, rules, and conventions. (A)

value. (1) A specific occurrence of an attribute; for example, "blue" for the attribute "color." (T) (2) A quantity assigned to a constant, a variable, a parameter, or a symbol.

variable. (1) In programming languages, a language object that may take different values, one at a time. The values of a variable are usually restricted to a certain data type. (I) (2) A quantity that can assume any of a given set of values. (A) (3) A name used to represent a data item whose value can be changed while the program is running. (4) In the Simple Network Management Protocol (SNMP), a match of an object instance name with an associated value. (5) In the NetView command list language, a character string beginning with "&" that is coded in a command list and is assigned a value during execution of the command list.

verb. See *LU 6.2 verb*.

version. A separately licensed program that usually has significant new code or new function.

virtual circuit. (1) In packet switching, the facilities provided by a network that give the appearance to the user of an actual connection. (T) See also *data circuit*. Contrast with *physical circuit*. (2) A logical connection established between two DTEs.

virtual route (VR). (1) In SNA, either (a) a logical connection between two subarea nodes that is physically realized as a particular explicit route or (b) a logical connection that is contained wholly within a subarea node for intranode sessions. A virtual route between distinct subarea nodes imposes a transmission priority on the underlying explicit route, provides flow control through virtual route pacing, and provides data integrity through sequence numbering of path information units (PIUs). (2) Contrast with *explicit route (ER)*. See also *path* and *route extension (REX)*.

virtual route (VR) pacing. In SNA, a flow control technique used by the virtual route control component of path control at each end of a virtual route to control the rate at which path information units (PIUs) flow over the virtual route. VR pacing can be adjusted according to traffic congestion in any of the nodes along the route. See also *session-level pacing*.

Virtual Telecommunications Access Method (VTAM). An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

VP. Variable pageable.

VR. Virtual route.

VTAM. (1) Virtual Telecommunications Access Method. (2) Synonymous with *ACF/VTAM*.

VTAM operator command. A command used to monitor or control a VTAM domain. See also *definition statement*.

W

WAN. Wide area network.

wide area network (WAN). (1) A network that provides communication services to a geographic area larger than that served by a local area network or a metropolitan area network, and that may use or provide public communication facilities. (T) (2) A data communication network designed to serve an area of hundreds or thousands of miles; for example, public and private packet-switching networks, and national telephone networks. (3) Contrast with *local area network (LAN)* and *metropolitan area network (MAN)*.

window. (1) A portion of a display surface in which display images pertaining to a particular application can

be presented. Different applications can be displayed simultaneously in different windows. (A) (2) An area with visible boundaries that presents a view of an object or with which a user conducts a dialog with a computer system. (3) In data communication, the number of data packets a data terminal equipment (DTE) or data circuit-terminating equipment (DCE) can send across a logical channel before waiting for authorization to send another data packet. The window is the main mechanism of pacing, or flow control, of packets. (4) See *pacing window*.

WinSock application programming interface (API). A socket-style transport interface developed for the Windows family of operating systems.

working directory. The directory to which an application program will write data if no path is specified.

workstation. (1) A functional unit at which a user works. A workstation often has some processing capability. (T) (2) One or more programmable or nonprogrammable devices that allow a user to do work. (3) A terminal or microcomputer, usually one that is connected to a mainframe or to a network, at which a user can perform applications.

X

X.25. (1) An International Telegraph and Telephone Consultative Committee (CCITT) recommendation for the interface between data terminal equipment and packet-switched data networks. (2) See also *packet switching*.

Z

Z time. Abbreviation for *Zulu time*. Synonym for *coordinated universal time (UTC)*.

Zulu time (Z). Synonym for *coordinated universal time (UTC)*.

Index

A

ACSSVC 147
ACTLU 175
ALLOCATE 62
AP_ALLOCATION_ERROR 229
AP_ALLOCATION_FAILURE_NO_RETRY 229
AP_ALLOCATION_FAILURE_RETRY 229
AP_CONV_FAILURE_NO_RETRY 230
AP_CONV_FAILURE_RETRY 230
AP_CONVERSATION_TYPE_MISMATCH 230
AP_CONVERSATION_TYPE_MIXED 230
AP_DEALLOC_ABEND 230
AP_DEALLOC_ABEND_PROGRAM 230
AP_DEALLOC_ABEND_SVC 230
AP_DEALLOC_ABEND_TIMER 230
AP_DEALLOC_NORMAL 231
AP_PIP_NOT_ALLOWED 229
AP_PIP_NOT_SPECIFIED_CORRECTLY 229
AP_PROG_ERROR_NO_TRUNC 231
AP_PROG_ERROR_PURGING 231
AP_PROG_ERROR_TRUNC 231
AP_SECURITY_NOT_VALID 229
AP_SVC_ERROR_NO_TRUNC 231
AP_SVC_ERROR_PURGING 231
AP_SVC_ERROR_TRUNC 231
AP_SYNC_LEVEL_NOT_SUPPORTED 230
AP_TP_BUSY 231
AP_TP_NAME_NOT_RECOGNIZED 229
AP_TRANS_PGM_NOT_AVAIL_NO_RTRY 229
AP_TRANS_PGM_NOT_AVAIL_RETRY 229
AP_UNEXPECTED_SYSTEM_ERROR 231
APPC API support
  default local LU pool 26
  option sets supported 23
  queue-level nonblocking 24
  verbs supported 43
APPC entry points
  APPC() 27
  GetAppcConfig( ) 39
  GetAppcReturnCode( ) 40
  WinAPPCCancelAsyncRequest( ) 32
  WinAPPCCancelBlockingCall( ) 33
  WinAPPCCleanup( ) 34
  WinAPPCIsBlocking( ) 35
  WinAPPCSetBlockingHook( ) 37
  WinAPPCStartup( ) 36
  WinAPPCUnhookBlockingHook( ) 38
  WinAsyncAPPC( ) 28
  WinAsyncAPPCEx( ) 30
APPC() 27
asynchronous verb completion 174
attach manager
  description 11
  identifying transition program name 12
  matching incoming allocation requests
    nonqueued programs 17
    queued programs 17
  starting 15
  starting programs 16

B

basic conversation verb control blocks
  ALLOCATE 62
  CONFIRM 66
  CONFIRMED 68
  DEALLOCATE 70
  FLUSH 73
  GET_ATTRIBUTES 75
  PREPARE_TO_RECEIVE 78
  RECEIVE_AND_POST 80
  RECEIVE_AND_WAIT 84
  RECEIVE_IMMEDIATE 87
  REQUEST_TO_SEND 90
  SEND_CONVERSATION 92
  SEND_DATA 95
  SEND_ERROR 98
  TEST_RTS 101
  TEST_RTS_AND_POST 103
BIND
  negotiating parameters 176

C

CANCEL 179
common data structure 195
common return codes
  AP_ALLOCATION_ERROR 229
  AP_ALLOCATION_FAILURE_NO_RETRY 229
  AP_ALLOCATION_FAILURE_RETRY 229
  AP_CONV_FAILURE_NO_RETRY 230
  AP_CONV_FAILURE_RETRY 230
  AP_CONVERSATION_TYPE_MISMATCH 230
  AP_CONVERSATION_TYPE_MIXED 230
  AP_DEALLOC_ABEND 230
  AP_DEALLOC_ABEND_PROG 230
  AP_DEALLOC_ABEND_SVC 230
  AP_DEALLOC_ABEND_TIMER 230
  AP_DEALLOC_NORMAL 231
  AP_PIP_NOT_ALLOWED 229
  AP_PIP_NOT_SPECIFIED_CORRECTLY 229
  AP_PROG_ERROR_NO_TRUNC 231

© Copyright IBM Corp. 1994, 1996

common return codes *(continued)*
AP_PROG_ERROR_PURGING 231
AP_PROG_ERROR_TRUNC 231
AP_SECURITY_NOT_VALID 229
AP_SVC_ERROR_NO_TRUNC 231
AP_SVC_ERROR_PURGING 231
AP_SVC_ERROR_TRUNC 231
AP_SYNC_LEVEL_NOT_SUPPORTED 230
AP_TP_BUSY 231
AP_TP_NAME_NOT_RECOGNIZED 229
AP_TRANS_PGM_NOT_AVAIL_NO_RTRY 229
AP_TRANS_PGM_NOT_AVAIL_RETRY 229
AP_UNEXPECTED_SYSTEM_ERROR 231
common services entry points
ACSSVC 147
GetCsvReturnCode 151
TrnsDt 152
WinCSV 147
WinCSVAsyncCSV 149
WinCSVCleanup 148
WinCSVStartup 150
common services verbs
CONVERT 160
GET_CP_CONVERT_TABLE 157
configuration information 173
CONFIRM 66
CONFIRMED 68
conversation 7
defining attributes 20, 21
security for incoming allocation requests 21
security for outgoing allocation requests 22
conversation state transitions
non-OK return codes 235
Pending Post state 235
Reset state 235
state change after RECEIVE verbs
primary_rc parameters 235
what_rcvd parameters 235
use of AP_ERROR 235
CONVERT 160
courtesy acknowledgment 178

D

DEALLOCATE 70
default local LU pool 26

F

FLUSH 73

G

GET_ATTRIBUTES 75
GET_CP_CONVERT_TABLE 157
GET_TP_PROPERTIES 51
GET_TYPE 54
GetAppcConfig( ) 39
GetAppcReturnCode( ) 40

H

highlighting conventions xii

I

INITSELF 175
introduction 5

L

LU
dependent 6
description of 6
independent 7
types 6
LU pools 173
LU-SSCP session
establishing 175
LUA
application programs 165
compatibility 176
connection capabilities 165
issuing RUI verbs
LUs, local and partner 166
restarting and resynchronizing 170
RUI sessions 181
sample LUA communication sequence 175
SNA layers 166
understanding 165
using SNA sessions
disconnecting 170
prerequisites 168
starting 168
stopping 169
transferring data on an LU-LU session 169
verbs 165, 174
LUA verbs
asynchronous verb completion 174
summary 174

M

mapped conversation verb control blocks
MC_ALLOCATE 105
MC_CONFIRM 109
MC_CONFIRMED 111
MC_DEALLOCATE 113
MC_FLUSH 115
MC_GET_ATTRIBUTES 117
MC_PREPARE_TO_RECEIVE 120
MC_RECEIVE_AND_POST 122

mapped conversation verb control blocks *(continued)*
MC_RECEIVE_AND_WAIT 125
MC_RECEIVE_IMMEDIATE 128
MC_REQUEST_TO_SEND 131
MC_SEND_CONVERSATION 132
MC_SEND_DATA 136
MC_SEND_ERROR 139
MC_TEST_RTS 141
MC_TEST_RTS_AND_POST 143
MC_ALLOCATE 105
MC_CONFIRM 109
MC_CONFIRMED 111
MC_DEALLOCATE 113
MC_FLUSH 115
MC_GET_ATTRIBUTES 117
MC_PREPARE_TO_RECEIVE 120
MC_RECEIVE_AND_POST 122
MC_RECEIVE_AND_WAIT 125
MC_RECEIVE_IMMEDIATE 128
MC_REQUEST_TO_SEND 131
MC_SEND_CONVERSATION 132
MC_SEND_DATA 136
MC_SEND_ERROR 139
MC_TEST_RTS 141
MC_TEST_RTS_AND_POST 143
message numbers 170

N

NOTIFY 176

O

options sets supported by Personal Communications 23

P

pacing 178
protocols
bracket 172
data-chaining 173
half-duplex contention flip/flop 172
pacing 171
purging 178

Q

queue-level nonblocking support
explanation of 24
three types of queues 24

R

reserved parameters 195
RUI verbs
common verb header 196
RUI verbs *(continued)*
LUA verb control format 195
RUI_BID 201
error return codes 203
successful execution 202
RUI_BID data structure 200
RUI_INIT 206
error return codes 207
successful execution 207
RUI_INIT_STATUS 213
RUI_PURGE 210
error return codes 211
successful execution 210
RUI_READ 214
error return codes 216
successful execution 215
truncated data 215
RUI_TERM 220
successful execution 220
RUI_WRITE 223
error return codes 225
successful execution 224

S

sample LUA communication sequence 175
SDT 176
segmentation 178
session 7
SNA
communication support 3
LU type 6.2 support 4
SNA messages
relationship to LUA verbs 175
SNA sense codes 177
specific data structure 195

T

transaction program
APPC 5
compared to an application 16
CPI Communications 6
default local LU pool 26
definitions 20
description of 5
queue-level nonblocking 24
supported option sets 23
writing 23
TrnsDt 152
type independent verb control blocks
GET_TP_PROPERTIES 51
GETE_TYPE 54
RECEIVE_ALLOCATE 56
TP_ENDED 58
TP_STARTED 60

U

UNBIND 176

V

verb control block
  structure 195
verb control blocks
  common fields 43
verb signals
  basic conversation verb control blocks
    ALLOCATE 62
    CONFIRM 66
    CONFIRMED 68
    DEALLOCATE 70
    FLUSH 73
    GET_ATTRIBUTES 75
    PREPARE_TO_RECEIVE 78
    RECEIVE_AND_POST 80
    RECEIVE_AND_WAIT 84
    RECEIVE_IMMEDIATE 87
    REQUEST_TO_SEND 90
    SEND_CONVERSATION 92
    SEND_DATA 95
    SEND_ERROR 98
    TEST_RTS 101
    TEST_RTS_AND_POST 103
  mapped conversation verb control blocks
    MC_ALLOCATE 105
    MC_CONFIRM 109
    MC_CONFIRMED 111
    MC_DEALLOCATE 113
    MC_FLUSH 115
    MC_GET_ATTRIBUTES 117
    MC_PREPARE_TO_RECEIVE 120
    MC_RECEIVE_AND_POST 122
    MC_RECEIVE_AND_WAIT 125
    MC_RECEIVE_IMMEDIATE 128
    MC_REQUEST_TO_SEND 131
    MC_SEND_CONVERSATION 132
    MC_SEND_DATA 136
    MC_SEND_ERROR 139
    MC_TEST_RTS 141
    MC_TEST_RTS_AND_POST 143
  verb control blocks
    common fields 43
verbs supported at the APPC API
  basic conversation verbs 44
  mapped conversation verbs 44
  type independent verbs 44

W

WinAPPCCancelAsynRequest( ) 32
WinAPPCCancelBlockingCall( ) 33
WinAPPCCleanup( ) 34
WinAPPCIsBlocking( ) 35
WinAPPCSetBlockingHook( ) 37
WinAPPCStartup( ) 36
WinAPPCUnhookBlockingHook( ) 38
WinAsyncAPPC( ) 28
WinAsyncAPPCEx( ) 30
WinAsyncCSV 149
WinCSV 147
WinCSVCleanup 148
WinCSVStartup 150
writing LUA APPC program
  calling Dynamic Link Libraries 183
  DLL interface, using 193
  prodecure entry points 183

Communicating Your Comments to IBM

Personal Communications
AS/400 and 3270 for Windows** 95
Client/Server
Communications Programming
Version 4.1

Publication No. SC31-8208-00

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

If you prefer to send comments by mail, use the RCF at the back of this book.

If you prefer to send comments by FAX, use this number:

United States and Canada: **1-800-227-5088**

If you prefer to send comments electronically, use this network ID:

- IBM Mail Exchange: **USIB2HPD at IBMMAIL**
- IBMLink: **CIBMORCF at RALVM13**
- Internet: **USIB2HPD@VNET.IBM.COM**

Make sure to include the following in your note:

Title and publication number of this book
Page number or topic to which your comment applies.

Help us help you!

Personal Communications
AS/400 and 3270 for Windows 95**
Client/Server
Communications Programming
Version 4.1

Publication No. SC31-8208-00

We hope you find this publication useful, readable and technically accurate, but only you can tell us! Your comments and suggestions will help us improve our technical publications. Please take a few minutes to let us know what you think by completing this form.

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

How satisfied are you that the information in this book is:	Satisfied	Dissatisfied
Accurate		
Complete		
Easy to find		
Easy to understand		
Well organized		
Applicable to your task		

Specific Comments or Problems:

Please tell us how we can improve this book:

Thank you for your response. When you send information to IBM, you grant IBM the right to use or distribute the information without incurring any obligation to you. You of course retain the right to use the information in any way you choose.

Your Internet Address: _____

Name

Address

Company or Organization

Phone No.

Help us help you!
SC31-8208-00

Cut or Fold
Along Line

IBM®

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

Information Development
Department CGMD
International Business Machines Corporation
PO BOX 12195
RESEARCH TRIANGLE PARK NC 27709-9990

Fold and Tape **Please do not staple** Fold and Tape

SC31-8208-00

Cut or Fold
Along Line

IBM®

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

SC31-82 8-

Labels

Labels	Page
dmpl0007	3
dmpl0002	7
dmpl0003	8
dmpl0004	9
DMPL0010	11

Artwork Definitions

id	File	Page	References
COV1	DPJL0SET	i	i
DPJL0C2	DPJL0SET	i	282

LERS Definitions

id	File	Page	References
DPJLER1	DPJL0SET	i	13, 27, 27, 45, 47, 51, 53, 62, 65, 105, 108, 147, 148, 152, 156, 183, 184, 200

Table Definitions

id	File	Page	References
T2C2HD	DPJL0AAM	19	19
T2C2FT	DPJL0AAM	19	
TABLE1	DPJL0CNV	175	175
TOPROW1	DPJL0UND	181	182

Figures

id	File	Page	References
C801	DPJL0NEW	3	1
			3
SESS	DPJL0INT	7	2
			7
C2A1	DPJL0INT	8	3
			8
C2A2	DPJL0INT	9	4
			8
TWOBOXE	DPJL0AAM	11	5
			11
KEYEX1	DPJL0GLO	246	6
			246
KEYEX2	DPJL0GLO		

		246	7 246
KEYEX3	DPJL0GLO	246	8 246

Headings

id	File	Page	References
NOTICES	DPJL0NOT	ix	Notices ii
TRADE	DPJL0NOT	x	Trademarks
THIS	DPJL0PRE	xi	About This Book
WHO	DPJL0PRE	xi	Who Should Read This Book
HOW	DPJL0PRE	xi	How to Use This Book
HIGH	DPJL0PRE	xii	Conventions Used in This Book
REL	DPJL0PRE	xiii	Related Publications
PART1	DPJL0NEW	1	Part 1, APPC API xi
OVERVEW	DPJL0NEW	3	Chapter 1, Introducing APPC xi
INTSNA	DPJL0NEW	3	SNA Communications Support
SNAFUNC	DPJL0NEW	4	SNA LU Type 6.2 Support
FNDCNCP	DPJL0INT	5	Chapter 2, Fundamental APPC Concepts xi
FUN01	DPJL0INT	5	What Is a Transaction Program?
FUN02	DPJL0INT	5	APPC Transaction Programs
SAACPI	DPJL0INT	6	CPI Communications Transaction Programs 5
FUN03	DPJL0INT	6	What Is a Logical Unit?
FUN04	DPJL0INT	6	LU Types
FUN05	DPJL0INT	6	Dependent and Independent LUs
SESSION	DPJL0INT	7	What Is a Session?
FUN06	DPJL0INT	7	What Is a Conversation?
USEAM	DPJL0AAM	11	Chapter 3, Using the Attach Manager xi, 8
AAM02	DPJL0AAM	12	Identifying the Transaction Program Name on Both Machines
AAM03	DPJL0AAM	15	Starting the Attach Manager
AAM04	DPJL0AAM	16	Starting Programs with the Attach Manager
DIFF	DPJL0AAM	16	Differentiating between an Application and a Transaction Program
AAM05	DPJL0AAM	17	Matching Incoming Allocation Requests with RECEIVE_ALLOCATE Verbs
AAM06	DPJL0AAM	20	Transaction Program Definitions
AAM07	DPJL0AAM	20	Defining Conversation Attributes
AAM08	DPJL0AAM	21	Conversation Security for an Incoming Allocation Request
AAM09	DPJL0AAM	22	Conversation Security for an Outgoing Allocation Request
TPMPLMT	DPJL0APP		

		23 Chapter 4, Implementing APPC Transaction Programs xi, 5
ITP01	DPJL0APP	23 Writing Transaction Programs
OPSETS	DPJL0APP	23 Option Sets Supported
QLEV	DPJL0APP	24 Queue-level Nonblocking 90, 131
DEFLO	DPJL0APP	26 Default Local LU Pool
OVR	DPJL0OVR	27 Chapter 5, Entry Points xi
VAPPC	DPJL0OVR	27 APPC 31
WNASY	DPJL0OVR	28 WinAsyncAPPC() 31, 32
WNAPEX	DPJL0OVR	30 WinAsyncAPPCEx() 27, 29
WNCARQ	DPJL0OVR	32 WinAPPCCancelAsyncRequest() 29, 31
WNCBLC	DPJL0OVR	33 WinAPPCCancelBlockingCall() 35, 37
WNCLUP	DPJL0OVR	34 WinAPPCCleanup() 36
WNBLOC	DPJL0OVR	35 WinAPPCIsBlocking() 37
WNSTAR	DPJL0OVR	36 WinAPPCStartup() 34
WNBLHK	DPJL0OVR	37 WinAPPCSetBlockingHook() 35, 38
WNUNBH	DPJL0OVR	38 WinAPPCUnhookBlockingHook() 35, 37
GETCG	DPJL0OVR	39 GetAppcConfig()
GETRT	DPJL0OVR	40 GetAppcReturnCode()
SIG	DPJL0SIG	43 Chapter 6, APPC Verbs xii
VBBL	DPJL0SIG	43 Verb Control Blocks
COMFD	DPJL0SIG	43 Common Fields
APPSUP	DPJL0SIG	43 APPC API Support
VBSUP	DPJL0SIG	43 Verbs Supported
INDVB	DPJL0SIG	45 Indication Verb Control Blocks
TPIN	DPJL0SIG	51 Type Independent Verb Control Blocks
GETTP	DPJL0SIG	51 GET_TP_PROPERTIES
GETTYP	DPJL0SIG	54 GET_TYPE
RECALL	DPJL0SIG	56 RECEIVE_ALLOCATE
TPEND	DPJL0SIG	58 TP_ENDED
TPSTA	DPJL0SIG	60 TP_STARTED 26
BASCB	DPJL0SIG	62 Basic Conversation Verb Control Blocks
ALLC	DPJL0SIG	62 ALLOCATE
CONF	DPJL0SIG	66 CONFIRM
CFED	DPJL0SIG	

DELL	DPJL0SIG
FLUS	DPJL0SIG
GETAT	DPJL0SIG
PRREC	DPJL0SIG
RCPST	DPJL0SIG
RCWT	DPJL0SIG
RCIMM	DPJL0SIG
RQSND	DPJL0SIG
SDCVN	DPJL0SIG
SDDAT	DPJL0SIG
SDERR	DPJL0SIG
TRTS	DPJL0SIG
TRTP	DPJL0SIG
MAPP	DPJL0SIG
MCALL	DPJL0SIG
MCCFM	DPJL0SIG
MCCCFD	DPJL0SIG
MCDEA	DPJL0SIG
MCFLU	DPJL0SIG
MCGET	DPJL0SIG
MCPRE	DPJL0SIG
MCREC	DPJL0SIG
REWT	DPJL0SIG
MCRIM	DPJL0SIG
MCRSD	DPJL0SIG
MCSDT	DPJL0SIG
MCSERR	DPJL0SIG
MCTTS	DPJL0SIG
MCRP	DPJL0SIG
PART2	DPJL0EXT
COMSV	DPJL0EXT
ENTRYP	DPJL0EXT
ACSSVC	DPJL0EXT
WNCSV	DPJL0EXT
WNCL	DPJL0EXT
WNACSV	DPJL0EXT
WNSTA	DPJL0EXT
GETRTX	DPJL0EXT
GTCVT	DPJL0EXT

68 CONFIRMED

70 DEALLOCATE

73 FLUSH

75 GET_ATTRIBUTES

78 PREPARE_TO_RECEIVE

80 RECEIVE_AND_POST

84 RECEIVE_AND_WAIT

87 RECEIVE_IMMEDIATE

90 REQUEST_TO_SEND

92 SEND_CONVERSATION

95 SEND_DATA

98 SEND_ERROR

101 TEST_RTS

103 TEST_RTS_AND_POST

105 Mapped Conversation Verb Control Blocks

105 MC_ALLOCATE

109 MC_CONFIRM

111 MC_CONFIRMED

113 MC_DEALLOCATE

115 MC_FLUSH

117 MC_GET_ATTRIBUTES

120 MC_PREPARE_TO_RECEIVE

122 MC_RECEIVE_AND_POST

125 MC_RECEIVE_AND_WAIT

128 MC_RECEIVE_IMMEDIATE

131 MC_REQUEST_TO_SEND

136 MC_SEND_DATA

139 MC_SEND_ERROR

141 MC_TEST_RTS

143 MC_TEST_RTS_AND_POST

145 Part 2, Common Services API
xi

147 Chapter 7, Procedure Entry Points and Common Services Verbs (CSV)
xii, 194

147 Procedure Entry Points

147 ACSSVC

147 WinCSV()

148 WinCSVCleanup()

149 WinAsyncCSV()

150 WinCSVStartup()

151 GetCsvReturnCode()

CONV	DPJL0EXT	157 GET_CP_CONVERT_TABLE
PART3	DPJL0CNV	160 CONVERT
CNV	DPJL0CNV	163 Part 3, LUA API xi, 6
CH2	DPJL0CNV	165 Chapter 8, Introducing the IBM Conventional LU Application xii
INT1	DPJL0CNV	165 Understanding LUA and SNA
INT2	DPJL0CNV	165 Connection Capabilities
INT3	DPJL0CNV	165 LUA Application Programs
INT4	DPJL0CNV	165 LUA Verbs
INT5	DPJL0CNV	166 LUs, Local LUs, and Partner LUs
INT6	DPJL0CNV	166 SNA Layers 204, 218, 227, 227
STOPSLU	DPJL0CNV	167 Using SNA Sessions
INT7	DPJL0CNV	169 Stopping an LU-LU Session by an SLU
INT8	DPJL0CNV	170 Message Numbers
INT9	DPJL0CNV	170 Restarting and Resynchronizing a Session
DACH	DPJL0CNV	171 Using Protocols to Control Requests and Responses
LUPOOL	DPJL0CNV	173 Using the Data-Chaining Protocol 169
UND	DPJL0UND	179 LUA LU Pool (optional) 179
ULU3	DPJL0UND	181 Chapter 9, Understanding LUA xii
ULU6	DPJL0UND	181 RUI Sessions
WRI	DPJL0WRI	181 Issuing RUI Verbs
WLA1	DPJL0WRI	183 Chapter 10, Writing LUA Application Programs xii, 166
WINR	DPJL0WRI	183 Calling the Dynamic Link Libraries
WNRUIC	DPJL0WRI	183 WinRUI 190
WNRUIS	DPJL0WRI	185 WinRUICleanup() 189
RUI	DPJL0WRI	189 WinRUIStartup() 185
WLA2	DPJL0WRI	190 RUI() 184
VP	DPJL0WRI	193 Using the DLL Interface
DEF	DPJL0DEF	193 LUA Verb Postings 193
LUADBFM	DPJL0DEF	195 Chapter 11, RUI Verbs xii, 174, 176
COMMVBD	DPJL0DEF	195 LUA Verb Control Block Format
RUIBIDD	DPJL0DEF	196 Common Verb Header 195
RUIXNIT	DPJL0DEF	200 RUI_BID Data Structure 195
		206 RUI_INIT 196, 213

RUIPURD	DPJL0DEF	210	RUI_PURGE
RUINTXT	DPJL0DEF	213	RUI_INIT_STATUS 206
RETRN1	DPJL0DEF	215	Returned Parameters 215
SUXCES1	DPJL0DEF	215	Successful Execution or Truncated Data 218
UNSUCX1	DPJL0DEF	218	Unsuccessful Execution 215
INTERD1	DPJL0DEF	219	Interaction with Other Verbs 217
APPNAXA	DPJL0RET	229	Appendix A, APPC Common Return Codes xii, 53, 55, 57, 59, 61, 65, 67, 69, 72, 74, 77, 79, 82, 86, 89, 91, 94, 96, 99, 102, 104, 108, 110, 112, 114, 116, 119, 121, 124, 127, 130, 132, 135, 137, 140, 142, 144
APPNAXB	DPJL0APB	233	Appendix B, APPC Conversation State Transitions xii
GLO	DPJL0GLO	237	Glossary

Index Entries

id	File	Page	References
ACSSVC	DPJL0INX	x	(1) ACSSVC 147
ALL01	DPJL0INX	x	(1) ALLOCATE 62
BINDX	DPJL0INX	x	(1) BIND x
NEGPAR	DPJL0INX	x	(1) BIND (2) negotiating parameters 176
BCV01	DPJL0INX	x	(1) basic conversation verb control blocks x, x, x, x, x, x, x, x, x, x, x, x, x, x, x
BCV011	DPJL0INX	x	(1) basic conversation verb control blocks (2) ALLOCATE 62
BCV012	DPJL0INX	x	(1) basic conversation verb control blocks (2) CONFIRM 66
BCV013	DPJL0INX	x	(1) basic conversation verb control blocks (2) CONFIRMED 68
BCV014	DPJL0INX	x	(1) basic conversation verb control blocks (2) DEALLOCATE 70
BCV015	DPJL0INX	x	(1) basic conversation verb control blocks (2) FLUSH 73
BCV016	DPJL0INX	x	(1) basic conversation verb control blocks (2) GET_ATTRIBUTES 75
BCV017	DPJL0INX	x	(1) basic conversation verb control blocks (2) PREPARE_TO_RECEIVE 78
BCV018	DPJL0INX	x	(1) basic conversation verb control blocks (2) RECEIVE_AND_POST

80

BCV019 DPJL0INX
x (1) basic conversation verb control blocks
(2) RECEIVE_AND_WAIT
84

BCV01A DPJL0INX
x (1) basic conversation verb control blocks
(2) RECEIVE_IMMEDIATE
87

BCV01B DPJL0INX
x (1) basic conversation verb control blocks
(2) REQUEST_TO_SEND
90

BCV01C DPJL0INX
x (1) basic conversation verb control blocks
(2) SEND_CONVERSATION
92

BCV01D DPJL0INX
x (1) basic conversation verb control blocks
(2) SEND_DATA
95

BCV01E DPJL0INX
x (1) basic conversation verb control blocks
(2) SEND_ERROR
98

BCV01F DPJL0INX
x (1) basic conversation verb control blocks
(2) TEST_RTS
101

BCV01G DPJL0INX
x (1) basic conversation verb control blocks
(2) TEST_RTS_AND_POST
103

CRC01 DPJL0INX
x (1) common return codes
x, x, x,

CRC01D	DPJL0INX		
		x	(1) common return codes (2) AP_CONV_FAILURE_NO_RETRY 230
CRC01E	DPJL0INX		
		x	(1) common return codes (2) AP_CONV_FAILURE_RETRY 230
CRC01F	DPJL0INX		
		x	(1) common return codes (2) AP_CONVERSATION_TYPE_MIXED 230
CRC01G	DPJL0INX		
		x	(1) common return codes (2) AP_DEALLOC_ABEND 230
CRC01H	DPJL0INX		
		x	(1) common return codes (2) AP_DEALLOC_ABEND_PROG 230
CRC01J	DPJL0INX		
		x	(1) common return codes (2) AP_DEALLOC_ABEND_SVC 230
CRC01K	DPJL0INX		
		x	(1) common return codes (2) AP_DEALLOC_ABEND_TIMER 230
CRC01L	DPJL0INX		
		x	(1) common return codes (2) AP_DEALLOC_NORMAL 231
CRC01M	DPJL0INX		
		x	(1) common return codes (2) AP_PROG_ERROR_NO_TRUNC 231
CRC01N	DPJL0INX		
		x	(1) common return codes (2) AP_PROG_ERROR_PURGING 231
CRC01P	DPJL0INX		
		x	(1) common return codes (2) AP_PROG_ERROR_TRUNC 231
CRC01Q	DPJL0INX		
		x	(1) common return codes (2) AP_SVC_ERROR_NO_TRUNC 231
CRC01R	DPJL0INX		
		x	(1) common return codes (2) AP_SVC_ERROR_PURGING 231
CRC01S	DPJL0INX		
		x	(1) common return codes (2) AP_SVC_ERROR_TRUNC 231
CRC01T	DPJL0INX		
		x	(1) common return codes (2) AP_TP_BUSY 231
CRC01U	DPJL0INX		
		x	(1) common return codes (2) AP_UNEXPECTED_SYSTEM_ERROR 231
CON01	DPJL0INX		
		x	(1) CONFIRM 66
CND01	DPJL0INX		
		x	(1) CONFIRMED 68
CAR01	DPJL0INX		
		x	(1) WinAPPCCancelAsynRequest( ) 32
CBC01	DPJL0INX		
		x	(1) WinAPPCCancelBlockingCall( ) 33
CLE01	DPJL0INX		
		x	(1) WinAPPCCleanup( ) 34
CIS01	DPJL0INX		
		x	(1) WinAPPCIsBlocking( )

			35
CST01	DPJL0INX		
		x	(1) conversation state transitions
			x, x, x, x, x
CST011	DPJL0INX		
		x	(1) conversation state transitions
			(2) use of AP_ERROR
			235
CST012	DPJL0INX		
		x	(1) conversation state transitions
			(2) Reset state
			235
CST013	DPJL0INX		
		x	(1) conversation state transitions
			(2) non-OK return codes
			235
CST014	DPJL0INX		
		x	(1) conversation state transitions
			(2) Pending Post state
			235
CST015	DPJL0INX		
		x	(1) conversation state transitions
			(2) state change after RECEIVE verbs
			x, x
CST0151	DPJL0INX		
		x	(1) conversation state transitions
			(2) state change after RECEIVE verbs
			(3) primary_rc parameters
			235
CST0152	DPJL0INX		
		x	(1) conversation state transitions
			(2) state change after RECEIVE verbs
			(3) what_rcvd parameters
			235
DEA01	DPJL0INX		
		x	(1) DEALLOCATE
			70
DLL01	DPJL0INX		
		x	(1) default local LU pool
			26
EXT01	DPJL0INX		
		x	(1) APPC entry points
			x, x, x, x, x, x, x, x, x, x, x, x
EXT011	DPJL0INX		
		x	(1) APPC entry points
			(2) WinAsyncAPPC( )
			28
EXT012	DPJL0INX		
		x	(1) APPC entry points
			(2) WinAsyncAPPCEx( )
			30
EXT013	DPJL0INX		
		x	(1) APPC entry points
			(2) WinAPPCCancelAsyncRequest( )
			32
EXT014	DPJL0INX		
		x	(1) APPC entry points
			(2) WinAPPCCancelBlockingCall( )
			33
EXT015	DPJL0INX		
		x	(1) APPC entry points
			(2) WinAPPCCleanup( )
			34
EXT016	DPJL0INX		
		x	(1) APPC entry points
			(2) WinAPPCIsBlocking( )
			35
EXT017	DPJL0INX		
		x	(1) APPC entry points
			(2) WinAPPCStartup( )
			36
EXT018	DPJL0INX		
		x	(1) APPC entry points
			(2) WinAPPCSetBlockingHook( )
			37
EXT019	DPJL0INX		
		x	(1) APPC entry points
			(2) WinAPPCUnhookBlockingHook( )
			38
EXT01A	DPJL0INX		
		x	(1) APPC entry points

(2) GetAppcConfig( )
39
EXT01B DPJL0INX
x (1) APPC entry points
(2) GetAppcReturnCode( )
40
EXT01C DPJL0INX
x (1) APPC entry points
(2) APPC()
27
FLU01 DPJL0INX
x (1) FLUSH
73
GTP01 DPJL0INX
x (1) GET_TP_PROPERTIES
51
GT01 DPJL0INX
x (1) GET_TYPE
54
GTA01 DPJL0INX
x (1) GET_ATTRIBUTES
75
GAC01 DPJL0INX
x (1) GetAppcConfig( )
39
GAR01 DPJL0INX
x (1) GetAppcReturnCode( )
40
GAA01 DPJL0INX
x (1) APPC()
27
HIG01 DPJL0INX
x (1) highlighting conventions
xii
INT01 DPJL0INX
x (1) introduction
5
LUAVBS DPJL0INX
x (1) LUA verbs
x, x
SUMVBS DPJL0INX
x (1) LUA verbs
(2) summary
174
LUASSY DPJL0INX
x (1) LUA verbs
(2) asynchronous verb completion
174
MCV01 DPJL0INX
x (1) mapped conversation verb control blocks
x, x, x, x, x, x, x, x, x, x, x, x, x, x, x, x
MCV011 DPJL0INX
x (1) mapped conversation verb control blocks
(2) MC_ALLOCATE
105
MCV012 DPJL0INX
x (1) mapped conversation verb control blocks
(2) MC_CONFIRM
109
MCV013 DPJL0INX
x (1) mapped conversation verb control blocks
(2) MC_CONFIRMED
111
MCV014 DPJL0INX
x (1) mapped conversation verb control blocks
(2) MC_DEALLOCATE
113
MCV015 DPJL0INX
x (1) mapped conversation verb control blocks
(2) MC_FLUSH
115
MCV016 DPJL0INX
x (1) mapped conversation verb control blocks
(2) MC_GET_ATTRIBUTES
117
MCV017 DPJL0INX
x (1) mapped conversation verb control blocks
(2) MC_PREPARE_TO_RECEIVE
120
MCV018 DPJL0INX
x (1) mapped conversation verb control blocks

			(2) MC_RECEIVE_AND_POST 122
MCV019	DPJL0INX		
		x	(1) mapped conversation verb control blocks (2) MC_RECEIVE_AND_WAIT 125
MCV01A	DPJL0INX		
		x	(1) mapped conversation verb control blocks (2) MC_RECEIVE_IMMEDIATE 128
MCV01B	DPJL0INX		
		x	(1) mapped conversation verb control blocks (2) MC_REQUEST_TO_SEND 131
MCV01C	DPJL0INX		
		x	(1) mapped conversation verb control blocks (2) MC_SEND_CONVERSATION 132
MCV01D	DPJL0INX		
		x	(1) mapped conversation verb control blocks (2) MC_SEND_DATA 136
MCV01E	DPJL0INX		
		x	(1) mapped conversation verb control blocks (2) MC_SEND_ERROR 139
MCV01F	DPJL0INX		
		x	(1) mapped conversation verb control blocks (2) MC_TEST_RTS 141
MCV01G	DPJL0INX		
		x	(1) mapped conversation verb control blocks (2) MC_TEST_RTS_AND_POST 143
MCA01	DPJL0INX		
		x	(1) MC_ALLOCATE 105
MCC01	DPJL0INX		
		x	(1) MC_CONFIRM 109
MCN01	DPJL0INX		
		x	(1) MC_CONFIRMED 111
MCD01	DPJL0INX		
		x	(1) MC_DEALLOCATE 113
MCF01	DPJL0INX		
		x	(1) MC_FLUSH 115
MCG01	DPJL0INX		
		x	(1) MC_GET_ATTRIBUTES 117
MPT01	DPJL0INX		
		x	(1) MC_PREPARE_TO_RECEIVE 120
MRP01	DPJL0INX		
		x	(1) MC_RECEIVE_AND_POST 122
MRW01	DPJL0INX		
		x	(1) MC_RECEIVE_AND_WAIT 125
MRI01	DPJL0INX		
		x	(1) MC_RECEIVE_IMMEDIATE 128
MTS01	DPJL0INX		
		x	(1) MC_REQUEST_TO_SEND 131
MSC01	DPJL0INX		
		x	(1) MC_SEND_CONVERSATION 132
MSD01	DPJL0INX		
		x	(1) MC_SEND_DATA 136
MSE01	DPJL0INX		
		x	(1) MC_SEND_ERROR 139
MTR01	DPJL0INX		
		x	(1) MC_TEST_RTS 141
MPO01	DPJL0INX		
		x	(1) MC_TEST_RTS_AND_POST

143

NAE01	DPJL0INX	x	(1)	AP_ALLOCATION_ERROR 229
NBO01	DPJL0INX	x	(1)	AP_ALLOCATION_FAILURE_NO_RETRY 229
NC01	DPJL0INX	x	(1)	AP_ALLOCATION_FAILURE_RETRY 229
NCF01	DPJL0INX	x	(1)	AP_SECURITY_NOT_VALID 229
NFR01	DPJL0INX	x	(1)	AP_TRANS_PGM_NOT_AVAIL_RETRY 229
NTM01	DPJL0INX	x	(1)	AP_TRANS_PGM_NOT_AVAIL_NO_RTRY 229
NDA01	DPJL0INX	x	(1)	AP_TP_NAME_NOT_RECOGNIZED 229
NDP01	DPJL0INX	x	(1)	AP_PIP_NOT_ALLOWED 229
NDS01	DPJL0INX	x	(1)	AP_PIP_NOT_SPECIFIED_CORRECTLY 229
NDT01	DPJL0INX	x	(1)	AP_CONVERSATION_TYPE_MISMATCH 230
NDN01	DPJL0INX	x	(1)	AP_SYNC_LEVEL_NOT_SUPPORTED 230
NPP01	DPJL0INX	x	(1)	AP_CONV_FAILURE_NO_RETRY 230
NPN01	DPJL0INX	x	(1)	AP_CONV_FAILURE_RETRY 230
NNT01	DPJL0INX	x	(1)	AP_CONVERSATION_TYPE_MIXED 230
NEP01	DPJL0INX	x	(1)	AP_DEALLOC_ABEND 230
NET01	DPJL0INX	x	(1)	AP_DEALLOC_ABEND_PROGRAM 230
NTB01	DPJL0INX	x	(1)	AP_DEALLOC_ABEND_SVC 230
NUS01	DPJL0INX	x	(1)	AP_DEALLOC_ABEND_TIMER 230
NDZ01	DPJL0INX	x	(1)	AP_DEALLOC_NORMAL 231
NPZ01	DPJL0INX	x	(1)	AP_PROG_ERROR_NO_TRUNC 231
NEZ01	DPJL0INX	x	(1)	AP_PROG_ERROR_PURGING 231
NTZ01	DPJL0INX	x	(1)	AP_PROG_ERROR_TRUNC 231
NSZ01	DPJL0INX	x	(1)	AP_SVC_ERROR_NO_TRUNC 231
NVZ01	DPJL0INX	x	(1)	AP_SVC_ERROR_PURGING 231
NNZ01	DPJL0INX	x	(1)	AP_SVC_ERROR_TRUNC 231
NBZ01	DPJL0INX	x	(1)	AP_TP_BUSY 231
NUZ01	DPJL0INX	x	(1)	AP_UNEXPECTED_SYSTEM_ERROR

231

OSS01 DPJL0INX
x (1) options sets supported by Personal Communications
23

OVR012 DPJL0INX
x (1) APPC API support
x, x, x, x

OVR0121 DPJL0INX
x (1) APPC API support
(2) verbs supported
43

OVR0122 DPJL0INX
x (1) APPC API support
(2) option sets supported
23

OVR0124 DPJL0INX
x (1) APPC API support
(2) queue-level nonblocking
24

OVR0125 DPJL0INX
x (1) APPC API support
(2) default local LU pool
26

PPC01 DPJL0INX
x (1) WinAsyncAPPC( )
28

PCE01 DPJL0INX
x (1) WinAsyncAPPCEx( )
30

QUE01 DPJL0INX
x (1) queue-level nonblocking support
x, x

QUE011 DPJL0INX
x (1) queue-level nonblocking support
(2) explanation of
24

QUE012 DPJL0INX
x (1) queue-level nonblocking support
(2) three types of queues
24

STA01 DPJL0INX
x (1) WinAPPCStartup( )
36

SBH01 DPJL0INX
x (1) WinAPPCSetBlockingHook( )
37

TIV01 DPJL0INX
x (1) type independent verb control blocks
x, x, x, x, x

TIV015 DPJL0INX
x (1) type independent verb control blocks
(2) GET_TP_PROPERTIES
51

TIV016 DPJL0INX
x (1) type independent verb control blocks
(2) GETE_TYPE
54

TIV017 DPJL0INX
x (1) type independent verb control blocks
(2) RECEIVE_ALLOCATE
56

TIV01A DPJL0INX
x (1) type independent verb control blocks
(2) TP_ENDED
58

TIV01B DPJL0INX
x (1) type independent verb control blocks
(2) TP_STARTED
60

UBH01 DPJL0INX
x (1) WinAPPCUnhookBlockingHook( )
38

VRB01 DPJL0INX
x (1) verbs supported at the APPC API
x, x, x

VRB012 DPJL0INX
x (1) verbs supported at the APPC API
(2) type independent verbs
44

VRB013 DPJL0INX
x (1) verbs supported at the APPC API

(2) basic conversation verbs
44

VRB014 DPJL0INX
x (1) verbs supported at the APPC API
(2) mapped conversation verbs
44

VSG01 DPJL0INX
x (1) verb signals
x, x, x

VSG012 DPJL0INX
x (1) verb signals
(2) verb control blocks
x

VSG0121 DPJL0INX
x (1) verb signals
(2) verb control blocks
(3) common fields
43

VSG015 DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
x, x, x, x, x, x, x, x, x, x, x, x, x, x, x, x

VSG0151 DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) ALLOCATE
62

VSG0152 DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) CONFIRM
66

VSG0153 DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) CONFIRMED
68

VSG0154 DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) DEALLOCATE
70

VSG0155 DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) FLUSH
73

VSG0156 DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) GET_ATTRIBUTES
75

VSG0157 DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) PREPARE_TO_RECEIVE
78

VSG0158 DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) RECEIVE_AND_POST
80

VSG0159 DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) RECEIVE_AND_WAIT
84

VSG015A DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) RECEIVE_IMMEDIATE
87

VSG015B DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) REQUEST_TO_SEND
90

VSG015C DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) SEND_CONVERSATION

92

VSG015D DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) SEND_DATA
95

VSG015E DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) SEND_ERROR
98

VSG015F DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) TEST_RTS
101

VSG015G DPJL0INX
x (1) verb signals
(2) basic conversation verb control blocks
(3) TEST_RTS_AND_POST
103

VSG016 DPJL0INX
x (1) verb signals
(2) mapped conversation verb control blocks
x, x, x, x, x, x, x, x, x, x, x, x, x, x, x, x

VSG0161 DPJL0INX
x (1) verb signals
(2) mapped conversation verb control blocks
(3) MC_ALLOCATE
105

VSG0162 DPJL0INX
x (1) verb signals
(2) mapped conversation verb control blocks
(3) MC_CONFIRM
109

VSG0163 DPJL0INX
x (1) verb signals
(2) mapped conversation verb control blocks
(3) MC_CONFIRMED
111

VSG0164 DPJL0INX
x (1) verb signals
(2) mapped conversation verb control blocks
(3) MC_DEALLOCATE
113

VSG0165 DPJL0INX
x (1) verb signals
(2) mapped conversation verb control blocks
(3) MC_FLUSH
115

VSG0166 DPJL0INX
x (1) verb signals
(2) mapped conversation verb control blocks
(3) MC_GET_ATTRIBUTES
117

VSG0167 DPJL0INX
x (1) verb signals
(2) mapped conversation verb control blocks
(3) MC_PREPARE_TO_RECEIVE
120

VSG0168 DPJL0INX
x (1) verb signals
(2) mapped conversation verb control blocks
(3) MC_RECEIVE_AND_POST
122

VSG0169 DPJL0INX
x (1) verb signals
(2) mapped conversation verb control blocks
(3) MC_RECEIVE_AND_WAIT
125

VSG016A DPJL0INX
x (1) verb signals
(2) mapped conversation verb control blocks
(3) MC_RECEIVE_IMMEDIATE
128

VSG016B DPJL0INX
x (1) verb signals
(2) mapped conversation verb control blocks
(3) MC_REQUEST_TO_SEND
131

VSG016C DPJL0INX

		x	(1) verb signals (2) mapped conversation verb control blocks (3) MC_SEND_CONVERSATION 132
VSG016D	DPJL0INX		
		x	(1) verb signals (2) mapped conversation verb control blocks (3) MC_SEND_DATA 136
VSG016E	DPJL0INX		
		x	(1) verb signals (2) mapped conversation verb control blocks (3) MC_SEND_ERROR 139
VSG016F	DPJL0INX		
		x	(1) verb signals (2) mapped conversation verb control blocks (3) MC_TEST_RTS 141
VSG016G	DPJL0INX		
		x	(1) verb signals (2) mapped conversation verb control blocks (3) MC_TEST_RTS_AND_POST 143
VCB01	DPJL0INX		
		x	(1) verb control blocks x
VCB011	DPJL0INX		
		x	(1) verb control blocks (2) common fields 43
ACHMGR	DPJL0INX		
		x	(1) attach manager x, x, x, x, x
ACHDES	DPJL0INX		
		x	(1) attach manager (2) description 11
ACHSTA	DPJL0INX		
		x	(1) attach manager (2) starting 15
ACHIDT	DPJL0INX		
		x	(1) attach manager (2) identifying transition program name 12
ACHPGM	DPJL0INX		
		x	(1) attach manager (2) starting programs 16
ACHMTC	DPJL0INX		
		x	(1) attach manager (2) matching incoming allocation requests x, x
MTCNONQ	DPJL0INX		
		x	(1) attach manager (2) matching incoming allocation requests (3) nonqueued programs 17
MTCYESQ	DPJL0INX		
		x	(1) attach manager (2) matching incoming allocation requests (3) queued programs 17
VERCSV	DPJL0INX		
		x	(1) common services verbs x, x
GETCVT	DPJL0INX		
		x	(1) common services verbs (2) GET_CP_CONVERT_TABLE 157
CONVRT	DPJL0INX		
		x	(1) common services verbs (2) CONVERT 160
ENPCSV	DPJL0INX		
		x	(1) common services entry points x, x, x, x, x, x, x
ACSSV1	DPJL0INX		
		x	(1) common services entry points (2) ACSSVC

147
WINCS1 DPJL0INX
x (1) common services entry points
(2) WinCSV
147
WINCLN DPJL0INX
x (1) common services entry points
(2) WinCSVCleanup
148
WINSNK DPJL0INX
x (1) common services entry points
(2) WinCSVAsyncCSV
149
WINSTA DPJL0INX
x (1) common services entry points
(2) WinCSVStartup
150
GETCRC DPJL0INX
x (1) common services entry points
(2) GetCsvReturnCode
151
EPTRNS DPJL0INX
x (1) common services entry points
(2) TrnsDt
152
TALKER DPJL0INX
x (1) conversation
x, x, x, 7
TALDEF DPJL0INX
x (1) conversation
(2) defining attributes
20, 21
TALINC DPJL0INX
x (1) conversation
(2) security for incoming allocation requests
21
TALOUT DPJL0INX
x (1) conversation
(2) security for outgoing allocation requests
22
CHANGE DPJL0INX
x (1) CONVERT
160
CTABLE DPJL0INX
x (1) GET_CP_CONVERT_TABLE
157
LU1 DPJL0INX
x (1) LU
x, x, x, x
LU1DES DPJL0INX
x (1) LU
(2) description of
6
LU1TYP DPJL0INX
x (1) LU
(2) types
6
LU1DEP DPJL0INX
x (1) LU
(2) dependent
6
LU1IND DPJL0INX
x (1) LU
(2) independent
7
LUA001 DPJL0INX
x (1) LUA
x, x, x, x, x, x, x, x, x, x, x, x
LUAUND DPJL0INX
x (1) LUA
(2) understanding
165
LUACON DPJL0INX
x (1) LUA
(2) connection capabilities
165
LUAAPP DPJL0INX
x (1) LUA
(2) application programs
165
LUAVER DPJL0INX

		x	(1) LUA (2) verbs 165, 174
LUALOC	DPJL0INX		
		x	(1) LUA (2) LUs, local and partner 166
LUASNA	DPJL0INX		
		x	(1) LUA (2) SNA layers 166
LUASAM	DPJL0INX		
		x	(1) LUA (2) sample LUA communication sequence 175
LUARES	DPJL0INX		
		x	(1) LUA (2) restarting and resynchronizing 170
LUACOM	DPJL0INX		
		x	(1) LUA (2) compatibility 176
LUARUI	DPJL0INX		
		x	(1) LUA (2) RUI sessions 181
LUAUSE	DPJL0INX		
		x	(1) LUA (2) using SNA sessions x, x, x, x, x
USEPRE	DPJL0INX		
		x	(1) LUA (2) using SNA sessions (3) prerequisites 168
USESTA	DPJL0INX		
		x	(1) LUA (2) using SNA sessions (3) starting 168
USETRN	DPJL0INX		
		x	(1) LUA (2) using SNA sessions (3) transferring data on an LU-LU session 169
USESTO	DPJL0INX		
		x	(1) LUA (2) using SNA sessions (3) stopping 169
USEDIS	DPJL0INX		
		x	(1) LUA (2) using SNA sessions (3) disconnecting 170
LUAISS	DPJL0INX		
		x	(1) LUA (2) issuing RUI verbs
RUIVER	DPJL0INX		
		x	(1) RUI verbs x, x
RUILUS	DPJL0INX		
		x	(1) RUI verbs (2) LUA verb control format 195
RUICOM	DPJL0INX		
		x	(1) RUI verbs (2) common verb header 196
MESNUM	DPJL0INX		
		x	(1) message numbers 170
PROCOL	DPJL0INX		
		x	(1) protocols x, x, x, x
PROPAC	DPJL0INX		
		x	(1) protocols (2) pacing 171
PRODUP	DPJL0INX		

		x	(1) protocols (2) half-duplex contention flip/flop 172
PROBRA	DPJL0INX		
		x	(1) protocols (2) bracket 172
PRODAT	DPJL0INX		
		x	(1) protocols (2) data-chaining 173
SESHIN	DPJL0INX		
		x	(1) session 7
SNA001	DPJL0INX		
		x	(1) SNA x, x
SNACOM	DPJL0INX		
		x	(1) SNA (2) communication support 3
SNATYP	DPJL0INX		
		x	(1) SNA (2) LU type 6.2 support 4
TRAPRO	DPJL0INX		
		x	(1) transaction program x, x, x, x, x, x, x, x, x
TRADES	DPJL0INX		
		x	(1) transaction program (2) description of 5
TRAAPP	DPJL0INX		
		x	(1) transaction program (2) APPC 5
TRACPI	DPJL0INX		
		x	(1) transaction program (2) CPI Communications 6
TRADIF	DPJL0INX		
		x	(1) transaction program (2) compared to an application 16
TRADEF	DPJL0INX		
		x	(1) transaction program (2) definitions 20
TRAWRI	DPJL0INX		
		x	(1) transaction program (2) writing 23
TRASUP	DPJL0INX		
		x	(1) transaction program (2) supported option sets 23
TRAQUE	DPJL0INX		
		x	(1) transaction program (2) queue-level nonblocking 24
TRALOC	DPJL0INX		
		x	(1) transaction program (2) default local LU pool 26
TRNSDT	DPJL0INX		
		x	(1) TrnsDt 152
WINCSV	DPJL0INX		
		x	(1) WinCSV 147
WINCLE	DPJL0INX		
		x	(1) WinCSVCleanup 148
WINSIN	DPJL0INX		
		x	(1) WinAsyncCSV 149
WINSUP	DPJL0INX		
		x	(1) WinCSVStartup 150
WRIPRO	DPJL0INX		
		x	(1) writing LUA APPC program

			x, x
WRICAL	DPJL0INX		
		x	(1) writing LUA APPC program (2) calling Dynamic Link Libraries x, 183
WRIENT	DPJL0INX		
		x	(1) writing LUA APPC program (2) prodecure entry points 183
WRIDLL	DPJL0INX		
		x	(1) writing LUA APPC program (2) DLL interface, using 193

List Items

id	File	Page	References
DEF0TO3	DPJL0CNV		
		179	2 179

Spots

id	File	Page	References
VAPPC	DPJL0OVR		
		27	(no text)
WNASY	DPJL0OVR		
		28	(no text)
WNAPEX	DPJL0OVR		
		30	(no text)
WNCARQ	DPJL0OVR		
		32	(no text)
WNCBLC	DPJL0OVR		
		33	(no text)
WNCLUP	DPJL0OVR		
		34	(no text)
WNBLOC	DPJL0OVR		
		35	(no text)
WNSTAR	DPJL0OVR		
		36	(no text)
WNBLHK	DPJL0OVR		
		37	(no text)
WNUNBH	DPJL0OVR		
		38	(no text)
GETCG	DPJL0OVR		
		39	(no text)
GETRT	DPJL0OVR		
		40	(no text)
TPID	DPJL0SIG		
		43	(no text)
ACSSVC	DPJL0EXT		
		147	(no text)
WINCSV	DPJL0EXT		
		147	(no text)
CSVCLNP	DPJL0EXT		
		148	(no text)
ASYCSV	DPJL0EXT		
		149	(no text) 147
VFORM	DPJL0EXT		
		160	(no text)

Tables

id	File	Page	References
T2C2	DPJL0AAM	19	1 19
FMP	DPJL0UND	181	2 181
STATE1	DPJL0UND	182	3 181

Processing Options

Runtime values:

Document fileid	DPJL0MST SCRIPT
Document type	USERDOC
Document style	IBMXAGD
Profile	EDFPRF40
Service Level	0030
SCRIPT/VS Release	4.0.0
Date	96.03.25
Time	16:24:42
Device	PSA
Number of Passes	4
Index	YES
SYSVAR D	YES
SYSVAR V	IDD2
SYSVAR X	YES
SYSVAR 1	ISO9000

Formatting values used:

Annotation	NO
Cross reference listing	YES
Cross reference head prefix only	NO
Dialog	LABEL
Duplex	YES
DVCF conditions file	(none)
DVCF value 1	ISO9000
DVCF value 2	(none)
DVCF value 3	(none)
DVCF value 4	(none)
DVCF value 5	(none)
DVCF value 6	(none)
DVCF value 7	(none)
DVCF value 8	(none)
DVCF value 9	(none)
Explode	NO
Figure list on new page	YES
Figure/table number separation	YES
Folio-by-chapter	NO
Head 0 body text	Part
Head 1 body text	Chapter
Head 1 appendix text	Appendix
Hyphenation	NO
Justification	NO
Language	ENGL
Keyboard	395
Layout	OFF
Leader dots	YES
Master index	(none)
Partial TOC (maximum level)	3
Partial TOC (new page after)	RHPAGE
Print example id's	NO
Print cross reference page numbers	YES
Process value	(none)
Punctuation move characters	.,
Read cross-reference file	(none)
Running heading/footing rule	NONE
Show index entries	NO
Table of Contents (maximum level)	3
Table list on new page	NO
Title page (draft) alignment	RIGHT
Write cross-reference file	(none)

Imbed Trace

Page 0	DPJL0SET
Page i	DPJL0EDN
Page vii	DPJL0NOT
Page x	CMTRADEN
Page x	DPJL0INX
Page x	DPJL0PRE
Page xiv	DPJL0NEW
Page 4	DPJL0INT
Page 9	DPJL0AAM
Page 22	DPJL0APP
Page 26	DPJL0OVR
Page 41	DPJL0SIG
Page 144	DPJL0EXT
Page 162	DPJL0CNV
Page 179	DPJL0UND
Page 182	DPJL0WRI
Page 194	DPJL0DEF
Page 228	DPJL0RET
Page 232	DPJL0APB
Page 236	DPJL0GLO
Page 276	RCFADDR
Page 279	RCFADDR
Page 282	DPJL0C2