IBM Communications Server for Data Center Deployment
on AIX or Linux

APPC Programmer's Guide

Version 7.0

SC23-8592-01

IBM Communications Server for Data Center Deployment
on AIX or Linux

APPC Programmer's Guide

Version 7.0

SC23-8592-01

Note:
Before using this information and the product it supports, be sure to read the general information under Appendix E, "Notices," on page 291.

Sixth Edition (December 2012)

This edition applies to IBM Communications Server for Data Center Deployment on AIX or Linux, Version 7.0, program number 5725-H32, and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters.

IBM welcomes your comments. You may send your comments to the following address.

International Business Machines Corporation
Attn: z/OS Communications Server Information Development
Department AKCA, Building 501
P.O. Box 12195, 3039 Cornwallis Road
Research Triangle Park, North Carolina 27709-2195

You can send us comments electronically by using one of the following methods:

Fax (USA and Canada):

1+919-254-1258

Send the fax to "Attn: z/OS Communications Server Information Development"

Internet email:

comsvrcf@us.ibm.com

World Wide Web:

http://www.ibm.com/systems/z/os/zos/webqs.html

If you would like a reply, be sure to include your name, address, telephone number, or FAX number. Make sure to include the following in your comment or note:

- Title and order number of this document
- Page number or topic related to your comment

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.

Contents

Tables

Figures

About This Book

This book is a guide for developing C programming language application programs that use Advanced Program-to-Program Communications (APPC) to exchange data in a Systems Network Architecture (SNA) environment.

This manual applies to IBM® Communications Server for Data Center Deployment (Communications Server), program product number 5725-H32, which is an IBM software product that enables a server running AIX®, or a computer running Linux, to exchange information with other nodes on an SNA network.

There are three different installation variants of IBM Communications Server for Data Center Deployment, depending on the hardware on which it operates:

IBM Communications Server for Data Center Deployment on AIX (CS/AIX)
IBM Communications Server for Data Center Deployment on AIX operates on a server running AIX Version 6.1 or 7.1 base operating system.

IBM Communications Server for Data Center Deployment on Linux (CS Linux)
IBM Communications Server for Data Center Deployment on Linux operates on the following:

- 32–bit Intel workstations running Linux (i686)
- 64–bit AMD64/Intel EM64T workstations running Linux (x86_64)
- IBM pSeries® computers running Linux (ppc64)

IBM Communications Server for Data Center Deployment on Linux for System z® (CS Linux for System z)
IBM Communications Server for Data Center Deployment on Linux for System z operates on System z mainframes running Linux for System z (s390x).

In this book, the name Communications Server is used to indicate any of these variants, and the term "Communications Server computer" is used to indicate any type of computer running Communications Server, except where differences are described explicitly.

The Communications Server implementation of APPC is based on IBM's implementation of APPC in its OS/2 products (such as **Communications Server for OS/2**), with modifications for the AIX / Linux environment.

Programs written to use the Communications Server implementation of APPC can exchange data with programs written to use other implementations of APPC that adhere to the SNA Logical Unit (LU) 6.2 architecture.

This book applies to Version 7.0 of Communications Server.

Who Should Use This Book

This book is intended for experienced C programmers who write Systems Network Architecture (SNA) transaction programs for systems with Communications Server. Programmers may or may not have prior experience with SNA or the communication facilities of Communications Server.

System Administrators

System Administrators install Communications Server, configure the system for network connection, and maintain the system. They should be familiar with the hardware on which Communications Server operates and with the AIX / Linux operating system. They must also be knowledgeable about the network to which the system is connected and understand SNA concepts in general.

Application Programmers

Application programmers design and code transaction and application programs that use the Communications Server programming interfaces to send and receive data over an SNA network. They should be thoroughly familiar with SNA, the remote program with which the transaction or application program communicates, and the AIX / Linux operating system programming and operating environments.

More detailed information about writing application programs is provided in the manual for each API. For additional information about Communications Server publications, see the bibliography.

How to Use This Book

This section explains how information is organized and presented in this book.

Organization of This Book

This book is organized as follows:

- Chapter 1, "Concepts," on page 1, introduces the fundamental concepts of APPC. It is intended for programmers who are not familiar with APPC.
- Chapter 2, "Writing Transaction Programs," on page 25, contains general information an APPC programmer needs when writing transaction programs (TPs).
- Chapter 3, "APPC Control Verbs," on page 63, describes each APPC control verb in detail. Each description includes the following: purpose, verb control block (VCB), supplied and returned parameters, conversation states in which the verb can be issued, conversation state changes after the verb has executed. Differences between the implementations of APPC for different operating systems are indicated where they occur; these are generally minor variations due to operating system differences.
- Chapter 4, "APPC Conversation Verbs," on page 91, describes each APPC conversation verb in detail. Each description includes the following: purpose, verb control block (VCB), supplied and returned parameters, conversation states in which the verb can be issued, conversation state changes after the verb has executed. Differences between the implementations of APPC for different operating systems are indicated where they occur; these are generally minor variations due to operating system differences.
- Chapter 5, "TP Server Verbs," on page 245, describes each APPC TP server verb in detail. Each description includes the following: purpose, verb control block (VCB), supplied and returned parameters, conversation states in which the verb can be issued, conversation state changes after the verb has executed. Differences between the implementations of APPC for different operating systems are indicated where they occur; these are generally minor variations due to operating system differences.
- Chapter 6, "Sample Transaction Programs," on page 261, describes the sample APPC transaction programs, which illustrate the use of APPC verbs. This

chapter also includes instructions for compiling, linking, and running the TPs on each of the supported operating systems.

- Appendix A, "Return Code Values," on page 265, lists all the possible return codes in the APPC interface in numerical order and gives their meanings.
- Appendix B, "Common Return Codes," on page 273, documents certain primary and secondary return codes that are common to several verbs.
- Appendix C, "APPC State Changes," on page 283, provides information about APPC conversation states: which verbs are permitted in each state, and the state to which the conversation changes on return from each verb.
- Appendix D, "SNA LU 6.2 Support," on page 289, provides reference information about how the Communications Server implementation of APPC relates to the SNA LU 6.2 architecture, and about the LU 6.2 control operator verbs whose function is provided in Communications Server by the command-line administration program **snaadmin** and by NOF (node operator facility) verbs.

Typographic Conventions

Table 1 shows the typographic styles used in this document.

Table 1. Typographic Conventions

Special Element	Sample of Typography
Document title	*IBM Communications Server for Data Center Deployment on AIX or Linux APPC Application Suite User's Guide*
File or path name	**sna_tps**
Program or application	**snaadmin**
Command or AIX / Linux utility	**vi**; **define_mode**
General reference to all values of a particular type	`AP_SEC_BAD_*` (indicates all of the return values that begin with `AP_SEC_BAD`)
Option or flag	**-I**
Parameter or Motif field	*primary_rc*; *what_rcvd*
Literal value or selection that the user can enter (including default values)	`0`; `32,767`
Constant or signal	`AP_DATA_COMPLETE_CONFIRM`
Return value	`AP_OK`; `AP_SYNC_LEVEL_NOT_SUPPORTED`; `TRUE`
Variable representing a supplied value	*lParam*; *ReturnedHandle*
Environment variable	`LD_RUN_PATH`
Programming verb	RECEIVE_ALLOCATE
User input	**cc -I**
Function, call, or entry point	`APPC_Async`; `WinAsyncAPPC`
Data structure	`WAPPCDATA`
Hexadecimal value	`0x20`

Graphic Conventions

UNIX

This symbol is used to indicate the start of a section of text that applies only to the AIX or Linux operating system. It applies to AIX / Linux servers and to the IBM Remote API Client running on AIX, Linux, Linux for pSeries or Linux for System z.

This symbol is used to indicate the start of a section of text that applies to the IBM Remote API Client on Windows.

This symbol indicates the end of a section of operating system specific text. The information following this symbol applies regardless of the operating system.

What Is New for This Release

Communications Server for Data Center Deployment Version 7.0 is a follow-on product to Distributed Communications Server Version 6.4, which continues to be supported.

Where to Find More Information

See the bibliography for other books in the Communications Server library, as well as books that contain additional information about topics related to SNA and AIX / Linux workstations.

Chapter 1. Concepts

This chapter introduces the fundamental concepts of advanced program-to-program communication (APPC) in a distributed processing environment:

- What is APPC?
- A simple mapped conversation
- Confirmation processing
- Sending and receiving status with data
- Conversation states
- Changing conversation states
- Synchronous and asynchronous APPC calls
- Receiving data asynchronously

UNIX

- Syncpoint support

- APPC and CPI-C (Common Programming Interface for Communications)
- TP Server API

What Is APPC?

APPC stands for Advanced Program-to-Program Communication, an application program interface (API) that enables peer-to-peer communications among programs in a Systems Network Architecture (SNA) environment.

Through APPC, application programs distributed across a network can work together, communicating with each other and exchanging data to accomplish a single processing task such as the following:

- Querying a remote database
- Copying a remote file
- Sending or receiving electronic mail

A complete sequence of communications between two application programs, which can accomplish one or more processing tasks, is referred to as a conversation.

Two communicating APPC applications can be on the same computer or on two separate computers; an application does not need to know where its partner application is located. An APPC application can run either on a server or on a client computer.

Transaction Programs

A transaction is a processing task accomplished by programs using APPC. Consequently, programs that use APPC are called transaction programs (TPs). These programs communicate as peers, on an equal (rather than a hierarchical) basis. Application TPs accomplish tasks for end users. Service TPs provide services to other programs.

Together, TPs distributed across a local- or wide-area network perform distributed transaction processing.

Communication between TPs

Many hardware and software elements in the SNA environment are required for two TPs to communicate with each other. Figure 1 illustrates the elements that pertain directly to programmers writing TPs.

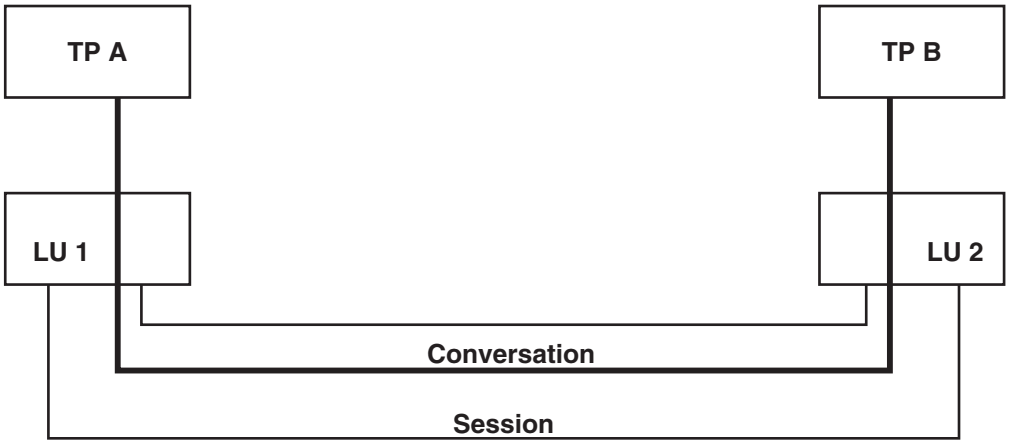

Figure 1. Elements for Writing TPs

Logical Unit 6.2

Each TP is associated with a logical unit (LU) on a particular Communications Server local node and accesses the network through that LU. Several TPs can be associated with the same LU. Each LU type uses a specific protocol. APPC is supported by LU 6.2.

Sessions

Before two TPs can communicate, their LUs must be connected through an LU-to-LU session. An LU-to-LU session is a logical connection between the two LUs. The session's mode (set of networking characteristics) determines how data moves between the two LUs.

Conversations

The communication between the two TPs occurs as a conversation within the LU-to-LU session. A TP can be involved in several conversations simultaneously.

APPC Verbs

A TP accesses APPC through APPC verbs. The TPs use these verbs to instruct APPC to start a conversation, send or receive data, and end a conversation. Each verb supplies parameters to APPC, which performs the desired function and returns parameters to the TP. Some verbs complete quickly, after some local processing (for example sending a small amount of data); other verbs may take some time to complete, depending on the partner TP and the communications path (for instance, waiting for data or confirmation from the partner TP).

The TP issuing the verb is referred to as the local TP; the other TP is referred to as the partner TP.

Similarly, the LU serving the local TP is the local LU and the LU serving the partner TP is the partner LU.

TPs and LUs residing on other network nodes are also called remote TPs and remote LUs.

The Conversation Process

A conversation begins when both of the following happen:

1. One TP (the invoking TP) instructs APPC to start another TP (the invoked TP) and allocate a conversation between the two TPs.
2. The invoked TP notifies APPC that it is ready to communicate with the invoking TP.

During the conversation, the two TPs exchange status information and application data.

A conversation ends when a TP instructs APPC to deallocate the conversation.

Conversation Types

A conversation can be mapped or basic. At allocation time, the invoking TP specifies whether a conversation is to be basic or mapped. Certain APPC verbs are used in mapped conversations only; others are used in basic conversations. You cannot use a basic-conversation verb in a mapped conversation or vice versa.

In general, mapped conversations are used by application TPs. Application TPs are programs that accomplish tasks for end users. Mapped conversations are less complex than basic conversations. In a mapped conversation, the sending TP sends one record at a time: the receiving TP receives one record at a time.

Basic conversations are normally used by service TPs. Service TPs are programs that provide services to other local programs. Basic conversations provide an experienced LU 6.2 programmer with a greater degree of control over the transmission and handling of data. For more information, see “Basic Conversations” on page 58.

Multiple Conversations

A TP can be involved in several conversations simultaneously. Each conversation requires an LU-to-LU session.

A common use of multiple conversations is for an invoked TP to invoke another TP, which, in turn, invokes another TP, and so on. Figure 2 on page 4 shows how TP A invokes TP B, and TP B then invokes TP C. TPs A and C are not in conversation with each other, but only with TP B.

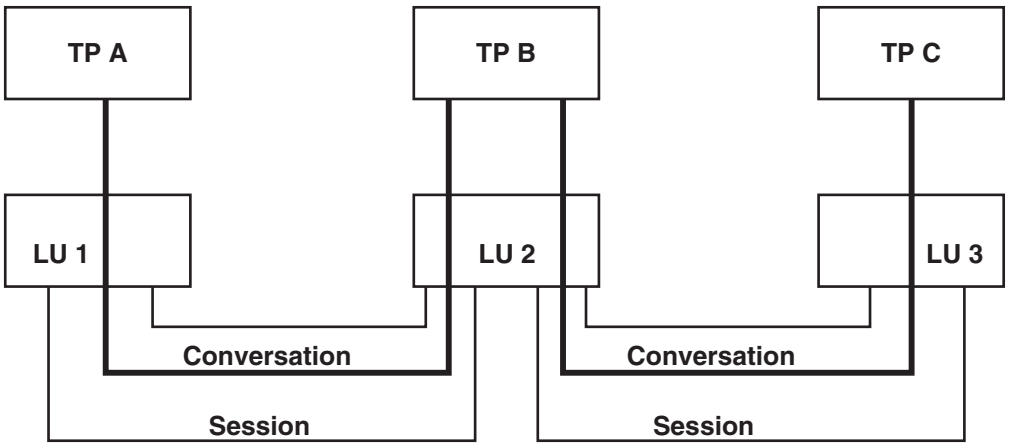

Figure 2. Invoking TPs Using Multiple Conversations

Half-Duplex and Full-Duplex Conversations

Depending on how the two TPs in a conversation need to interact, the conversation can operate in two ways.

Half-duplex

In a half-duplex conversation, control is passed between the two TPs so that one TP is always in control of the conversation at any time. The controlling TP can send data, or can pass control to the other TP. The other TP can receive data but cannot send it; however, it can send an error indication to the controlling TP, or can request control of the conversation.

Half-duplex conversations are normally used in the following situations:

- When the progress of the conversation depends on the data being transmitted; for example, if the first TP sends a request that the second TP must process, and the subsequent operation of both TPs depends on whether the request is accepted or rejected.
- When the first TP simply sends data to the second TP and does not need to receive a response.

Because control is passed between the two TPs, half-duplex conversations allow the TPs to confirm each stage of the conversation before continuing. After sending a quantity of data, the first TP can request confirmation that the data has been received and processed correctly before it continues. At this point, the second TP may confirm the data and allow the first TP to send more, or it may return an error response and then itself take control of the conversation (for example to provide more details of the error or to return a corrected version of the data).

Full-duplex

In a full-duplex conversation, both TPs can send data at any time; neither TP is considered to be in control.

Full-duplex conversations are normally used when the data being sent in the two directions is independent, so that the progress of the conversation does not depend on the data being transmitted. Because neither TP is considered to be in control of the conversation, confirmation processing is not supported; a TP can send an error response at any time, but this does not prevent the other TP from continuing to send data.

The TP that allocates the conversation specifies whether the conversation will operate as a full-duplex or half-duplex conversation. This choice then applies for the duration of the conversation; you cannot change between full-duplex and half-duplex operation during the conversation. To issue a verb in a full-duplex conversation, the TP must set an additional option in the *opext* parameter of the verb, as described in Chapter 4, "APPC Conversation Verbs," on page 91.

Not all APPC implementations support full-duplex operation. If the remote TP is using an APPC implementation that does not support full-duplex operation, the TPs can operate only in half-duplex mode.

The following sections describe the operation of half-duplex conversations. Because full-duplex conversations do not include the verbs used for confirmation processing and passing control between the TPs, some of the information in these sections does not apply to full-duplex conversations. See "Full-Duplex Conversations" on page 14 for a summary of the operation of a full-duplex conversation.

A Simple Mapped Conversation (half-duplex)

Table 2 shows an example of a simple mapped conversation showing the APPC verbs used to start a conversation, exchange data, and end the conversation. The arrow indicates the flow of data. All *primary_rc* values are AP_OK unless shown otherwise.

Table 2. A Simple Mapped Conversation

Invoking TP	Flow	Invoked TP
TP_STARTED MC_ALLOCATE MC_SEND_DATA MC_DEALLOCATE		
	—>	
TP_ENDED		RECEIVE_ALLOCATE MC_RECEIVE_AND_WAIT (*what_rcvd* = AP_DATA_COMPLETE) MC_RECEIVE_AND_WAIT (*primary_rc* = AP_DEALLOC_NORMAL) TP_ENDED

The characters MC_ at the beginning of many of the verbs stand for mapped conversation. Parameters and results of APPC verbs are in parentheses.

Starting a Conversation

To start a conversation, the invoking TP issues the following verbs:

- TP_STARTED, which identifies the application to APPC as an invoking TP
- MC_ALLOCATE, which requests that APPC establish a conversation between the invoking TP and the invoked TP

The invoked TP issues the RECEIVE_ALLOCATE verb, which informs APPC that the invoked TP is ready to begin a conversation. Communications Server associates the RECEIVE_ALLOCATE verb with the MC_ALLOCATE verb issued by the invoking TP in order to establish the conversation between the two TPs.

Sending Data

The MC_SEND_DATA verb supplies application data to be transmitted to the partner TP. This data is held in the local LU's send buffer; it is not transmitted to the partner TP until one of the following events occurs:

- The send buffer fills up.
- The TP issues a verb that forces APPC to flush the buffer (send data to the partner TP).

In addition to the data, the send buffer also contains the MC_ALLOCATE request (which precedes the data). In this example, the MC_DEALLOCATE verb flushes the buffer, transmitting the MC_ALLOCATE request and data to the partner TP. Other verbs that flush the buffer are MC_CONFIRM and MC_FLUSH.

Receiving Data

The MC_RECEIVE_AND_WAIT verb receives data from the partner TP. If no data is currently available, the local TP waits for it to arrive.

As well as receiving data, the verb may receive a status indicator from the partner TP (such as an indication that the conversation is ending, a request to confirm receipt of data, and so on). For more information about how the TP handles these indicators, see "Confirmation Processing (half-duplex)" and "Conversation States (half-duplex)" on page 10.

In the example, the invoked TP issues the first MC_RECEIVE_AND_WAIT verb to receive data. When it has finished receiving the complete data record (*what_rcvd*= `AP_DATA_COMPLETE`), it issues the MC_RECEIVE_AND_WAIT verb again to receive a return code. The return code `AP_DEALLOC_NORMAL` indicates that the conversation was deallocated.

The MC_RECEIVE_IMMEDIATE verb performs the same function as the MC_RECEIVE_AND_WAIT verb, except that it does not wait if data is not currently available from the partner TP. Instead, it returns a no-data-available response to the calling TP.

Ending a Conversation

To end a conversation, one of the TPs issues the MC_DEALLOCATE verb, which causes APPC to deallocate the conversation between the two TPs.

After this conversation is deallocated, each TP either issues another [MC_]ALLOCATE or RECEIVE_ALLOCATE verb to start another conversation (with this or another partner TP), or issues the TP_ENDED verb.

A TP can participate in multiple conversations simultaneously. In this case, the TP issues the TP_ENDED verb when all conversations have been deallocated.

Confirmation Processing (half-duplex)

Using confirmation processing, a TP sends a confirmation request with the data; the receiving TP confirms receipt of the data or indicates that an error occurred. Each time the two TPs exchange a confirmation request and response, they are synchronized.

The example of confirmation processing in Table 3 on page 7 shows two ways of confirming the transfer of data: requesting confirmation after sending the data (by

using the CONFIRM verb), and requesting confirmation at the end of a transaction (by requesting confirmation on the DEALLOCATE verb). Confirmation can also be requested on the PREPARE_TO_RECEIVE verb; this asks the partner TP to confirm receipt of data, and then begin to send data itself. For more information, see "State Changes" on page 11. A pair of TPs may choose to use only one of these mechanisms; in the following example, the invoked TP uses the PREPARE_TO_RECEIVE verb without requesting confirmation; this simply tells the partner TP to send data.

Table 3. Confirmation Processing

Invoking TP	Flow	Invoked TP
TP_STARTED MC_ALLOCATE (*sync_level* = AP_CONFIRM_SYNC_LEVEL) MC_SEND_DATA MC_CONFIRM		
	—>	
		RECEIVE_ALLOCATE MC_RECEIVE_AND_WAIT (*primary_rc* = AP_OK) (*what_rcvd* = AP_DATA_COMPLETE) MC_RECEIVE_AND_WAIT (*primary_rc*= AP_OK) (*what_rcvd* = AP_CONFIRM) MC_SEND_ERROR
	<—	
(MC_CONFIRM returns, *primary_rc* = AP_PROG_ERROR_PURGING)		
		MC_PREPARE_TO_RECEIVE (*ptr_type* = AP_FLUSH)
	<—	
MC_RECEIVE_AND_WAIT (*primary_rc*= AP_OK) (*what_rcvd* = AP_SEND) MC_SEND_DATA MC_CONFIRM		
	—>	
		MC_RECEIVE_AND_WAIT (*primary_rc*= AP_OK) (*what_rcvd* = AP_DATA_COMPLETE) MC_RECEIVE_AND_WAIT (*primary_rc*= AP_OK) (*what_rcvd* = AP_CONFIRM) MC_CONFIRMED
	<—	
MC_DEALLOCATE (*dealloc_type* = AP_SYNC_LEVEL)		
	—>	
		MC_RECEIVE_AND_WAIT (*primary_rc* = AP_OK) (*what_rcvd* = AP_CONFIRM_DEALLOCATE) MC_CONFIRMED
	<—	
TP_ENDED		TP_ENDED

Establishing the Synchronization Level

The *sync_level* parameter of the MC_ALLOCATE verb determines the synchronization level of the conversation. Possible values for synchronization levels are:

- AP_CONFIRM_SYNC_LEVEL under which the TPs can request confirmation of receipt of data and respond to such requests
- AP_NONE under which confirmation processing does not occur

UNIX

A third level, AP_SYNCPT (Syncpoint) can also be used, but requires additional software. For more information, see "Syncpoint Support" on page 23.

Sending a Confirmation Request

The MC_CONFIRM verb has two effects:

- To flush the local LU's send buffer, which sends any data contained in the buffer to the partner TP
- To send a confirmation request, which the partner TP receives through the *what_rcvd* parameter of a receive verb

The MC_CONFIRM verb does not complete until confirmation (or an indication that an error was detected) is received from the partner TP.

Receiving Data and a Confirmation Request

The *what_rcvd* parameter of the MC_RECEIVE_AND_WAIT verb indicates the following:

- Status of the data received is complete or incomplete
- Future processing expected of the local TP (for example, a confirmation request or an indication that it should begin to send data)

When the invoked TP finishes receiving the complete data record (*what_rcvd* = AP_DATA_COMPLETE), it issues the MC_RECEIVE_AND_WAIT verb again and receives a confirmation request (*what_rcvd* = AP_CONFIRM).

Responding to the Confirmation Request

The invoked TP normally issues the MC_CONFIRMED verb to confirm receipt of data; this frees the invoking TP to resume processing.

If the invoked TP has detected an error in the received data, it can instead issue the MC_SEND_ERROR verb to indicate this error condition.

Deallocating the Conversation

The MC_DEALLOCATE verb sends a confirmation request with the data when both of the following conditions are true:

- The conversation's synchronization level (established by the *sync_level* parameter of the MC_ALLOCATE verb) is AP_CONFIRM_SYNC_LEVEL.

- The *dealloc_type* parameter of the MC_DEALLOCATE verb is set to AP_SYNC_LEVEL.

The *what_rcvd* parameter of the final MC_RECEIVE_AND_WAIT verb issued by the invoked TP contains AP_CONFIRM_DEALLOCATE, indicating that a confirmation of receipt of data is required before APPC will deallocate the conversation. The invoking TP waits for this confirmation until the invoked TP issues the MC_CONFIRMED verb to indicate that data was received successfully (or it could instead issue the MC_SEND_ERROR verb to indicate that data was not received successfully).

Sending and Receiving Status with Data (half-duplex)

In Table 3 on page 7, the invoking TP used the MC_SEND_DATA verb to send data and then the MC_CONFIRM verb to request confirmation from the invoked TP. It is possible to use a parameter on the [MC_]SEND_DATA verb to indicate that APPC should also perform the function of the [MC_]CONFIRM verb (or [MC_]DEALLOCATE, [MC_]FLUSH, or [MC_]PREPARE_TO_RECEIVE) after sending the data, instead of having to issue two separate verbs.

Similarly, the invoked TP in Table 3 on page 7 issued the MC_RECEIVE_AND_WAIT verb twice, first to receive the data and then to receive the status information that the invoking TP requested confirmation. It is possible to use a parameter on any of the [MC_]RECEIVE verbs to indicate that APPC should return status information about the same receive verb as the data, instead of having to issue two separate receive verbs.

Table 4 shows the use of the "send type" parameter on MC_SEND_DATA to perform the function of the MC_CONFIRM verb, and the "return status with data" parameter on MC_RECEIVE_AND_WAIT to receive status information with data. All *primary_rc* values can be assumed to be AP_OK unless shown otherwise.

Table 4. Receiving Status Information with Data

Invoking TP	Flow	Invoked TP
TP_STARTED MC_ALLOCATE MC_SEND_DATA (*type* = AP_SEND_DATA_CONFIRM)		
	—>	
		RECEIVE_ALLOCATE MC_RECEIVE_AND_WAIT (*rtn_status* = NO) (*what_rcvd*= AP_DATA_COMPLETE) MC_RECEIVE_AND_WAIT (*rtn_status* = NO) (*what_rcvd* = AP_CONFIRM_WHAT_RECEIVED) MC_CONFIRMED
	<—	
MC_SEND_DATA (*type* = AP_NONE) MC_CONFIRM		
	—>	
		MC_RECEIVE_AND_WAIT (*rtn_status* = YES) (*what_rcvd* = AP_DATA_COMPLETE_CONFIRM) MC_CONFIRMED
	<—	

Table 4. Receiving Status Information with Data (continued)

Invoking TP	Flow	Invoked TP
MC_DEALLOCATE TP_ENDED		
	—>	
		MC_RECEIVE_AND_WAIT (*primary_rc* = AP_DEALLOC_NORMAL) TP_ENDED

Sending Status Information with Data

In Table 4 on page 9, the first MC_SEND_DATA verb issued by the invoking TP has a send type of AP_SEND_DATA_CONFIRM. This indicates that APPC should perform the function of the MC_CONFIRM verb after sending the data. Instead of CONFIRM, the [MC_]SEND_DATA verb can also perform the function of [MC_]FLUSH, [MC_]DEALLOCATE, or [MC_]PREPARE_TO_RECEIVE, or can specify (as for the second MC_SEND_DATA verb in the example) that no additional function is to be performed.

Receiving Status Information with Data

In Table 4 on page 9, the third MC_RECEIVE_AND_WAIT verb issued by the invoked TP has a *rtn_status* parameter of AP_YES. This indicates that, if data followed by status information is available, APPC can return the status information about this verb in addition to the data. The verb returns with a *what_rcvd* parameter of AP_DATA_COMPLETE_CONFIRM, which indicates that the invoking TP sent data and then requested confirmation. The first two MC_RECEIVE_AND_WAIT verbs have the *rtn_status* parameter set to AP_NO, so APPC does not return status information with the data; the first verb receives the data, and the second receives the status information.

Conversation States (half-duplex)

In a half-duplex conversation, APPC operates as a half-duplex process, which means that only one of the two TPs is permitted to send data at any time. In general, one TP will send a certain amount of data, and then do one of the following:

- Ask the other TP to confirm receipt of the data
- Allow the other TP to send

At any one time, the TP regards the conversation as being in a particular conversation state; the conversation state governs which APPC verbs can be issued by a TP at a particular time. For example, a TP cannot issue the MC_SEND_DATA verb if the conversation is not in Send or Send-Pending state. For more information about the APPC verbs that can be issued in each state, see Appendix C, "APPC State Changes," on page 283.

Following is a list of possible conversation states:

Confirm
The TP has received a request for confirmation of receipt of data; it must respond positively or send error information to the partner TP.

Confirm-Deallocate
The TP has received a request for confirmation and must respond

positively or send error information. If the TP responds positively, the partner TP deallocates the conversation.

Confirm-Send
The TP has received a request for confirmation; it must respond positively or send error information. After responding, the TP can begin to send data.

Pending-Post
The TP is receiving data asynchronously. The TP can perform other processing not related to this conversation. When the TP finishes receiving data, the state is usually Receive.

Receive
The TP can receive application data and status information from the partner TP. When the conversation is in Receive state, the TP can also send error information and request permission to send data.

Reset The conversation has not started or has been terminated.

Send The TP can send data to the partner TP and request confirmation. When the conversation is in Send state, the TP can also begin to receive data, which causes the state to change to Receive.

Send-Pending
The TP has received data together with a SEND indication from the partner TP. This state is similar to Send state, except that the TP can use an extra parameter on the [MC_]SEND_ERROR verb to indicate the source of a detected error.

The TP's View of the Conversation

It is the conversation rather than the TP that is in a particular state. A TP may be conducting several conversations, each of which is in a different state. If a conversation is said to be in Send state, this always means that from the viewpoint of the local TP, the conversation is in Send state. To the partner TP, the conversation is in another state (such as Receive).

State Changes

A change in the conversation state occurs on the completion of an APPC verb. The state change can result from any of the following:

- A verb issued by the local TP (for example, issuing RECEIVE_AND_WAIT in Send state changes the conversation state to Receive)
- A verb issued by the partner TP (for example, if the partner TP issues the CONFIRM verb, the local TP receives indication of this on one of the RECEIVE verbs; at this point the conversation changes from Receive to Confirm state)
- An error condition

The new state of the conversation generally depends on the primary return code of the completed APPC verb. For more information, see the individual verb descriptions in Chapter 3, “APPC Control Verbs,” on page 63 and Chapter 4, “APPC Conversation Verbs,” on page 91, or see Appendix C, “APPC State Changes,” on page 283.

State Checks

A state check (state error) occurs when a TP issues an APPC verb, and the conversation is not in the appropriate state. For instance, a state check would occur if a TP issued the MC_SEND_DATA verb while the conversation was in Receive state. When a state check occurs, APPC does not execute the verb; it returns

state-check information through primary and secondary return codes. For more information about the state check error codes that can be returned for each verb, see the individual verb descriptions in Chapter 3, "APPC Control Verbs," on page 63 and Chapter 4, "APPC Conversation Verbs," on page 91.

Changing Conversation States (half-duplex)

In Table 5, the conversation states appear in the left and right margins. This table shows how APPC verbs can change the state of the conversation from Send to Receive and from Receive to Send.

Table 5. Using APPC Verbs to Change Conversation States

State	Invoking TP	Flow	Invoked TP	State
	TP_STARTED			
Reset				
	MC_ALLOCATE *(sync_level* = AP_CONFIRM_SYNC_LEVEL)			
Send				
	MC_SEND_DATA MC_PREPARE_TO_RECEIVE (*ptr_type* = AP_SYNC_LEVEL)			
		—>		
				Reset
			RECEIVE_ALLOCATE	
				Receive
			MC_RECEIVE_AND_WAIT (*primary_rc* = AP_OK) (*what_rcvd* = AP_DATA_COMPLETE) MC_RECEIVE_AND_WAIT (*primary_rc* = AP_OK) (*what_rcvd*= AP_CONFIRM_SEND)	
				Confirm-Send
			MC_CONFIRMED	
				Send
		<—		
	(MC_PREPARE_TO_RECEIVE returns, *primary_rc* = AP_OK)			
Receive				
			MC_SEND_DATA MC_CONFIRM	
		<—		
	MC_RECEIVE_AND_WAIT (*primary_rc* = AP_OK) (*what_rcvd*= AP_DATA_COMPLETE) MC_RECEIVE_AND_WAIT (*primary_rc* = AP_OK) (*what_rcvd*= AP_CONFIRM)			
Confirm				
	MC_REQUEST_TO_SEND MC_CONFIRMED			
Receive				
		—>		

Table 5. Using APPC Verbs to Change Conversation States (continued)

State	Invoking TP	Flow	Invoked TP	State
			(MC_CONFIRM returns, *primary_rc* = AP_OK, *rts_rcvd* = AP_YES)	
				Send
			MC_PREPARE_TO_RECEIVE (*ptr_type* = AP_SYNC_LEVEL)	
		<—		
	MC_RECEIVE_AND_WAIT (*primary_rc*= AP_OK) (*what_rcvd* = AP_CONFIRM_SEND)			
Confirm-Send				
	MC_CONFIRMED			
		—>		
			(MC_PREPARE_TO_RECEIVE returns, *primary_rc* = AP_OK)	
				Receive
Send				
	MC_SEND_DATA MC_DEALLOCATE (*dealloc_type* = AP_SYNC_LEVEL)			
		—>		
			MC_RECEIVE_AND_WAIT (*primary_rc* = AP_OK) (*what_rcvd*= AP_DATA_COMPLETE) MC_RECEIVE_AND_WAIT (*primary_rc*= AP_OK) (*what_rcvd* =AP_CONFIRM_DEALLOCATE)	
				Confirm-Deallocate
			MC_CONFIRMED	
		<—		
			(MC_DEALLOCATE returns, *primary_rc*= AP_OK)	
Reset				Reset
	TP_ENDED		TP_ENDED	

Initial States

Before the conversation is allocated, the state is Reset for both TPs. After the conversation is allocated, the initial state is Send for the invoking TP and Receive for the invoked TP.

Changing to Receive State

The MC_PREPARE_TO_RECEIVE verb enables a TP to change the conversation from Send to Receive state. This verb does the following:

- Flushes the local LU's send buffer.

- Sends the AP_CONFIRM_SEND indicator to the partner TP through the *what_rcvd* parameter of a receive verb. This indicator tells the partner TP that an MC_CONFIRMED response is expected before the partner TP can begin to send data.
- Performs confirmation processing because the following conditions are true:
  - The synchronization level of the conversation is set to AP_CONFIRM_SYNC_LEVEL.
  - The parameter *ptr_type* is set to AP_SYNC_LEVEL.

Issuing the MC_RECEIVE_AND_WAIT verb while the conversation is in Send state also flushes the LU's send buffer and changes the conversation state to Receive. Changing the conversation state in this manner does not support confirmation processing.

Changing to Send State

The MC_REQUEST_TO_SEND verb informs the partner TP (for which the conversation is in Send state) that the local TP (for which the conversation is in Receive state) wants to send data.

This request is communicated to the partner TP through the *rts_rcvd* parameter of the MC_CONFIRM verb. (The *rts_rcvd* parameter is also returned to MC_SEND_DATA and other verbs.)

When the partner TP issues the MC_PREPARE_TO_RECEIVE verb, the conversation state changes to Receive for the partner TP, making it possible for the local TP to send data.

Issuing the MC_REQUEST_TO_SEND verb does not change the state of the conversation. Upon receiving a request to send, the partner TP is not required to change the conversation state; it can ignore the request.

Full-Duplex Conversations

Table 6 shows an example of a full-duplex conversation, showing the APPC verbs used to start a conversation, exchange data, and end the conversation. The arrow indicates the flow of data. All *primary_rc* values are AP_OK unless shown otherwise.

Table 6. A Full-Duplex Conversation

Invoking TP	Flow	Invoked TP
TP_STARTED MC_ALLOCATE MC_SEND_DATA		
		RECEIVE_ALLOCATE
MC_SEND_DATA	—>	
	<—	MC_SEND_DATA
MC_RECEIVE_AND_WAIT (*what_rcvd* = AP_DATA_COMPLETE) MC_DEALLOCATE		
		MC_RECEIVE_AND_WAIT (*what_rcvd* = AP_DATA_COMPLETE) MC_RECEIVE_AND_WAIT (*primary_rc* = AP_DEALLOC_NORMAL)
	<—	MC_SEND_DATA MC_DEALLOCATE TP_ENDED

Table 6. A Full-Duplex Conversation (continued)

Invoking TP	Flow	Invoked TP
MC_RECEIVE_AND_WAIT (*what_rcvd* = AP_DATA_COMPLETE)		
MC_RECEIVE_AND_WAIT (*what_rcvd* = AP_DEALLOC_NORMAL)		
TP_ENDED		

The characters MC_ at the beginning of many of the verbs stand for mapped conversation. Parameters and results of APPC verbs are in parentheses.

Starting a Conversation

To start a conversation, the invoking TP issues the following verbs:

- TP_STARTED, which identifies the application to APPC as an invoking TP
- MC_ALLOCATE, which requests that APPC establish a conversation between the invoking TP and the invoked TP. The parameters on the MC_ALLOCATE verb specify that the conversation will be full-duplex.

The invoked TP issues the RECEIVE_ALLOCATE verb, which informs APPC that the invoked TP is ready to begin a conversation. Communications Server associates the RECEIVE_ALLOCATE verb with the MC_ALLOCATE verb issued by the invoking TP in order to establish the conversation between the two TPs. The returned parameters on the RECEIVE_ALLOCATE verb specify that the conversation will be full-duplex.

Note: After a full-duplex conversation has been started, the TP must set an additional option in the *opext* parameter of all verbs issued in this conversation to operate in full-duplex mode. See the individual verb descriptions in Chapter 4, "APPC Conversation Verbs," on page 91 for details.

Sending Data

The MC_SEND_DATA verb supplies application data to be transmitted to the partner TP. This data is held in the local LU's send buffer; it is not transmitted to the partner TP until one of the following events occurs:

- The send buffer fills up.
- The TP issues a verb that forces APPC to flush the buffer (send data to the partner TP).

In addition to the data, the send buffer also contains the MC_ALLOCATE request (which precedes the data). In this example, the invoking TP's second MC_SEND_DATA verb fills the buffer, forcing the MC_ALLOCATE request and the data to be transmitted to the partner TP. Other verbs that flush the buffer are MC_DEALLOCATE and MC_FLUSH.

Because this is a full-duplex conversation, both TPs can send data at the same time. In the example, the invoked TP sends data before it has received any data sent by the invoking TP.

Receiving Data

The MC_RECEIVE_AND_WAIT verb receives data from the partner TP. If no data is currently available, the local TP waits for it to arrive.

As well as receiving data, the verb may receive a status indicator from the partner TP (such as an indication that the conversation is ending). For more information about how the TP handles these indicators, see "Ending a Conversation."

In the example, the invoked TP issues the first MC_RECEIVE_AND_WAIT verb to receive data, and receives the complete data record (*what_rcvd*= `AP_DATA_COMPLETE`).

The MC_RECEIVE_IMMEDIATE verb performs the same function as the MC_RECEIVE_AND_WAIT verb, except that it does not wait if data is not currently available from the partner TP. Instead, it returns a no-data-available response to the calling TP.

Ending a Conversation

To end a conversation, one of the TPs issues the MC_DEALLOCATE verb, indicating that it has no more data to send. The other TP receives the return code `AP_DEALLOC_NORMAL` on a subsequent receive verb, indicating that the conversation was deallocated.

In a half-duplex conversation, the MC_DEALLOCATE verb causes APPC to deallocate the conversation between the two TPs, so that the other TP cannot continue the conversation after it has received the `AP_DEALLOC_NORMAL` return code. However, in a full-duplex conversation, the other TP may still have data to send, or may already have sent data that the first TP has not yet received. For this reason, the conversation does not end at this point; instead, the first TP operates in receive-only mode, so that it continues to issue receive verbs (but cannot send any further data).

When the second TP receives the `AP_DEALLOC_NORMAL` return code, it now operates in send-only mode. It cannot issue any more receive verbs (because there will be no more data to receive), but it can continue to send data. In the example, the invoked TP issues another MC_SEND_DATA verb before deallocating the conversation.

After both TPs have deallocated this conversation, each TP either issues another [MC_]ALLOCATE or RECEIVE_ALLOCATE verb to start another conversation (with this or another partner TP), or issues the TP_ENDED verb.

A TP can participate in multiple conversations simultaneously. In this case, the TP issues the TP_ENDED verb when all conversations have been deallocated.

Conversation States

A TP regards a full-duplex conversation as being in a particular conversation state, in the same way as for a half-duplex conversation. However, the possible conversation states in a full-duplex conversation are different from those in a half-duplex conversation, as follows. For more information about the APPC verbs that can be issued in each state, see Appendix C, "APPC State Changes," on page 283.

Following is a list of possible conversation states:

Send-Receive
The TP can send data or error information, and can receive application data and status information from the partner TP.

Receive-Only
The local TP has deallocated the conversation. It can continue to receive data and status information from the partner TP, but cannot send any further data.

Send-Only
The remote TP has deallocated the conversation. The local TP can continue to send data to the partner TP, but will not receive any further data and so is not permitted to issue any further receive verbs.

Reset The conversation has not started or has been terminated.

Half-Duplex Verbs Not Permitted in Full-Duplex Conversations

The following verbs apply only in half-duplex conversations, and are not required in full-duplex conversations. Any of these verbs issued in a full-duplex conversation will receive an error return code.

- [MC_]CONFIRM
- [MC_]CONFIRMED
- [MC_]PREPARE_TO_RECEIVE
- [MC_]RECEIVE_AND_POST
- [MC_]REQUEST_TO_SEND
- [MC_]TEST_RTS
- [MC_]TEST_RTS_AND_POST

Sending and Receiving Expedited Data

In addition to the normal data sent using [MC_]SEND_DATA and received using receive verbs, APPC TPs may also send and receive SNA expedited data. This data is handled separately by the SNA network, and may arrive at the destination before normal data. A TP sends expedited data using the [MC_]SEND_EXPEDITED_DATA verb, and receives it using the [MC_]RECEIVE_EXPEDITED_DATA verb; expedited data will never be returned on the standard receive verbs.

Not all APPC implementations support expedited data. If the remote TP is using an APPC implementation that does not support expedited data, the local TP cannot send or receive it.

Because expedited data flows separately from normal data, a TP can issue [MC_]SEND_EXPEDITED_DATA or [MC_]RECEIVE_EXPEDITED_DATA in any conversation state except Reset state. These verbs do not cause any state change.

Synchronous and Asynchronous APPC Calls

UNIX

On AIX / Linux systems, Communications Server provides two alternative entry points to the APPC library:

- Synchronous entry point, APPC. If the application uses this entry point, Communications Server does not return control to the application until verb processing has finished.
- Asynchronous entry point, APPC_Async. If the application uses this entry point, Communications Server returns control to the application immediately. When verb processing has finished, Communications Server uses an application-supplied callback routine to return the results of the verb processing to the application.

WINDOWS

On Windows systems, the Remote API provides three alternative entry points to the APPC library:

- Synchronous entry point, APPC. If the application uses this entry point, the Remote API does not return control to the application until verb processing has finished.
- Asynchronous entry point, WinAsyncAPPC. If the application uses this entry point, the Remote API returns control to the application immediately. When verb processing has finished, the Remote API indicates this by posting a message to the application's window procedure.
- Asynchronous entry point, WinAsyncAPPCEx. When verb processing has finished, the Remote API indicates this by signaling an event handle provided by the application.

For more information about these entry points, see Chapter 2, “Writing Transaction Programs,” on page 25.

Using the asynchronous entry point enables an application to continue with other processing while waiting for a verb to complete. The application may issue verbs on other APPC conversations, or issue verbs to start new conversations, or it may perform other processing not related to APPC. However, other verbs issued on the same conversation may be queued and not processed until the outstanding verb has completed; for more information, see “Non-Blocking Operation” on page 21 below.

The only exception to this is when the [MC_]RECEIVE_AND_POST verb is outstanding. In these cases, the application can issue a limited range of verbs on the same conversation. For more information, see the following section.

Receiving Data Asynchronously

The APPC verbs MC_RECEIVE_AND_POST and RECEIVE_AND_POST enable a TP to receive data asynchronously, without regard for other events occurring within the program. Therefore, the program can perform other tasks while receiving data.

UNIX

For AIX / Linux systems, the parameters for [MC_]RECEIVE_AND_POST include the address of a callback routine, which APPC uses to inform the TP when data has been received. This callback routine is independent of the callback routine supplied on the asynchronous APPC entry point. [MC_]RECEIVE_AND_POST may be issued using either the synchronous or the asynchronous entry point; APPC uses the callback routine supplied in the verb parameters to return received data to the TP and uses the callback routine supplied to the asynchronous entry point only if a null address is supplied in the VCB.

WINDOWS

On Windows systems, verb completion is by signaling an event handle provided by the application. The *sema* parameter contains the event handle (obtained by calling either the Windows CreateEvent or OpenEvent functions), which APPC uses to inform the TP when data has been received.

The operation of [MC_]RECEIVE_AND_POST is similar to the operation of [MC_]RECEIVE_AND_WAIT issued using the asynchronous entry point; control returns to the application immediately, and the requested data is subsequently returned on the callback routine. The main difference is that issuing [MC_]RECEIVE_AND_POST puts the conversation into a defined state, Pending-Post state, in which the TP can issue a limited range of APPC verbs on this conversation while waiting for the callback routine to be called. The verbs that can be issued in Pending-Post state are:

- GET_TYPE
- CANCEL_CONVERSATION
- [MC_]DEALLOCATE with a deallocate type of AP_ABEND, AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER
- [MC_]GET_ATTRIBUTES
- [MC_]RECEIVE_EXPEDITED_DATA
- [MC_]REQUEST_TO_SEND
- [MC_]SEND_ERROR
- [MC_]SEND_EXPEDITED_DATA
- [MC_]TEST_RTS
- TP_ENDED

If the application issues [MC_]RECEIVE_AND_WAIT (or any other APPC verb) using the asynchronous entry point, it must not issue any other APPC verbs on this conversation until the callback routine has been called. The only exception is that it can issue CANCEL_CONVERSATION to cancel the outstanding receive operation and end the conversation.

In Table 7, the invoked TP receives data asynchronously.

Table 7. Receiving Data Asynchronously

State	Invoking TP	Flow	Invoked TP	State
	TP_STARTED			
Reset				

Table 7. Receiving Data Asynchronously (continued)

State	Invoking TP	Flow	Invoked TP	State
	MC_ALLOCATE			
Send				
	MC_FLUSH			
		—>		
				Reset
			RECEIVE_ALLOCATE	
				Receive
			MC_RECEIVE_AND_POST (*primary_rc*=AP_OK)	
				Pending-Post
			(TP performs other tasks or issues verbs such as Request to Send or Get Attributes. Most other APPC verbs cannot be used in this conversation state (see Appendix C, "APPC State Changes," on page 283 for information about permitted verbs).	
	MC_SEND_DATA MC_DEALLOCATE			
Reset				
	TP_ENDED			
		—>		
			Data is received. APPC calls the supplied callback routine. MC_RECEIVE_AND_POST returns *primary_rc*=AP_OK, *what_rcvd*=AP_DATA_COMPLETE	
				Receive
			(TP checks that the callback routine has been called.) MC_RECEIVE_AND_WAIT (*primary_rc*=AP_DEALLOC_NORMAL)	
				Reset
			TP_ENDED	

In Table 7 on page 19, the invoked TP follows these steps to receive data asynchronously:

1. Issues the MC_RECEIVE_AND_POST verb. One of the parameters is the address of the callback routine which APPC calls (on AIX / Linux) or event handle which APPC signals (on Windows) when the data is received.
2. Verifies that the *primary_rc* (primary return code) is AP_OK, which indicates that the TP has begun to receive data asynchronously.
3. Performs tasks not related to this conversation while receiving data asynchronously. Most APPC verbs are not valid in this conversation state.
4. Waits for the callback routine to be called (on AIX / Linux), or event handle to be signaled (on Windows), which indicates that the TP has finished receiving data asynchronously.
5. Verifies the *primary_rc* of the MC_RECEIVE_AND_POST verb again. The second *primary_rc* indicates whether the data was received without error.

6. Verifies that the *what_rcvd* parameter of the MC_RECEIVE_AND_POST verb is AP_DATA_COMPLETE.
7. Issues the MC_RECEIVE_AND_WAIT verb to receive the deallocation indicator.

Note: The [MC_]RECEIVE_AND_POST verb returns the *primary_rc* and *secondary_rc* parameters twice; first after issuing the verb, to indicate whether or not the verb has successfully begun to wait for data, and second after the data has been received.

After the invoked TP issues the MC_RECEIVE_AND_POST verb and gets a *primary_rc* of AP_OK, the conversation changes to Pending-Post state.

When the TP has finished receiving data asynchronously and APPC calls the supplied callback routine (on AIX / Linux) or signals the supplied event handle (on Windows), the conversation changes to Receive state because the *what_rcvd* parameter contains AP_DATA_COMPLETE.

Non-Blocking Operation

Communications Server supports queue-level non-blocking operation for APPC conversation verbs, so that a TP can issue multiple verbs on the same conversation without having to wait for each verb to complete. This is normally used in conjunction with the asynchronous APPC entry point, which allows the TP to continue operation even though processing for a previous verb has not yet completed, but it may also be used with the synchronous entry point in a multi-threaded TP that issues APPC verbs from more than one thread. To issue a verb in non-blocking mode, the TP sets an option in the *opext* parameter of the verb, as described in Chapter 4, "APPC Conversation Verbs," on page 91.

For each TP and conversation, APPC provides a number of queues on which verbs can be held while waiting to be processed. Each queue handles a different subset of the valid APPC verbs, so that each verb is associated with a different queue.

- If the TP issues a verb and there are no verbs already being processed for the appropriate queue, the verb is processed immediately.
- If the TP issues a non-blocking verb and another verb is already being processed for the appropriate queue, the verb is added to the queue (behind any other verbs already waiting on the appropriate queue). It will be processed after the verbs already queued have completed (except for the Allocate queue, as described below).
- If the TP issues a blocking verb and another verb is already being processed for the appropriate queue, the verb is rejected with an error return code.

The queues available to the TP, and the verbs that each queue handles, are as follows.

Allocate queue

For each active TP, there is a single queue that handles the following verbs:

- [MC_]ALLOCATE
- [MC_]SEND_CONVERSATION

Two or more verbs on this queue can be processed at the same time, so they are not guaranteed to complete in the same order in which they were issued.

Send-Receive queue (half-duplex conversations only)
For each active half-duplex conversation, there is a single queue that handles the following verbs:

- [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 [MC_]RECEIVE_AND_POST verb is not held on this queue. This verb cannot be issued, in either blocking or non-blocking mode, if any of the Receive verbs is already being processed for the conversation.

Send queue (full-duplex conversations only)
For each active full-duplex conversation, there is a single queue that handles the following verbs:

- [MC_]DEALLOCATE
- [MC_]FLUSH
- [MC_]SEND_DATA
- [MC_]SEND_ERROR

Receive queue (full-duplex conversations only)
For each active full-duplex conversation, there is a single queue that handles the following verbs:

- [MC_]RECEIVE_AND_WAIT
- [MC_]RECEIVE_IMMEDIATE

Send-Expedited queue
For each active conversation (either full-duplex or half-duplex), there is a single queue that handles the following verbs:

- [MC_]REQUEST_TO_SEND (half-duplex conversations only)
- [MC_]SEND_EXPEDITED_DATA

Receive-Expedited queue
For each active conversation (either full-duplex or half-duplex), there is a single queue that handles the following verbs:

- [MC_]RECEIVE_EXPEDITED_DATA

The CANCEL_CONVERSATION verb is not held on any queue. Issuing this verb cancels any other verbs that are already queued for this conversation.

The following conversation verbs are not associated with any queue and so can be issued at any time regardless of the verbs already queued. The non-blocking mode option in the *opext* parameter of the verb is ignored.

- GET_TYPE
- [MC_]GET_ATTRIBUTES
- [MC_]TEST_RTS
- [MC_]TEST_RTS_AND_POST

APPC control verbs are always issued in blocking mode.

Syncpoint Support

UNIX

The Communications Server APPC API provides support for sessions and conversations that use LU 6.2 Syncpoint protocols. This means that it can be used in conjunction with transaction monitors that require Syncpoint Level 2 support. It does not itself provide the components necessary for a full Syncpoint implementation, but provides the underlying support for a vendor-supplied implementation. The vendor must provide the following components:

- Syncpoint Manager (SPM)
- Conversation-Protected Resource Manager (C-PRM)
- Resynchronization TP

This manual does not attempt to explain Syncpoint functions; it describes only the support that Communications Server provides to enable them to be implemented. If you are developing a Syncpoint Manager for use with Communications Server APPC, you should already be familiar with Syncpoint concepts; refer to the IBM LU 6.2 manuals for more information if necessary.

Some parameter and return code values in this manual are marked “only used by TPs that support Syncpoint processing” or “only used if the conversation's *sync_level* is `AP_SYNCPT`”. If you are writing APPC applications that do not use Syncpoint functions, do not attempt to use these parameters. In most cases, the Syncpoint Manager is responsible for converting between these values and the appropriate Syncpoint functions, as noted in the parameter descriptions.

If you are writing applications to work with a Syncpoint implementation provided by your Communications Server supplier or by another vendor, the vendor should provide you with the additional information necessary to use this implementation.

APPC and CPI-C

The Common Programming Interface for Communications (CPI-C) application programming interface, another Communications Server API, provides many of the functions of APPC but with a different style of interface.

Where an APPC application sets parameters in a verb control block and then calls a single entry point to APPC with the address of the block, a CPI-C program calls a different entry point for each verb and passes the required information as parameters on the call.

Although the programming interfaces for APPC and CPI-C are different, the actual data transmitted between programs is the same. This means that a CPI-C application can communicate with an APPC TP, just as two APPC TPs or two CPI-C applications can communicate with each other. The APPC TP does not need to know whether its partner is an APPC TP or a CPI-C application.

The only restriction on an APPC TP for communications with a CPI-C application is that it must not send Program Initialization Parameters (PIP data) when allocating the conversation, because CPI-C does not support receiving PIP data. For more information about PIP data, see the description of the [MC]ALLOCATE verb in Chapter 4, "APPC Conversation Verbs," on page 91.

TP Server API

UNIX

The TP server verbs are an extension to the APPC API to allow applications to participate in starting TPs in response to allocation requests (Attaches).

Communications Server provides a default mechanism to start TPs automatically. TPs that can be automatically started are configured in the **sna_tps** file, as described in the *IBM Communications Server for Data Center Deployment on AIX or Linux Administration Guide*.

Some applications, such as transaction monitors, need more control over starting TPs than this default mechanism supplies (such as access to the allocation request). The TP server extensions provide the level of support needed by such applications. For more information about TP servers, see "Writing TP Servers" on page 59.

Chapter 2. Writing Transaction Programs

This chapter contains information about the following topics and will help you write transaction programs (TPs):

- Categories of APPC verbs
- APPC verb summary
- APPC entry points

UNIX

- AIX / Linux considerations

WINDOWS

- Windows considerations

- Configuration information
- Conversation security
- Starting TPs
- LU-to-LU sessions
- Basic conversations

UNIX

- Writing TP servers

- Writing portable TPs

Categories of APPC Verbs

APPC verbs fall into the following categories:

- Control verbs, described in Chapter 3, "APPC Control Verbs," on page 63
- Conversation verbs, described in Chapter 4, "APPC Conversation Verbs," on page 91

Control Verbs

Control verbs start and end TPs and obtain information about the properties of TPs:

- TP_STARTED
- TP_ENDED
- RECEIVE_ALLOCATE
- GET_TP_PROPERTIES

UNIX

- SET_TP_PROPERTIES

- GET_LU_STATUS

Conversation Verbs

Conversation verbs enable TPs to allocate a conversation, send and receive data, change conversation states, and deallocate a conversation.

The following verbs can be issued in either a basic or mapped conversation:

- GET_TYPE
- CANCEL_CONVERSATION

Most conversation verbs fall into two groups:

- Mapped-conversation verbs, which a TP can issue only in a mapped conversation
- Basic-conversation verbs, which a TP can issue only in a basic conversation

Conversation verbs are grouped by type, mapped or basic, as shown in Table 8:

Table 8. Mapped and Basic Conversation Verbs

Mapped-Conversation Verbs	Basic-Conversation Verbs
MC_ALLOCATE	ALLOCATE
MC_CONFIRM	CONFIRM
MC_CONFIRMED	CONFIRMED
MC_DEALLOCATE	DEALLOCATE
MC_FLUSH	FLUSH
MC_GET_ATTRIBUTES	GET_ATTRIBUTES
MC_PREPARE_TO_RECEIVE	PREPARE_TO_RECEIVE
MC_RECEIVE_AND_POST	RECEIVE_AND_POST
MC_RECEIVE_AND_WAIT	RECEIVE_AND_WAIT
MC_RECEIVE_IMMEDIATE	RECEIVE_IMMEDIATE
MC_RECEIVE_EXPEDITED_DATA	RECEIVE_EXPEDITED_DATA
MC_REQUEST_TO_SEND	REQUEST_TO_SEND
MC_SEND_CONVERSATION	SEND_CONVERSATION
MC_SEND_DATA	SEND_DATA
MC_SEND_ERROR	SEND_ERROR
MC_SEND_EXPEDITED_DATA	SEND_EXPEDITED_DATA
MC_TEST_RTS	TEST_RTS
MC_TEST_RTS_AND_POST	TEST_RTS_AND_POST

Mapped and basic verbs have the same functions in their respective types of conversation, but may have slightly different parameters and return codes.

TP Server Verbs

UNIX

TP server verbs allow applications to start TPs in response to requests from Communications Server:

REGISTER_TP_SERVER
UNREGISTER_TP_SERVER

REGISTER_TP
UNREGISTER_TP
QUERY_ATTACH
ACCEPT_ATTACH
REJECT_ATTACH
ABORT_ATTACH

Note: TP server verbs, as described in Chapter 5, "TP Server Verbs," on page 245, must be issued using the asynchronous entry point `APPC_Async`, and not the synchronous entry point `APPC`.

APPC Verb Summary

This section briefly describes each APPC verb. The APPC verbs are grouped by function. (The total functionality of a verb may be broader than this summary indicates. For a detailed explanation of a particular verb, see Chapter 3, "APPC Control Verbs," on page 63, or Chapter 4, "APPC Conversation Verbs," on page 91.)

Starting a Conversation

The following verbs are used to start a conversation between two TPs. For more information, see "Starting TPs" on page 54.

TP_STARTED

This verb is issued by the invoking TP. It notifies APPC that the invoking TP is starting. Upon successful execution, this verb returns a TP identifier (*tp_id*) for the invoking TP.

UNIX

SET_TP_PROPERTIES

This verb is used by a TP to set properties relating to the local TP, which are then used when allocating new conversations. This enables the TP to specify the following:

- Logical Unit of Work (a transaction between APPC TPs to accomplish a particular task) with which a conversation is associated
- User ID for use when allocating a new conversation and indicating that the conversation security has already been verified

MC_ALLOCATE or ALLOCATE

This verb is issued by the invoking TP. It allocates a session between the local LU and a remote LU and establishes a conversation between the invoking TP and the invoked TP.

The ALLOCATE verb can establish either a basic or mapped conversation. The MC_ALLOCATE verb can start only a mapped conversation. Either verb can start either a full-duplex or a half-duplex conversation.

Once the conversation is allocated, APPC returns a conversation identifier (*conv_id*) through this verb.

MC_SEND_CONVERSATION or SEND_CONVERSATION
This verb is issued by the invoking TP. It allocates a session between the local LU and a remote LU, establishes a conversation between the invoking TP and the invoked TP, sends a single data record on this conversation, and deallocates the conversation.

RECEIVE_ALLOCATE
This verb is issued by the invoked TP. It confirms that the invoked TP is ready to begin a conversation with the invoking TP, which issued the [MC_]ALLOCATE verb. Upon successful execution, RECEIVE_ALLOCATE returns a TP identifier (*tp_id*) for the invoked TP and the conversation identifier (*conv_id*).

Sending Data

The following verbs send data to the partner TP:

MC_SEND_DATA or SEND_DATA
This verb puts data in the local LU's send buffer for transmission to the partner TP.

The data collected in the local LU's send buffer is transmitted to the partner LU (and partner TP) when one of the following occurs:

- The send buffer fills up.
- The local TP issues a verb that flushes the local LU's send buffer, such as [MC_]FLUSH or [MC_]CONFIRM. ([MC_]CONFIRM applies to half-duplex conversations only.)

The [MC_]SEND_DATA verb can also perform the function of the [MC_]CONFIRM, [MC_]DEALLOCATE, [MC_]FLUSH, or [MC_]PREPARE_TO_RECEIVE verbs. ([MC_]CONFIRM and [MC_]PREPARE_TO_RECEIVE apply to half-duplex conversations only.)

MC_SEND_EXPEDITED_DATA or SEND_EXPEDITED_DATA
This verb puts data in the local LU's expedited send buffer for transmission to the partner TP.

The data collected in the local LU's send buffer is transmitted to the partner LU (and partner TP) in the same way as for the [MC_]SEND_DATA verb. However, because the data is sent over the network as expedited data, it may arrive before data that was sent earlier using [MC_]SEND_DATA.

MC_FLUSH or FLUSH
This verb flushes the local LU's send buffer, sending the contents of the buffer to the partner LU (and TP). If the send buffer is empty, no action occurs.

MC_CONFIRM or CONFIRM (half-duplex conversation only)
This verb sends both the contents of the local LU's send buffer and a confirmation request to the partner TP.

MC_PREPARE_TO_RECEIVE or PREPARE_TO_RECEIVE (half-duplex conversation only)
This verb changes the state of the conversation from Send to Receive. Before changing the conversation state, this verb performs the equivalent of the [MC_]FLUSH or [MC_]CONFIRM verb. After this verb has successfully executed, the TP can receive data.

MC_REQUEST_TO_SEND or REQUEST_TO_SEND (half-duplex conversation only)
This verb informs the partner TP that the local TP wants to send data. If the partner TP issues the [MC_]PREPARE_TO_RECEIVE or [MC_]RECEIVE_AND_WAIT verb, the conversation state changes to Receive for the partner TP, which makes it possible for the local TP to begin to send data.

Receiving Data

The following verbs enable a TP to receive data from its partner TP:

MC_RECEIVE_AND_WAIT or RECEIVE_AND_WAIT
Issuing this verb while the conversation is in Receive state causes the local TP to receive any data that is currently available from the partner TP. If no data is available, the local TP waits for data to arrive.

Issuing this verb while the conversation is in Send state flushes the LU's send buffer and changes the conversation state to Receive. The local TP then begins to receive data.

MC_RECEIVE_AND_POST or RECEIVE_AND_POST
Issuing this verb while the conversation is in Receive state changes the conversation state to Pending_Post and causes the local TP to receive data asynchronously. This enables the local TP to proceed with processing while data is still arriving at the local LU.

Issuing this verb while the conversation is in Send state flushes the LU's send buffer and changes the conversation state to Pending_Post. The local TP then begins to receive data asynchronously.

MC_RECEIVE_IMMEDIATE or RECEIVE_IMMEDIATE
This verb receives any data that is currently available from the partner TP. If no data is available, the local TP does not wait. Unlike the other RECEIVE verbs, this verb can be issued only in Receive state and not in Send state.

MC_RECEIVE_EXPEDITED_DATA or RECEIVE_EXPEDITED_DATA
This verb receives any expedited data that is currently available from the partner TP. If no data is available, the verb can either return immediately or wait until data arrives.

Confirming Receipt of Data or Reporting Errors

The following verbs confirm receipt of data or report an error:

MC_CONFIRMED or CONFIRMED
This verb replies to a confirmation request from the partner TP. It informs the partner TP that the local TP has received and processed the data without error.

MC_SEND_ERROR or SEND_ERROR
This verb notifies the partner TP that the local TP has encountered an application-level error.

Getting Information

The following verbs provide information to TPs:

MC_GET_ATTRIBUTES or GET_ATTRIBUTES
This verb is used by a TP to obtain the attributes of the conversation.

GET_TYPE
This verb is used by a TP to determine the conversation type (basic or mapped) of a particular conversation, and whether the conversation operates in full-duplex or half-duplex mode. With this information, the TP can determine the correct verbs to issue on this conversation.

UNIX

GET_LU_STATUS
This verb is used by a TP to obtain information about the number of sessions between its local LU and a specified partner LU, and whether the number of sessions has dropped to `0` (zero) at any time since the verb was last issued. This enables the TP to check whether it has lost connectivity to its partner TP (in which case it may need to resynchronize).

GET_TP_PROPERTIES
This verb is used by a TP to obtain information about the attributes of the local TP and of the Logical Unit of Work (a transaction between APPC TPs to accomplish a particular task) in which the TP is participating.

MC_TEST_RTS or TEST_RTS
This verb determines whether a REQUEST_TO_SEND notification has been received from the partner TP.

MC_TEST_RTS_AND_POST or TEST_RTS_AND_POST
This verb notifies the application asynchronously when a REQUEST_TO_SEND notification is received from the partner TP.

Ending a Conversation

Either the invoked or invoking TP can end the conversation. The following verbs end a conversation:

MC_DEALLOCATE or DEALLOCATE
This verb deallocates a conversation between two TPs. Before deallocating the conversation, this verb performs the equivalent of the [MC_]FLUSH or [MC_]CONFIRM verb.

CANCEL_CONVERSATION
This verb deallocates a conversation between two TPs. It is equivalent to MC_DEALLOCATE or DEALLOCATE with *dealloc_type* set to `AP_ABEND` or any of the `AP_ABEND_*` values, except that it can be issued while other verbs are outstanding on the same conversation. (The results of these outstanding verbs are undefined, and will not be returned to the application. For example, if you issue CANCEL_CONVERSATION while [MC_]SEND_DATA is outstanding, you cannot determine whether any of the data has been sent to the partner TP.)

TP_ENDED
This verb is issued by both the invoking and invoked TPs. It notifies APPC that the TP is ending. Issuing this verb also terminates any other conversations that may be active.

Starting a Transaction Program (TP)

UNIX

The following verbs are used to enable an application to participate in the Communications Server TP loading process.

REGISTER_TP_SERVER
This verb is used by a TP to notify Communications Server that the application is capable of automatically starting transaction programs (TPs).

REGISTER_TP
This verb is used to register with Communications Server the name of a TP, whose TP start requests (Attaches) are to be handled by the application.

QUERY_ATTACH
This verb is used by the application to determine the parameters on the request to start a TP, so the application can determine whether to start the TP.

ACCEPT_ATTACH
This verb is used to notify Communications Server that the application intends to start the TP that corresponds to this Attach.

REJECT_ATTACH
This verb is used to notify Communications Server that the application does not intend to start the TP that corresponds to this Attach.

ABORT_ATTACH
This verb is used to end the processing of the Attach by this TP server after the Attach has been accepted using an ACCEPT_ATTACH verb because the TP server or TP has encountered an error during further processing.

UNREGISTER_TP
This verb is used to notify Communications Server that the application no longer wishes to process Attaches for this previously registered TP.

UNREGISTER_TP_SERVER
This verb is used to notify Communications Server that the application does not want to receive Attach notifications for the specified TP.

APPC Entry Points: AIX or Linux Systems

UNIX

An application accesses APPC using the following entry points:

APPC Issues an APPC verb synchronously. Communications Server does not return control to the application until verb processing has finished.

APPC_Async
Issues an APPC verb asynchronously. Communications Server returns control to the application immediately, with a returned value indicating whether verb processing is still in progress or has completed. In most

cases, verb processing is still in progress when control returns to the application; Communications Server then uses an application-supplied callback routine to return the results of the verb processing. In some cases, verb processing is complete when Communications Server returns control to the application; Communications Server does not use the application's callback routine.

Note: TP server verbs, as described in Chapter 5, "TP Server Verbs," on page 245, must be issued using the asynchronous entry point APPC_Async, and not the synchronous entry point APPC.

The entry points APPC and APPC_Async are defined in the APPC header file **/usr/include/sna/appc_c.h** (for AIX) or **/opt/ibm/sna/include/appc_c.h** (for Linux).

An application that performs a single task, and can suspend while waiting for information either from Communications Server or from the remote system, need only use the APPC (synchronous) entry point.

If the application performs multiple tasks (such as communicating with more than one remote program at a time, or performing other processing in addition to APPC verbs), you may need to ensure that it does not suspend while waiting for information. In this case, the application should use the APPC_Async (asynchronous) entry point, supplying a callback routine that Communications Server can use to return information when it is available.

The following sections describe these entry points, and also describe some additional application-defined functions which the application must supply to them.

APPC Entry Point

An application uses APPC to issue an APPC verb synchronously. Communications Server does not return control to the application until verb processing has finished.

Function Call

```
void APPC       (
                 void * vcb
                );
```

For compatibility with earlier APPC implementations, Communications Server also provides the entry points APPC_C and APPC_P, which can be used in the same way as APPC.

Supplied Parameters

When the application uses the APPC entry point to issue a verb, it supplies the following parameter:

vcb Pointer to a Verb Control Block (VCB) that contains the parameters for the verb being issued. The VCB structure for each verb is described in Chapter 3, "APPC Control Verbs," on page 63 and Chapter 4, "APPC Conversation Verbs," on page 91. These structures are defined in the APPC header file **/usr/include/sna/appc_c.h** (for AIX) or **/opt/ibm/sna/include/appc_c.h** (for Linux).

Note: The APPC VCBs contain many parameters marked as "reserved"; some of these are used internally by the Communications Server software, and others are not used in this version but may be used in

future versions. Your application must not attempt to access any of these reserved parameters; instead, it must set the entire contents of the VCB to zero to ensure that all of these parameters are zero, before it sets other parameters that are used by the verb. This ensures that Communications Server will not misinterpret any of its internally-used parameters, and also that your application will continue to work with future Communications Server versions in which these parameters may be used to provide new functions.

To set the VCB contents to zero, use `memset`:

memset(vcb, 0, sizeof(vcb));

Returned Values

The function does not return a value. When the call returns, the application can examine the parameters in the VCB to determine whether the verb completed successfully.

APPC_Async Entry Point

An application uses `APPC_Async` to issue an APPC verb asynchronously. Communications Server returns control to the application immediately, with a returned value indicating whether verb processing is still in progress or has completed. In most cases, verb processing is still in progress when control returns to the application. Later, Communications Server uses an application-supplied callback routine to return the results of the verb processing. In some cases, verb processing is complete when Communications Server returns control to the application, so Communications Server does not use the application's callback routine.

Function Call

```
unsigned short APPC_Async  (
                            void *        vcb,
                            AP_CALLBACK   (*comp_proc),
                            AP_CORR       corr
                           );
typedef void (*AP_CALLBACK) (
                             void *          vcb,
                             unsigned char   tp_id[8],
                             AP_UINT32       conv_id,
                             AP_CORR corr
                            );
typedef union ap_corr  {
                        void *          corr_p;
                        AP_UINT32       corr_l;
                        AP_INT32        corr_i;
                       } AP_CORR;
```

Parameter types such as `AP_UINT32`, used in these entry points and in the APPC VCBs, are defined in the common header file **/usr/include/sna/values_c.h** (for AIX) or **/opt/ibm/sna/include/values_c.h** (for Linux), which is included by the APPC header file **/usr/include/sna/appc_c.h** (for AIX) or **/opt/ibm/sna/include/appc_c.h** (for Linux).

Supplied Parameters

When the application uses the `APPC_Async` entry point to issue a verb, it supplies the following parameters:

vcb Pointer to a Verb Control Block (VCB) that contains the parameters for the

verb being issued. The VCB structure for each verb is described in Chapter 3, "APPC Control Verbs," on page 63 and Chapter 4, "APPC Conversation Verbs," on page 91. These structures are defined in the APPC header file **appc_c.h**.

Note: The APPC VCBs contain many parameters marked as "reserved"; some of these are used internally by the Communications Server software, and others are not used in this version but may be used in future versions. Your application must not attempt to access any of these reserved parameters; instead, it must set the entire contents of the VCB to zero to ensure that all of these parameters are zero, before it sets other parameters that are used by the verb. This ensures that Communications Server will not misinterpret any of its internally-used parameters, and also that your application will continue to work with future Communications Server versions in which these parameters may be used to provide new functions.

To set the VCB contents to zero, use `memset`:

```
memset(vcb, 0, sizeof(vcb));
```

comp_proc
The callback routine that Communications Server will call when the verb completes. For more information about the requirements for a callback routine, see "Callback Routine for Asynchronous Verb Completion" on page 35.

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

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

Returned Values

The asynchronous entry point returns one of the following values:

AP_COMPLETED
The verb has already completed. The application can examine the parameters in the VCB to determine whether the verb completed successfully. Communications Server does not call the supplied callback routine for this verb.

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

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

Using the Asynchronous Entry Point

When using the asynchronous entry point, note the following:

- If an application specifies a null pointer in the *comp_proc* parameter, the verb will complete synchronously (as though the application issued the verb using the synchronous entry point).
- If the call to APPC_Async is made from within an application callback, specifying a null pointer in the *comp_proc* parameter is not permitted. In such cases, Communications Server rejects the verb with primary return code value AP_PARAMETER_CHECK and secondary return code value AP_SYNC_NOT_ALLOWED.
- The application must not attempt to use or modify any parameters in the VCB until the callback routine has been called.
- Multiple verbs do not necessarily complete in the order in which they were issued. In particular, if an application issues an asynchronous verb followed by a synchronous verb, the completion of the synchronous verb does not guarantee that the asynchronous verb has already completed.
- The [MC_]RECEIVE_AND_POST verb includes a pointer to a callback routine as one of the VCB parameters. This verb can be issued using either the synchronous or the asynchronous entry point. Communications Server uses the callback routine specified in the VCB to return the results of this verb. The callback routine specified on the asynchronous entry point is used only if the application supplies a null pointer for the callback routine in the VCB.

Callback Routine for Asynchronous Verb Completion

When using the asynchronous entry point, the application must supply a pointer to a callback routine. This section describes how Communications Server uses this routine, and the functions that it must perform.

Function Call

```
AP_CALLBACK (*comp_proc);
typedef void (*AP_CALLBACK)  (
                               void *          vcb,
                               unsigned char   tp_id[8],
                               AP_UINT32       conv_id,
                               AP_CORR         corr
                             );
typedef union ap_corr  {
                          void *               corr_p;
                          AP_UINT32            corr_l;
                          AP_INT32             corr_i;
                       } AP_CORR;
```

Supplied Parameters

Communications Server calls the callback routine with the following parameters:

vcb Pointer to the VCB supplied by the application. The VCB now includes the returned parameters set by Communications Server.

tp_id The 8-byte TP identifier of the TP in which the verb was issued.

conv_id
The conversation identifier of the conversation in which the verb was issued.

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

The callback routine need not use all of these parameters. The callback routine can perform all the necessary processing on the returned VCB, or it can simply set a variable to inform the main program that the verb has completed.

Returned Values

The function does not return a value.

Using the Callback Routine for Asynchronous Verb Completion

When using the callback routine for asynchronous verb completion, the application can issue additional asynchronous APPC verbs from within the callback routine, if required. Communications Server rejects any synchronous verbs issued from within a callback routine with the primary and secondary return codes AP_PARAMETER_CHECK and AP_SYNC_NOT_ALLOWED.

APPC Entry Points: Windows Systems

WINDOWS

A Windows application accesses APPC using the following entry points:

WinAPPCStartup
Registers the application as a Windows APPC user, and determines whether the APPC software supports the level of function required by the application.

WinAsyncAPPC
Issues an APPC verb. The verb normally completes asynchronously and does not block; APPC indicates the completion by posting a message to the application window.

WinAsyncAPPCEx
Issues an APPC verb. If the verb completes asynchronously, APPC indicates the completion by signaling an event handle. Use this function instead of the blocking versions of the verbs to allow multiple sessions to be handled on the same thread.

WinAPPCCancelAsyncRequest
Cancels an outstanding asynchronous verb (one issued using the WinAsyncAPPC entry point). Depending on which verb is outstanding, this may also end the conversation or the TP or deactivate the session being used by a conversation.

WinAPPCCleanup
Unregisters the application when it has finished using APPC.

APPC Issues an APPC verb. The verb blocks; that is, the application's processing is suspended until APPC has finished processing the verb and returned the results.

WinAPPCCancelBlockingCall
Cancels an outstanding blocking verb (one issued using the APPC entry point). Depending on which verb is outstanding, this may also end the conversation or the TP, or deactivate the session being used by a conversation. For more information about the circumstances in which this call may be required, see “Blocking Verbs” on page 42.

WinAPPCIsBlocking
Checks whether there is a blocking verb outstanding for this application. For more information about the circumstances in which this call may be required, see “Blocking Verbs” on page 42.

WinAPPCSetBlockingHook
Specifies the blocking procedure that APPC uses while processing blocking

verbs; this replaces APPC's default blocking procedure. The blocking procedure is called repeatedly until the verb processing has completed. For more information, see "Blocking Verbs" on page 42.

WinAPPCUnhookBlockingHook
Unregisters the blocking procedure specified by a previous WinAPPCSetBlockingHook call, so that APPC reverts to using the default blocking procedure.

GetAppcConfig
Returns information about remote LUs configured for use by a specified local LU and mode. This function is provided for use by 5250 emulation programs; the information returned is taken from the 5250 user records in the Communications Server configuration.

GetAppcReturnCode
Generates a printable character string for the primary and secondary return codes obtained on an APPC verb.

These entry points are defined in the Windows APPC header file **winappc.h**. This file is installed in the subdirectory **/sdk** within the directory where you installed the Windows Client software.

The application must call WinAPPCStartup before attempting to issue any APPC verbs.

It then issues APPC verbs using one of the following entry points:

- WinAsyncAPPC or WinAsyncAPPCEx (asynchronous). If you are developing new applications for Windows, use one of these entry points.
- APPC (blocking). This entry point is provided for compatibility with the AIX / Linux APPC implementation. "Blocking Verbs" on page 42 provides more information about how blocking verbs operate in the Windows environment.

An application that provides 5250 emulation can use GetAppcConfig to obtain information about remote APPC LUs that can be accessed using a given local LU.

If a verb returns with return codes other than AP_OK, the application can use GetAppcReturnCode to obtain a text string representation of these return codes, which can be used to generate standard error messages.

When the application has finished issuing APPC verbs, it must call WinAPPCCleanup before terminating. After calling WinAPPCCleanup, the application must not attempt to issue any more APPC verbs (unless it first calls WinAPPCStartup to reinitialize).

The following sections describe these Windows entry points.

WinAPPCStartup

The application uses WinAPPCStartup to register as a Windows APPC user and to determine whether the APPC software supports the Windows APPC version that the application requires.

Function Call

```
int WINAPI WinAPPCStartup  (
                             WORD            wVersionRequired;
                             WAPPCDATA far * lpData;
                           )
```

```
typedef struct
 {
   WORD          wVersion;
   char          szDescription[128];
 } WAPPCDATA;
```

Supplied Parameters

When the application uses the WinAPPCStartup entry point to issue a verb, it supplies the following parameters:

wVersionRequired
The version of Windows APPC that the application requires. The low-order byte specifies the major version number, and the high-order byte specifies the minor version number. For example:

Version	wVersionRequired
1.0	0x0001
1.1	0x0101
2.0	0x0002

If the application can use more than one version, it specifies the highest version that it can use.

Returned Values

WinAPPCStartup returns one of the following values:

0 (zero)
The application was registered successfully and the Windows APPC software supports either the version number specified by the application or a lower version. The application should check the version number in the WAPPCDATA structure to ensure that it is high enough.

WAPPCVERNOTSUPPORTED
The version number specified by the application was lower than the lowest version supported by the Windows APPC software. The application was not registered.

WAPPCSYSNOTREADY
The application was not registered. This may be because the Remote API Client on Windows software has not been started, or the local node is not active, or because of another system failure such as a resource shortage.

If the return value from WinAPPCStartup is 0 (zero), the WAPPCDATA structure contains information about the support provided by the Windows APPC software. If the return value is nonzero, the contents of this structure are undefined and the application should not check them. The parameters in this structure are as follows:

wVersion
The Windows APPC version number that the software supports, in the same format as the *wVersionRequired* parameter. If the software supports the requested version number, this parameter is set to the same value as the *wVersionRequired* parameter; otherwise it is set to the highest version that the software supports, which is lower than the version number supplied by the application. The application must check the returned value and take action as follows:

- If the returned version number is the same as the requested version number, the application can use this Windows APPC implementation.

- If the returned version number is lower than the requested version number, the application can use this Windows APPC implementation but must not attempt to use features that are not supported by the returned version number. If it cannot do this because it requires features not available in the lower version, it should fail its initialization and not attempt to issue any APPC verbs.

szDescription
A text string describing the Windows APPC software.

WinAsyncAPPC

The application uses this function to issue an APPC verb. If the verb completes asynchronously, APPC indicates the completion by posting a message to the application's Windows handle.

Before using the `WinAsyncAPPC` call for the first time, the application must use the `RegisterWindowMessage` call to obtain the message identifier that APPC will use for messages indicating asynchronous verb completion. For more information, see "Usage" on page 40.

Function Call

```
HANDLE WINAPI WinAsyncAPPC  (
                              HWND  hWnd,
                              long  vcbptr
                            );
```

Supplied Parameters

The supplied parameters are:

hWnd A Windows handle that APPC will use to post a message indicating asynchronous verb completion.

vcbptr A pointer to the VCB structure for the verb. This parameter is defined as a long integer, and so needs to be cast from a pointer to a long integer. For more information about the VCB structure and on its usage for individual verbs, see Chapter 3, "APPC Control Verbs," on page 63 and Chapter 4, "APPC Conversation Verbs," on page 91.

Note: The APPC VCBs contain many parameters marked as "reserved"; some of these are used internally by the Communications Server software, and others are not used in this version but may be used in future versions. Your application must not attempt to access any of these reserved parameters; instead, it must set the entire contents of the VCB to zero to ensure that all of these parameters are zero, before it sets other parameters that are used by the verb. This ensures that Communications Server will not misinterpret any of its internally-used parameters, and also that your application will continue to work with future Communications Server versions in which these parameters may be used to provide new functions.

To set the VCB contents to zero, use `memset`:

memset(vcb, 0, sizeof(vcb));

Returned Values

The return value from the function is one of the following:

`Handle` The function call was successful (accepted). When the verb later completes,

APPC uses this handle as an identifier in the message passed to the application's window procedure (for more information, see "Usage"). The application also uses this handle as a parameter to the `WinAPPCCancelAsyncRequest` call if it needs to cancel the outstanding verb.

`0` (zero)
The function call was not successful (not accepted).

Usage

Before using `WinAsyncAPPC` for the first time, the application must use the `RegisterWindowMessage` call to obtain the message identifier that APPC will use for messages indicating asynchronous verb completion. `RegisterWindowMessage` is a standard Windows function call, not specific to APPC; refer to your Windows documentation for more information about the function. (You do not need to issue the call again before subsequent APPC verbs; the returned value will be the same for all calls issued by the application.)

The application must pass the string "WinAsyncAPPC" to the function; the returned value is a message identifier.

Each time an APPC verb that was issued using the `WinAsyncAPPC` entry point completes asynchronously, APPC posts a message to the Windows handle specified on the `WinAsyncAPPC` call. The format of the message is as follows:

- The message identifier is the value returned from the `RegisterWindowMessage` call.
- The *lParam* argument contains the address of the VCB that was supplied to the original `WinAsyncAPPC` call; the application can use this address to access the returned parameters in the VCB structure.
- The *wParam* argument contains the handle that was returned to the original `WinAsyncAPPC` call.

WinAsyncAPPCEx

The application uses this function to issue an APPC verb. If the verb completes asynchronously, APPC indicates the completion by signaling an event handle. Use this function instead of the blocking versions of the verbs to allow multiple sessions to be handled on the same thread.

Function Call

```
HANDLE WINAPI WinAsyncAPPCEx  (
                                HANDLE  eventhandle,
                                long    vcbptr
                              );
```

Supplied Parameters

The supplied parameters are:

eventhandle
An event handle that APPC will signal to indicate asynchronous verb completion.

vcbptr A pointer to the VCB structure for the verb. This parameter is defined as a long integer, and so needs to be cast from a pointer to a long integer. For more information about the VCB structure and on its usage for individual verbs, see Chapter 3, "APPC Control Verbs," on page 63 and Chapter 4, "APPC Conversation Verbs," on page 91.

Note: The APPC VCBs contain many parameters marked as "reserved"; some of these are used internally by the Communications Server software, and others are not used in this version but may be used in future versions. Your application must not attempt to access any of these reserved parameters; instead, it must set the entire contents of the VCB to zero to ensure that all of these parameters are zero, before it sets other parameters that are used by the verb. This ensures that Communications Server will not misinterpret any of its internally-used parameters, and also that your application will continue to work with future Communications Server versions in which these parameters may be used to provide new functions.

To set the VCB contents to zero, use `memset`:

```
memset(vcb, 0, sizeof(vcb));
```

Returned Values

The return value from the function is one of the following:

`Handle` The function call was successful (accepted) and the return value is an asynchronous task handle. When the verb later completes, APPC uses this handle for event notification to the application (for more information, see "Usage"). The application also uses this handle as a parameter to the `WinAPPCCancelAsyncRequest` call if it needs to cancel the outstanding verb.

`0` (zero)
The function call was not successful (not accepted).

Usage

This function is intended for use with `WaitForSingleObject` or `WaitForMultipleObjects` in the Windows API. When the asynchronous operation is complete, the application is notified through the signaling of the event. Upon signaling of the event, examine the primary return code and secondary return code for any error conditions.

WinAPPCCancelAsyncRequest

The application uses this function to cancel an oustanding APPC verb (issued using the `WinAsyncAPPC` entry point).

Function Call

```
int WINAPI WinAPPCCancelAsyncRequest (HANDLE  Handle);
```

Supplied Parameters

The supplied parameter is:

Handle The handle that was returned on the original `WinAsyncAPPC` call for the verb.

Returned Values

The return value from the function is one of the following:

`0` (zero)
The outstanding verb was canceled successfully.

`WAPPCINVALID`
The supplied parameter did not match the handle of any outstanding APPC verb.

WAPPCALREADY
The APPC verb identified by the supplied handle has already completed. (The application may already have processed the message resulting from the verb completion, or the message may still be waiting in the application's message queue.)

Usage
In addition to canceling the outstanding verb, APPC may also end the conversation or TP on which the verb was issued, bring down the session, or both. The action taken depends on the verb that was canceled. APPC also posts a message to the application indicating completion of the canceled verb; the primary return code for the verb is AP_CANCELLED.

WinAPPCCleanup
The application uses the WinAPPCCleanup function to unregister as a Windows APPC user, after it has finished issuing APPC verbs.

Function Call
```
BOOL WINAPI WinAPPCCleanup (void);
```

Supplied Parameters
No parameters are supplied with the WinAPPCCleanup function.

Returned Values
The return value from the function is one of the following:

TRUE The application was unregistered successfully.

FALSE An error occurred during processing of the call, and the application was not unregistered. Check the log files for messages indicating the cause of the failure.

Blocking Verbs
This section describes how blocking verbs operate in the Windows environment if the calling application is single-threaded, and provides information that you need to be aware of when writing applications to use blocking verbs. (Typically a Windows application would use multiple threads to avoid the problem of a blocking verb blocking the entire application.)

Although a verb issued to the APPC entry point appears to suspend the application until verb processing is completed, the APPC library has to yield control of the system while waiting for the Remote API Client to complete the processing, in order to allow other processes to run. To do this, the application uses a blocking function, which is called repeatedly while the library is waiting; the function enables Windows messages to be sent to other processes. For more information about this function, see “Default Blocking Function” on page 43.

It is possible for the blocking function to dispatch a message to the application that issued the original blocking verb; in this case, the application can be re-entered even though it has a blocking call outstanding. In these circumstances, the application can continue with other processing not related to issuing APPC verbs. However, it cannot issue another verb to the APPC entry point (or to any other Remote API Client API) while the first verb is outstanding; the verb will be rejected with the primary return code AP_THREAD_BLOCKING.

The application can check whether a blocking verb is outstanding (that is, whether it has been re-entered as a result of a received message while the verb was outstanding) by using the WinAPPCIsBlocking function (for more information, see "WinAPPCIsBlocking" on page 44). If this function indicates that a blocking call is outstanding, the application should not attempt to issue further APPC verbs using the blocking entry point. The application can, however, do the following:

- Continue with other processing.
- Issue APPC verbs using the asynchronous entry point.
- Issue WinAPPCCancelBlockingCall to cancel the outstanding blocking verb.

Default Blocking Function

The standard blocking function used by the Windows APPC library is as follows:

```
BOOL far pascal DefaultBlockingHook (void)  {
       MSG msg;
       /* get the next message if any */
       if ( PeekMessage (&msg,NULL,0,0,PM_NOREMOVE)  )  {
             if ( msg.message = = WM_QUIT  )
                   return FALSE;   // let app process WM_QUIT
             PeekMessage (&msg,NULL,0,0,PM_REMOVE);
             TranslateMessage (&msg);
             DispatchMessage (&msg);
       }
       /* TRUE if no WM_QUIT received */
       return TRUE;
}
```

If the application needs to have other processing performed as part of the blocking function, it can specify its own blocking function to replace the default one provided by APPC. To do this, it uses the WinAPPCSetBlockingHook call (see "WinAPPCSetBlockingHook" on page 45).

A blocking function must return FALSE if it receives a WM_QUIT message; this means that Windows APPC returns control to the application, which can then process the message and terminate. Otherwise, the function must return TRUE.

APPC

The application uses this function to issue an APPC verb, which blocks until verb processing is completed. For compatibility with earlier APPC implementations, the Remote API Client also provides the entry points APPC_C and APPC_P, which can be used in the same way as APPC.

This entry point provides support for synchronous APPC verbs on Windows, which may assist in migrating from other operating system environments.

Function Call

```
void WINAPI APPC  (
                    long  vcbptr
                  )
```

Supplied Parameters

The supplied parameter is:

vcbptr A pointer to the VCB structure for the verb. This parameter is defined as a long integer, and so needs to be cast from a pointer to a long integer. For the definition of the VCB structure for each APPC verb, see Chapter 3, "APPC Control Verbs," on page 63 and Chapter 4, "APPC Conversation Verbs," on page 91.

Note: The APPC VCBs contain many parameters marked as "reserved"; some of these are used internally by the Communications Server software, and others are not used in this version but may be used in future versions. Your application must not attempt to access any of these reserved parameters; instead, it must set the entire contents of the VCB to zero to ensure that all of these parameters are zero, before it sets other parameters that are used by the verb. This ensures that Communications Server will not misinterpret any of its internally-used parameters, and also that your application will continue to work with future Communications Server versions in which these parameters may be used to provide new functions.

To set the VCB contents to zero, use `memset`:

```
memset(vcb, 0, sizeof(vcb));
```

Returned Values

The function does not return a value. When the call returns, the application should check the *primary_rc* and *secondary_rc* parameters in the VCB structure to determine whether the verb completed successfully. For information about the parameters returned in the VCB structure, see the descriptions of individual verbs in Chapter 3, "APPC Control Verbs," on page 63 and Chapter 4, "APPC Conversation Verbs," on page 91.

WinAPPCCancelBlockingCall

The application uses the `WinAPPCCancelBlockingCall` function to cancel an outstanding blocking APPC verb (issued using the `APPC` entry point).

Function Call

```
BOOL WINAPI WinAPPCCancelBlockingCall (void);
```

Supplied Parameters

No parameters are supplied for this entry point. (There can be only one blocking verb outstanding at any time, so there is no need to identify the particular verb to be canceled.)

Returned Values

The return value from the function is one of the following:

TRUE The outstanding verb was canceled successfully.

FALSE Either no blocking APPC verb was outstanding, or an error occurred during processing of the call and the verb was not canceled.

Usage

In addition to canceling the outstanding verb, APPC also ends the conversation on which the verb was issued and brings down the session. If the verb is one that relates to a TP rather than to a conversation (such as RECEIVE_ALLOCATE or TP_STARTED), APPC ends the TP.

WinAPPCIsBlocking

The application uses the `WinAPPCIsBlocking` function to check whether there is a blocking APPC verb outstanding (a verb issued using the `APPC` entry point).

Function Call

```
BOOL WINAPI WinAPPCIsBlocking (void);
```

Supplied Parameters
No parameters are supplied with this function.

Returned Values
The return value from the function is one of the following:

TRUE A blocking APPC verb is outstanding. If necessary, the application can use the `WinAPPCCancelBlockingCall` function to cancel it.

FALSE A blocking APPC verb is not outstanding.

WinAPPCSetBlockingHook
The application uses this call to specify its own blocking function, which APPC will use instead of the default blocking function. For more information about how the blocking function operates and on the functions it must perform, see "Blocking Verbs" on page 42.

Function Call
```
FARPROC WINAPI WinAPPCSetBlockingHook (FARPROC lpBlockFunc);
```

Supplied Parameters
The supplied parameter is:

lpBlockFunc
The procedure instance address of the application's blocking function. The application should use the `MakeProcInstance` call to obtain this address; refer to your Windows documentation for more information.

Returned Values
The return value is the procedure instance address of the previous blocking function. If the application is using more than one blocking function, and will need to restore the previous blocking function later, it should save this address; it can then issue `WinAPPCSetBlockingHook` again using the saved value, to restore the previous blocking function. If it is using only one blocking function, or will not need to restore the previous value, it can ignore the return value from this call.

Usage
The new blocking function remains in effect until the application issues one of the following calls:

- `WinAPPCSetBlockingHook` (with a different procedure instance address) to specify a new blocking function or to restore a previous one
- `WinAPPCUnhookBlockingHook` (see "WinAPPCUnhookBlockingHook"), to stop using the current blocking function and return to the default blocking function

WinAPPCUnhookBlockingHook
The application uses this call to remove its own blocking function, which it has previously specified using `WinAPPCSetBlockingHook`, and revert to using APPC's default blocking function.

Function Call
```
BOOL WINAPI WinAPPCUnhookBlockingHook (void);
```

Supplied Parameters
No parameters are supplied for this function.

Returned Values

The return value from the function is one of the following:

TRUE The blocking function was removed successfully; any further blocking calls will use the default blocking function.

FALSE The call did not complete successfully.

GetAppcConfig

The GetAppcConfig function is provided for use by 5250 emulation programs. The function returns information about the remote LUs that a specified local LU can access, as defined in the 5250 emulation user records in the Communications Server configuration.

To determine the information required by this call, Communications Server checks the user name configured for the Windows client against the 5250 user records defined in the configuration (or, if the user name is not explicitly defined, checks for a <DEFAULT> record). In the appropriate user record, it matches the local LU alias and mode name supplied on this call against the session definitions, and returns the remote LU alias for each matching session.

The application supplies a Windows handle to which APPC can post a message when the verb completes asynchronously. Before using GetAppcConfig for the first time, the application must use RegisterWindowMessage to obtain the message identifier that APPC will use for the message indicating asynchronous completion of the call, and WinAPPCStartup to register as a Windows APPC application. For more information, see the description of WinAPPCStartup in "WinAPPCStartup" on page 37 and "Usage" on page 48.

An alternative method of indicating completion of the call is to supply a pointer to an integer value (the *AsyncRetCode* parameter) in which APPC can return values to indicate that the call has failed, is in progress, or has completed. Windows applications are recommended to use the first method, supplying a Windows handle.

Function Call

```
HANDLE WINAPI GetAppcConfig (
                              HWND       hWnd,
                              char far   *LocalLU,
                              char far   *Mode,
                              int far    *NumRemLU,
                              int        MaxRemLU,
                              char far   *RemLU,
                              int far    *AsyncRetCode
                            );
```

Supplied Parameters

The supplied parameters are:

hWnd A Windows handle that APPC will use to post a message indicating asynchronous completion of this call. If this parameter is used, the pointer to the *AsyncRetCode* parameter must be a null pointer.

LocalLU

A pointer to the alias of the local LU for which configuration information is required. This is an ASCII string of up to eight characters, terminated with a null character (binary zero); if the LU alias is shorter than eight characters, it must be followed immediately by the null character and not space-padded.

To indicate the default local LU, set this parameter to point to a string consisting of eight ASCII spaces followed by a null character.

Mode A pointer to the name of the mode (used by the local LU) for which configuration information is required. This is an ASCII string of up to eight characters, terminated with a null character (binary zero); if the mode name is shorter than eight characters, it must be followed immediately by the null character and not space-padded. For 5250 emulation programs, the mode name is normally `QPCSUPP`.

NumRemLU
A pointer to an integer that APPC can use to return the number of remote LUs configured.

MaxRemLU
The maximum number of remote LU aliases that can be accommodated in the supplied data buffer (see the following parameter). Each LU alias requires 9 bytes, so the length of the supplied buffer must be at least nine times the supplied value of *MaxRemLU*.

RemLU
A buffer to contain the returned remote LU aliases.

AsyncRetCode
If the application is using the recommended method of indicating completion, this parameter is reserved; the application must supply a null pointer.

If the application is using the alternative method, this parameter is a pointer to the integer that APPC uses for the asynchronous return code from the function. In this case, the *hWnd* parameter must be a null handle.

Returned Values

When the call returns, the application can test the value of the expression "*ReturnedHandle* & APPC_CFG_SUCCESS " to determine whether the function was successful.

If the value of the expression "*ReturnedHandle* & APPC_CFG_SUCCESS" is `TRUE`, and the application is using the recommended method to indicate completion, the return value is a handle. When the function later completes, APPC uses this handle as an identifier in the message passed to the application's window procedure (for more information, see "Usage" on page 48).

If the value of the expression "*ReturnedHandle* & APPC_CFG_SUCCESS" is `TRUE`, and the application is using the alternative method to indicate completion, the *AsyncRetCode* parameter is set to `APPC_CFG_PENDING`. The application should test this value periodically to check for completion. When the function later completes, APPC sets this parameter to one of the asynchronous return codes listed in "Usage" on page 48.

If the value of the expression is `FALSE`, the function call was not accepted. The value of *ReturnedHandle* is one of the following:

`APPC_CFG_ERROR_NO_APPC_INIT`
The application has not issued the `WinAPPCStartup` call. This call must be issued before `GetAppcConfig` is used.

`APPC_CFG_ERROR_INVALID_HWND`
The application supplied a Windows handle that was not valid.

APPC_CFG_ERROR_BAD_POINTER
The application supplied a null pointer to a Windows handle, to use the alternative method for indicating completion, but supplied a pointer for the *AsyncRetCode* parameter that was not valid.

APPC_CFG_ERROR_UNCLEAR_COMPLETION_MODE
The application supplied both a Windows handle (in the *hWnd* parameter) and a non-null pointer to the *AsyncRetCode* parameter, so APPC could not determine how to indicate asynchronous completion.

APPC_CFG_ERROR_TOO_MANY_REQUESTS
Too many `GetAppcConfig` requests are already outstanding. The application should yield, to allow other processes to run, and retry the call later.

APPC_CFG_ERROR_GENERAL_FAILURE
A system error occurred.

Usage

Before using `GetAppcConfig` for the first time, the application must use the `RegisterWindowMessage` call to obtain the message identifier that APPC will use for messages indicating asynchronous completion. `RegisterWindowMessage` is a standard Windows function call, not specific to APPC; refer to your Windows documentation for more information about the function. (The application does not need to issue the call again before subsequent `GetAppcConfig` calls; the returned value will be the same for all calls issued by the application.)

The application must pass the value `WIN_APPC_CFG_COMPLETION_MSG` to the function; the returned value is a message identifier.

An application can indicate completion by using a Windows handle or by using an alternative method, as follows:

- If the application is using a Windows handle to indicate completion, APPC posts a message to this Windows handle when the call completes asynchronously. The format of the message is as follows:
  - The message identifier is the value returned from the `RegisterWindowMessage` call.
  - The *wParam* argument contains the handle that was returned to the original `GetAppcConfig` call.
  - The *lParam* argument contains one of the following asynchronous return codes:

    APPC_CFG_SUCCESS_NO_DEFAULT_REMOTE
    The configuration was retrieved successfully. No default remote LU is configured for the specified local LU and mode.

    APPC_CFG_SUCCESS_DEFAULT_REMOTE
    The configuration was retrieved successfully. A default remote LU is configured for the specified local LU and mode. (Communications Server does not return this value, because it does not have a concept of configuring default remote LUs; however, the application should allow for this return code to ensure compatibility with other APPC implementations.)

    APPC_CFG_ERROR_NO_DEFAULT_LOCAL_LU
    The application supplied a blank local LU alias, indicating the default local LU, but no default local LU is configured.

APPC_CFG_ERROR_BAD_LOCAL_LU
The supplied local LU alias did not match any configured local LU alias used for 5250 emulation.

APPC_CFG_ERROR_GENERAL_FAILURE
A system error occurred.

- If the application is using the alternative method to indicate completion, APPC sets the asynchronous return code to one of the return codes in the list for the *lParam* argument (in the Windows message) when the call completes.

The application can test for success or failure by testing the expressions "*RetCode* & APPC_CFG_SUCCESS" or "*RetCode* & APPC_CFG_FAILURE", where *RetCode* is the *lParam* argument in the Windows message or the *AsyncRetCode* parameter returned to the application. If "*RetCode* & APPC_CFG_SUCCESS" is `TRUE`, the call was successful; if "*RetCode* & APPC_CFG_FAILURE" is `TRUE`, the call failed.

If the call was successful, the application can then check the values of the *NumRemLU* and *RemLU* parameters:

- *NumRemLU* contains the total number of remote LUs configured. If this number is greater than the supplied *MaxRemLU* parameter, the supplied buffer was not large enough to contain all the remote LU aliases. The application can use the returned aliases, or can reissue `GetAppcConfig` with a large enough buffer to contain all the aliases.
- *RemLU* contains the aliases of the remote LUs. Each alias is a string of up to eight characters followed by a null character, and occupies 9 bytes of the buffer. The number of LU aliases returned is the smaller of the supplied parameter *MaxRemLU* and the returned parameter *NumRemLU*.

To determine the information required by this call, Communications Server checks the user name configured for the Windows client against the 5250 user records defined in the configuration (or, if the user name is not explicitly defined, checks for a <DEFAULT> record). In the appropriate user record, it matches the local LU alias and mode name supplied on this call against the session definitions, and returns the remote LU alias for each matching session.

GetAppcReturnCode

This call returns a printable character string interpreting the return codes from a supplied VCB. The string can be used to generate application error messages for return codes other than `AP_OK`.

This call provides strings for display to the end user of an APPC application. For return codes indicating configuration problems or user errors (for example if a required component is not configured or not started), the string should provide sufficient information to help the user correct the problem. For return codes indicating application errors (for example if the application has issued a verb that is not valid or failed to supply a required parameter), the user is not generally able to correct the problem; in these cases, the string is meaningful only to an application developer.

Function Call

```
int WINAPI GetAppcReturnCode (
                                long           vcbptr,
                                unsigned int   buffer_length,
                                unsigned char far *  buffer_addr
                               );
```

Supplied Parameters

The supplied parameters are:

vcbptr A pointer to the VCB structure for the verb. This parameter is defined as a long integer, and so needs to be cast from a pointer to a long integer. For more information about the VCB structure and on its usage for individual verbs, see Chapter 3, "APPC Control Verbs," on page 63 or Chapter 4, "APPC Conversation Verbs," on page 91.

buffer_length
The length (in bytes) of the buffer supplied by the application to hold the returned data string. The recommended length is 256 bytes.

buffer_addr
The address of the buffer supplied by the application to hold the returned data string.

Returned Values

The return value from the function is one of the following:

0 (zero)
The function completed successfully. The returned character string is in the buffer identified by the *buffer_addr* parameter. This string is terminated by a null character (binary zero), but does not include a trailing new-line (\n) character.

0x20000001
APPC could not read from the supplied VCB, or could not write to the supplied data buffer.

0x20000002
The supplied data buffer is too small to hold the returned character string.

0x20000003
The dynamic link library **APPCST32.DLL**, which generates the returned character strings for this function, could not be loaded.

AIX or Linux Considerations

UNIX

This section summarizes the information you need to consider when developing TPs for use in the AIX or Linux environment.

Multiple Processes

If the process that issued TP_STARTED or RECEIVE_ALLOCATE then forks to create a child process, the child process cannot use the *tp_id* that was returned to the parent process. It can, however, issue its own TP_STARTED or RECEIVE_ALLOCATE to obtain its own *tp_id*.

Two or more instances of the same TP can run as different processes, but each instance is assigned its own *tp_id*.

You can write an application in which one process contains many TPs, each with its own *tp_id*. However, you need to design the application carefully to avoid "deadlock" situations, in which an APPC verb cannot complete because of the state of other conversations and TPs in the same process. This might happen if the program is waiting on one conversation for information to be sent to it before returning some other data, and another conversation from the same process is waiting for this data before it can send the information originally required by the first conversation. To some extent this can be avoided by using a separate process for each TP.

Compiling and Linking the APPC Application

AIX Applications

To compile and link 32–bit applications, use the following options:

-bimport:/usr/lib/sna/appc_r.exp -I /usr/include/sna

To compile and link 64–bit applications, use the following options:

-bimport:/usr/lib/sna/appc_r64_5.exp -I /usr/include/sna

Linux Applications

Before compiling and linking an APPC application, specify the directory where shared libraries are stored, so that the application can find them at run time. To do this, set the environment variable `LD_RUN_PATH` to **/opt/ibm/sna/lib**, or to **/opt/ibm/sna/lib64** if you are compiling a 64–bit application.

To compile and link 32–bit applications, use the following options:

-I /opt/ibm/sna/include -L /opt/ibm/sna/lib -lappc -lsna_r -lpthread -lpLiS

To compile and link 64–bit applications, use the following options:

-I /opt/ibm/sna/include -L /opt/ibm/sna/lib64 -lappc -lsna_r -lpthread -lpLiS

The option `-lpLiS` is required only if you will be running the application on a Communications Server server; you do not need to use it if you are building the application on an IBM Remote API Client and it will run only on the client. As an alternative to using this option, you can set the the environment variable `LD_PRELOAD` to **/usr/lib/libpLiS.so** before compiling and linking the application.

Windows Considerations

WINDOWS

This section summarizes the processing considerations that you need to be aware of when developing applications on a Remote API Client for Windows. Windows processing considerations are:

- Compiling and linking APPC programs
- Terminating applications

Compiling and Linking APPC Programs

The following processing considerations are important when you compile and link APPC programs on Windows:

Compiler options for structure packing
The VCB structures for APPC verbs are not packed. Do not use compiler options that change this packing method. BYTE parameters are on BYTE boundaries, WORD parameters are on WORD boundaries, and DWORD parameters are on DWORD boundaries

Header files
The main APPC header file to be included in Windows APPC applications is named **winappc.h**. If your application uses the `GetAppcConfig` call, you also need to include the **appccfg.h** header file. These files are installed in the subdirectory **\sdk** for 32–bit applications, or **\sdk64** for 64–bit applications, within the directory where you installed the Windows Client software.

Load-time linking
To link the TP to APPC at load time, link the TP to the library **\sdk\wappc32.lib** for 32–bit applications, or **\sdk64\wappc32.lib** for 64–bit applications.

Run-time linking
To link the TP to APPC at run time, include the following calls in the TP:

- `LoadLibrary` to load the APPC dynamic link library **wappc32.dll**.
- `GetProcAddress` to specify APPC on each of the APPC entry points required (such as `WinAsyncAPPC`, `WinAPPCStartup`, and `WinAPPCCleanup`)
- `FreeLibrary` when the library is no longer required

Terminating Applications

APPC cannot tell when an application terminates under Windows. Therefore if an application must close (for example, if it receives a WM_CLOSE message), the application should issue the `WinAPPCCleanup` call. Failure to issue the call leaves the system in an indeterminate state; however, as much cleanup as possible is done when APPC later detects that the application has terminated.

Configuration Information

The Communications Server configuration file, which is set up and maintained by the System Administrator, contains information that is required for TPs to communicate. For additional information about configuration, refer to the *IBM Communications Server for Data Center Deployment on AIX or Linux Administration Guide*.

Invoked TP

Before writing an invoked TP, you must coordinate the local TP name with the System Administrator. The name can contain up to 64 characters.

UNIX

If you intend to use the extended form of the RECEIVE_ALLOCATE verb, in which the application can specify a local LU from which to accept incoming conversation requests, you must also coordinate the local LU alias (the name by which the local LU is known to the local TP) with the System Administrator. This alias can contain up to eight characters.

For more information, see the RECEIVE_ALLOCATE verb in Chapter 3, "APPC Control Verbs," on page 63.

Invoking TP

The following list summarizes the information you need to obtain from (or coordinate with) your System Administrator before writing an invoking TP:

Local LU Alias
Name by which the local LU is known to the local TP. This name can contain up to eight characters. For more information, see the description of the TP_STARTED verb in Chapter 3, "APPC Control Verbs," on page 63.

Partner TP Name
This name can contain up to 64 characters. For more information, see the description of the [MC_]ALLOCATE verb in Chapter 4, "APPC Conversation Verbs," on page 91.

Partner LU Alias
Name by which the partner LU is known to the local TP. This name can contain up to eight characters. For more information, see the description of the [MC_]ALLOCATE verb in Chapter 4, "APPC Conversation Verbs," on page 91.

Mode Name
Set of characteristics to be used in an LU-to-LU session. This name can contain up to eight characters. For more information, see the description of the [MC_]ALLOCATE verb in Chapter 4, "APPC Conversation Verbs," on page 91.

Conversation Security
If conversation security is to be used, a valid combination of user ID and password is required to access the invoked TP. The user ID and password can contain up to 10 characters. Both parameters are case-sensitive; the system distinguishes between uppercase and lowercase letters. Security information is stored in a security file. For more information, see "Overview of Conversation Security."

Overview of Conversation Security

You can use conversation security to require that the invoking TP provide a user ID and password before APPC allocates a conversation with the invoked TP.

In configuring the invoked TP, the System Administrator indicates whether to use conversation security. If so, the invoking TP must supply a combination of *user_id* and *password* as parameters of the [MC_]ALLOCATE verb. These parameters must match one of the combinations of *user_id* and *password* parameters established during configuration.

An invoked TP that in turn invokes another TP is a special case (see Chapter 1, “Concepts,” on page 1). Assume that TP A invokes TP B, which requires security information, and TP B in turn invokes TP C, which also requires security information. Through the [MC_]ALLOCATE verb, TP B can specify that conversation security has already been verified. In this case, APPC takes the user ID that was supplied by TP A to TP B, and sends this user ID to TP C with an “already verified” indication; TP C does not need to check the password.

UNIX

In some cases, a TP may need to indicate “already verified” security when it has not itself been invoked by another TP, but has obtained and verified the appropriate security information by another means (for example, by a user entering a user ID and password during a logon sequence). Communications Server supports this as follows:

- If the TP specifying “already verified” was itself invoked by another TP that specified a user ID and password, APPC sends this user ID.
- Otherwise, APPC takes the AIX / Linux user name with which the TP is running, truncated to 10 characters if necessary, and uses this as the conversation security user ID. Ensure that this name consists of valid AE-string characters and is a valid user name for the TP being invoked.
- If the application uses a different method of obtaining the security information (for example, if it requires the user to specify a user ID and password explicitly, rather than relying on the AIX / Linux system security), then it can use the SET_TP_PROPERTIES verb to specify this *user_id* to APPC before issuing the [MC_]ALLOCATE verb.

Communications Server also supports LU-LU session security, which provides security checking when starting the session between the local and remote APPC LUs. LU-LU session security is specified during configuration, and does not require any action in APPC programs. For more information, refer to the *IBM Communications Server for Data Center Deployment on AIX or Linux Administration Guide*.

Starting TPs

A conversation occurs between an invoking TP and an invoked TP. This section describes how the invoking and invoked TPs are started.

Invoking TPs

The invoking TP is started by a user entering a command, by a shell script, or by batch file command.

Invoked TPs

The invoked TP can be started by a user, automatically by Communications Server, or automatically by a TP server application. When the System Administrator configures each invoked TP, the System Administrator must specify whether the TP is started automatically or by the user.

Invoked TPs: User-Started

If an invoked TP is configured to be started by a user, the user can start the invoked TP either before or after the invoking TP. A TP started in this manner is called a queued, operator-started TP:

- If the user starts the invoking TP first, and does not start the invoked TP before the timeout value for starting the TP (see "Timeout Values for Invoked TPs" on page 56) is reached, the incoming Allocate fails.
- If the user starts the invoked TP before the invoking TP issues the [MC_]ALLOCATE verb, the invoked TP waits until the Attach from the invoking TP arrives, or until the RECEIVE_ALLOCATE timeout value (see "Timeout Values for Invoked TPs" on page 56) is reached.

Invoked TPs: Automatically Started by the Communications Server Attach Manager

An invoked TP can be configured to start automatically under one of the following conditions:

- The first time an Attach (the SNA message from the remote LU containing the allocation request) is received by the LU that serves the invoked TP. A TP started in this manner is called a queued, automatically started TP.

  If the invoked TP is not running, the first incoming Allocate starts it; a response to the incoming Allocate is held until the RECEIVE_ALLOCATE verb in the invoked TP is executed (or until a timeout occurs; see "Timeout Values for Invoked TPs" on page 56). At that time, APPC assigns a conversation ID, which is returned to both TPs as an identifier for the conversation.

  If the invoked TP is already running, the Attach is queued until the invoked TP issues another RECEIVE_ALLOCATE verb, or until it finishes running and can be restarted (or until a timeout occurs; see "Timeout Values for Invoked TPs" on page 56).
- Each time an Attach is received by the LU that serves the invoked TP. A new instance of the program is loaded and started with each incoming Attach. A TP started in this manner is called a nonqueued, automatically started TP.

  The Attach is queued until the RECEIVE_ALLOCATE verb in the invoked TP is executed (or until a timeout occurs; see "Timeout Values for Invoked TPs" on page 56). When RECEIVE_ALLOCATE is executed, APPC assigns a conversation ID, which is returned to both TPs as an identifier for the conversation.

  After it has ended a conversation, the invoked TP may terminate, or it may issue another RECEIVE_ALLOCATE. For frequently-used programs, this provides a way of avoiding the performance overhead of starting a new instance of the program for each conversation. Each time an Attach is received for a nonqueued, automatically started TP, Communications Server checks whether there is already a RECEIVE_ALLOCATE outstanding from an instance of this TP. If so, this TP is used for the incoming conversation; otherwise, Communications Server starts a new instance of the program.

Invoked TPs: Automatically Started by a TP Server Application

UNIX

When an Attach arrives at the Communications Server node, Communications Server distributes Attaches to TP server applications that have registered to receive the Attaches. The process Communications Server uses to route Attaches to an appropriate TP server consists of the following stages:

1. One or more applications register to receive Attaches for LU and TP names. A TP server application can use a wildcard to specify the scope of Attaches that the TP server is registered to receive. A TP server application can use a wildcard for any of the following:
   - Local LU alias
   - TP name
   - Fully qualified partner LU name, which can use a wildcard for any of the following:
     - Partial network name
     - Whole network name
     - Partial network name followed by CP name
     - Fully qualified partner LU name

   Only a single TP server application can register for a given TP and LU combination, including wildcards. For example, one TP server application can register TPNAME1 and *, while the same TP or another TP server application registers TPNAME1 and LUNAME1. Registration of this type is legal, but two TP server applications cannot both register TPNAME1 and LUNAME1. The second registration attempt will fail.
2. When an Attach arrives at Communications Server, Communications Server attempts to find the TP server application whose registration most closely matches the TP name, LU alias and fully qualified LU name received on the Attach. The matches are examined for greatest closeness in the following order:
   a. TP name match
   b. LU alias match
   c. Exact fully qualified partner LU name match
   d. Wildcard fully qualified partner LU name match

   When Communications Server finds a match, it delivers the Attach to the TP server application. The TP server has the following options:
   - Reject the Attach, in which case Communications Server returns the Attach to the invoking TP and includes an error code provided by the TP server application
   - Accept the Attach, in which case Communications Server informs the invoking TP that the Attach has been accepted
3. If no matches are found after trying all combinations above search criteria, Communications Server rejects the Attach and returns the Attach to the invoking TP with the appropriate error code.

Timeout Values for Invoked TPs

The Communications Server configuration specifies two timeout values that define how long APPC waits to establish a conversation between two TPs, as follows:

Timeout for Starting TP
: This value defines how long an Attach is queued waiting for the invoked TP to be started and to issue the RECEIVE_ALLOCATE verb. If RECEIVE_ALLOCATE is not issued within this time, the [MC_]ALLOCATE verb in the invoking TP fails. This timeout is defined in the configuration of the local LU that the TP uses.

Timeout for Servicing TP
This value defines how long a RECEIVE_ALLOCATE verb issued by the invoked TP waits for an Attach from the invoking TP. If an Attach is not received within this time, the RECEIVE_ALLOCATE verb in the invoked TP fails. The configuration can specify one of the following:

Infinite timeout
RECEIVE_ALLOCATE waits indefinitely

Zero timeout
RECEIVE_ALLOCATE fails unless the Attach has already been received

Finite timeout
A specific timeout value is provided

This timeout is defined for the invoked TP in the Communications Server invokable TP data file.

For more information about the configuration of invoked TPs, refer to the *IBM Communications Server for Data Center Deployment on AIX or Linux Administration Guide*.

LU-to-LU Sessions

An LU-to-LU session is a logical connection between two LUs. Conversations between TPs occur within sessions. One conversation can use a session at a time; many conversations can reuse the same session serially.

Communications Server enables an LU type 6.2 to have multiple sessions (two or more concurrent sessions with different partner LUs) and parallel sessions (two or more concurrent sessions with the same partner LU).

During configuration, the System Administrator determines how many sessions a particular LU supports and whether the LU supports parallel sessions.

Contention

When both LUs attempt to allocate a conversation on the same session at the same time, one must win (the contention winner) and one must lose (the contention loser). The contention-winner LU and the contention-loser LU are determined when the session is established.

In a session, the contention-loser LU must ask permission of the contention-winner LU before allocating a conversation. The contention winner may or may not grant permission. The contention-winner LU, on the other hand, simply allocates a conversation when desired.

During configuration, the System Administrator can define modes. A mode is a set of networking characteristics. Among the characteristics the System Administrator can specify within a mode definition is the number of contention-winner and contention-loser sessions for the local LU and partner LU that use the mode. (The TP issuing the [MC_]ALLOCATE verb specifies a mode, local LU, and partner LU.)

Basic Conversations

Basic conversations are generally used by service TPs. Service TPs are programs that provide services to other local programs. They are more complex than mapped conversations but provide an experienced LU 6.2 programmer with a greater degree of control over the transmission and handling of data. This section summarizes the characteristics of basic conversations where they differ from mapped conversations.

Logical Records

In a basic conversation, data is sent in the form of logical records. A logical record is a record that has the general data stream (GDS) syntax described in this section. For more information about GDS syntax, refer to the IBM publication *SNA Formats*.

The sending TP must format the data into logical records, and the receiving TP must decode the logical records into usable data. A TP can send multiple logical records with a single SEND_DATA verb, or it can send a single logical record in multiple parts (called segments) with multiple SEND_DATA verbs. A TP can receive multiple logical records with a single receive verb (RECEIVE_AND_WAIT, RECEIVE_IMMEDIATE, or RECEIVE_AND_POST), or it can receive a single logical record in multiple parts with multiple receive verbs.

If a logical record is a single record, it consists of the following fields:

- A 2-byte record length (LL) field
- A 2-byte GDS identifier (ID) field (for example, `0x12FF` identifies the data as application data)
- A data field that can range in length from 0–32,763 bytes

The first four bytes are called the LLID.

If a logical record has multiple parts, the first part has the same format as a single record, and all subsequent parts consist of the following fields:

- A 2-byte record length (LL) field
- A data field that can range in length from 0–32,765 bytes

The hexadecimal value for the LL field includes the two bytes for the LL field (and the two bytes for the ID field, if it is present). For example, a single part GDS with no zero bytes of data has a value of `0x0004` for its LL field. The LL field must be in high-low format, rather than byte-swapped format. For example, a length of 230 bytes is represented as `0x00E6`, rather than `0xE600`.

Bit 0 of byte 0 of the LL (the most significant bit) is used to indicate length continuation (segmentation). The following example shows ten bytes of data (each data byte has the value `DD`) split into three GDS segments. The first and second segments each contain four bytes of data, and the last segment contains two bytes of data.

```
8008 12FF DDDD DDDD
8006 DDDD DDDD
0004 DDDD
```

The following values for the LL field are not valid (except when sending a PS header as described in “Sending PS Headers in Logical Records” on page 59):

- `0x0000`

- 0x0001
- 0x8000
- 0x8001

In a mapped conversation, the sending TP sends one data record at a time, and the receiving TP receives one data record at a time. No record conversion is required of the TPs.

Sending PS Headers in Logical Records

If the conversation's synchronization level is AP_SYNCPT, the application may need to send and receive data in the form of PS Headers. The Syncpoint Manager is responsible for setting up the appropriate PS Headers to send to the partner application, based on Syncpoint functions required by the application, and for performing the required Syncpoint processing based on the PS Headers it receives from the partner application.

An LL field of 0x0001 indicates that the data is a PS Header; the sending application must specify an LL field of 0x0001 instead of specifying the length of the data field, and the receiving application must interpret the data as a PS Header if it receives an LL field of 0x0001. If the conversation's synchronization level is not AP_SYNCPT, the value 0x0001 is not a valid LL field, and will be rejected.

Reporting Errors and Abends

In a basic conversation, a TP can indicate whether an error or abend (abnormal program termination) was caused by a service TP or by a program using the service TP. This enables two communicating service TPs to distinguish between errors they may have caused and errors that may have been caused by the programs they serve.

Error Log

In case of an error or abend in a basic conversation, a TP can send an error message, in the form of a general data stream (GDS) error log variable, to the local log and to the partner LU.

Timeouts Versus Critical Errors

In a basic conversation, a TP can indicate whether an abend was caused by a timeout or by a critical error.

Writing TP Servers

UNIX

Use the following operational guidelines for writing TP servers:

1. TP server verbs must be issued using the asynchronous entry point APPC_Async, and not the synchronous entry point APPC.
2. Use REGISTER_TP_SERVER to register the application as a TP server. The REGISTER_TP_SERVER verb provides the address of a callback function used in later Attach notifications.
3. Use REGISTER_TP to register the TP names and local and remote LUs for which the TP server wishes to process Attaches. The TP server can use wildcards for both TP names and LU names so that it can choose to process

Attaches from a TP over a particular pair of LUs or to Attaches for all TPs over all LUs, or any combination of those conditions.

4. Use QUERY_ATTACH (with the unique identifier received when the notification callback is made after an Attach is received for the registered TP or LU) to query the Attach parameters to determine whether and how to process the Attach. The TP server can reject the Attach using REJECT_ATTACH or accept the Attach using ACCEPT_ATTACH and start a suitable application to process the Attach.
5. Issue the standard RECEIVE_ALLOCATE call to retrieve the Attach. The previously reserved *dload_id* parameter is used to specify the unique identifier of the Attach.
6. Use ABORT_ATTACH to cancel any further processing if, after issuing ACCEPT_ATTACH, an error is encountered.
7. Deregister the TP server for any of the previously registered TPs and LUs using UNREGISTER_TP.
8. Deregister the application as a TP server using the UNREGISTER_TP_SERVER verb.

TP Server Responsibilities

When a TP server handles an Attach, the TP server inherits a number of responsibilities normally performed by the Communications Server Attach Manager. These responsibilities include the following:

- Starting TPs to handle the conversations if a TP is configured to be automatically started
- Handling conversation security including parsing the Attach data returned on QUERY_ATTACH
- Conveying information from the Attach to the TP that the TP server starts to process the ensuing conversation

Default TP Server

Communications Server provides a default TP server, **snatpsrvd**, that is installed on all systems. This TP server uses the **sna_tps** file as the source for the configuration of the TPs that it can load. Other TP servers can modify and use this file by using DEFINE_TP_LOAD_INFO verbs. For more information about the DEFINE_TP_LOAD_INFO verb, refer to the *IBM Communications Server for Data Center Deployment on AIX or Linux NOF Programmer's Guide*. APPC provides the *tp_file_updates* parameter on a REGISTER_TP verb so that a TP server is notified when a change has been made to the **sna_tps** file and can take the required action.

Writing Portable TPs

The following guidelines are provided for writing TPs that they will are portable to other operating system environments:

- Include the APPC header file without any pathname prefix. Use include options on the compiler to locate the file (see the appropriate section for your operating system, earlier in this chapter). This enables the TP to be used in an environment with a different file system.

- Use the symbolic constant names for parameter values and return codes, not the numeric values shown in the header file; this ensures that the correct value will be used regardless of the way these values are stored in memory.
- Include a check for return codes other than those applicable to your current operating system (for example using a "default" case in a switch statement), and provide appropriate diagnostics.
- Use the asynchronous entry point.
- The [MC_]RECEIVE_AND_POST verb cannot be used if the TP is to be completely portable. If you use this verb, you will need to rewrite sections of the TP for use in other environments. You may want to restrict the use of this verb to a few specific routines, to allow easier modification.
- The CANCEL_CONVERSATION verb and the extended form of the RECEIVE_ALLOCATE verb are specific to the Communications Server APPC implementation, and may not be provided by other APPC implementations. If you use CANCEL_CONVERSATION or the extended form of RECEIVE_ALLOCATE, you will need to rewrite sections of the TP for use in other environments. You may want to restrict the use of these features to a few specific routines, to allow easier modification.

Chapter 3. APPC Control Verbs

This chapter contains a description of each APPC control verb. The following information is provided for each verb:

- Definition of the verb.
- Structure defining the verb control block (VCB) used by the verb. The structure is defined in the APPC header file **/usr/include/sna/appc_c.h** (AIX), **/opt/ibm/sna/include/appc_c.h** (Linux), or **sdk/winappc.h** (Windows). Parameters beginning with *reserv* are reserved.
- Parameters (VCB fields) supplied to and returned by APPC. For each parameter, the following information is provided:
  - Description
  - Possible values
  - Additional information
- Conversation state or states in which the verb can be issued.
- State or states to which the conversation can change upon return from the verb. Conditions that do not cause a state change are not noted. For example, parameter checks and state checks do not cause a state change.
- Additional information describing the use of the verb.

Most parameters supplied to and returned by APPC are hexadecimal values. To simplify coding, these values are represented by meaningful symbolic constants defined in the header file **values_c.h**, which is included by the APPC header file **appc_c.h**. For example, the *opcode* parameter of the TP_STARTED verb is the hexadecimal value represented by the symbolic constant `AP_TP_STARTED`. The file **values_c.h** also includes definitions of parameter types such as `AP_UINT16` that are used in the APPC VCBs.

It is important that you use the symbolic constant and not the hexadecimal value when setting values for supplied parameters, or when testing values of returned parameters. This is because different operating systems store these values differently in memory, so the value shown may not be in the format recognized by your system.

WINDOWS

For Windows, the constants for supplied and returned parameter values are defined in the Windows APPC header file **winappc.h**.

The notation "[MC_]verb" refers to both the mapped and basic form of an APPC verb. For example, [MC_]SEND_DATA refers to the MC_SEND_DATA and SEND_DATA verbs.

Note: The APPC VCBs contain many parameters marked as "reserved"; some of these are used internally by the Communications Server software, and others are not used in this version but may be used in future versions. Your

application must not attempt to access any of these reserved parameters; instead, it must set the entire contents of the VCB to zero to ensure that all of these parameters are zero, before it sets other parameters that are used by the verb. This ensures that Communications Server will not misinterpret any of its internally-used parameters, and also that your application will continue to work with future Communications Server versions in which these parameters may be used to provide new functions.

To set the VCB contents to zero, use `memset`:

memset(vcb, 0, sizeof(vcb));

The control verbs are described in the following order:

TP_STARTED
TP_ENDED
RECEIVE_ALLOCATE

UNIX

GET_LU_STATUS

GET_TP_PROPERTIES

UNIX

SET_TP_PROPERTIES

TP_STARTED

The TP_STARTED verb is issued by the invoking TP. It notifies APPC that the TP is starting, and specifies the local LU that it will use.

If the TP is using dependent LUs for multiple concurrent conversations, it must issue a separate TP_STARTED verb (followed by [MC_]ALLOCATE) for each conversation, to obtain a different LU for each conversation; this is because each dependent LU can support only one conversation at a time.

In response to this verb, APPC generates a TP identifier for the invoking TP. This identifier is a required parameter for subsequent APPC verbs issued by the invoking TP.

VCB Structure: TP_STARTED

UNIX

The definition of the VCB structure for the TP_STARTED verb is as follows:

```
typedef struct tp_started
{
  AP_UINT16      opcode;
```

```
  unsigned char  opext;                /* Reserved            */
  unsigned char  format;               /* Reserved            */
  AP_UINT16      primary_rc;
  AP_UINT32      secondary_rc;
  unsigned char  lu_alias[8];
  unsigned char  tp_id[8];
  unsigned char  tp_name[64];
  unsigned char  delay_start;          /* Reserved            */
  unsigned char  enable_pool;          /* Reserved            */
  unsigned char  pip_dlen;             /* Reserved            */
} TP_STARTED;
```

VCB Structure: TP_STARTED (Windows)

WINDOWS

The definition of the VCB structure for the TP_STARTED verb is as follows:

```
typedef struct tp_started
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     lu_alias[8];
  unsigned char     tp_id[8];
  unsigned char     tp_name[64];
} TP_STARTED;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode AP_TP_STARTED

lu_alias

Alias by which the local LU is known to the local TP. This name must match an LU alias established during configuration.

This parameter is an 8-byte ASCII character string. It can consist of any of the following characters:

- Uppercase letters
- Numerals 0–9
- Blanks
- Special characters $, #, %, and @

The first character of this string cannot be a blank (unless the whole string consists of blanks).

If the LU alias is shorter than eight characters, pad it on the right with ASCII blanks (0x20).

Depending on the configuration, you may be able to specify that the application uses a default local LU (check with your System Administrator); to do this, set *lu_alias* to a string of eight binary zeros. For compatibility with other APPC implementations, Communications Server

also accepts a string of eight ASCII blanks to indicate the default LU; however, new applications should use binary zeros.

tp_name
Name of the local TP. The first eight characters of this name are translated into ASCII, and used by Communications Server administration programs to identify the TP in a list of running APPC TPs.

This parameter is a 64-byte case-sensitive EBCDIC character string. The *tp_name* parameter normally consists of characters from the type-AE EBCDIC character set (unless it is the name of a service TP). These characters are as follows:

- Uppercase and lowercase letters
- Numerals 0–9
- Special characters $, #, @, and period (.)

If the TP name is fewer than 64 bytes, use EBCDIC blanks (`0x40`) to pad it on the right.

The SNA convention for naming a service TP is an exception to the normal *tp_name* parameter; the name consists of up to four characters, of which the first character is a hexadecimal byte between `0x00` and `0x3F`. The other characters are from the EBCDIC AE character set.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
`AP_OK`

tp_id Identifier for the local TP.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
`AP_PARAMETER_CHECK`

secondary_rc
Possible values are:

AP_BAD_LU_ALIAS
The value of the *lu_alias* parameter was not valid.

UNIX

AP_INVALID_FORMAT
The reserved parameter *format* was set to a nonzero value.

AP_SYNC_NOT_ALLOWED
Using the synchronous APPC entry point, the application issued this verb within a callback routine. Any verb issued from a callback routine must use the asynchronous entry point.

State Check: No state check errors occur for this verb.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
AP_COMM_SUBSYSTEM_NOT_LOADED
AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT

APPC does not return secondary return codes with these primary return codes.

State When Issued

TP_STARTED must be the first APPC verb issued by the invoking TP. Consequently, no conversations are active and no conversation state exists.

A single APPC program can issue more than one TP_STARTED verb. Each verb creates a logically different APPC TP, although they are all executing in the same process.

State Change

Not applicable (no conversations have been started, so there is no conversation state).

TP_ENDED

The TP_ENDED verb is issued by both the invoking and the invoked TPs. It notifies APPC that the TP is ending. In response to this verb, APPC frees the resources used by the TP.

If an APPC conversation is still in progress, TP_ENDED performs the function of the [MC_]DEALLOCATE verb with *dealloc_type* set to AP_ABEND (for a mapped conversation) or AP_ABEND_PROG (for a basic conversation). After this verb executes, the TP identifier and conversation identifier are no longer valid; the TP cannot issue any more APPC verbs for the conversation.

VCB Structure: TP_ENDED

UNIX

The definition of the VCB structure for the TP_ENDED verb is as follows:

```
typedef struct tp_ended
{
  AP_UINT16         opcode;
  unsigned char     opext;                /* Reserved      */
  unsigned char     format;               /* Reserved      */
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  unsigned char     type;
} TP_ENDED;
```

VCB Structure: TP_ENDED (Windows)

WINDOWS

The definition of the VCB structure for the TP_ENDED verb is as follows:

```
typedef struct tp_ended
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned char     type;
} TP_ENDED;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode AP_TP_ENDED

tp_id Identifier for the local TP.

The value of this parameter was returned by the TP_STARTED verb for the invoking TP or by the RECEIVE_ALLOCATE verb for the invoked TP.

type Specifies how to end the TP. Possible values are:

AP_SOFT
If any APPC conversations are active, APPC performs the function of the [MC_]DEALLOCATE verb for each conversation, in order to inform the partner TP that the conversation has ended. The TP_ENDED verb does not return until [MC_]DEALLOCATE has completed.

AP_HARD
APPC closes all sessions used by the TP, and TP_ENDED returns immediately.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
: AP_OK

APPC does not return a *secondary_rc* when the verb executes successfully.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
: AP_PARAMETER_CHECK

secondary_rc
: Possible values are:

AP_BAD_TP_ID
: APPC did not recognize the *tp_id* as an assigned TP identifier.

AP_BAD_TYPE
: The value of the *type* parameter was not valid.

UNIX

AP_INVALID_FORMAT
: The reserved parameter *format* was set to a nonzero value.

AP_SYNC_NOT_ALLOWED
: The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

State Check: No state check errors occur for this verb.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc
: AP_COMM_SUBSYSTEM_ABENDED
AP_COMM_SUBSYSTEM_NOT_LOADED
AP_INVALID_VERB

AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation (or conversations, if the TP is involved in more than one) can be in any state when the TP issues this verb.

State Change

After successful execution (*primary_rc* is AP_OK), there is no APPC state.

RECEIVE_ALLOCATE

The RECEIVE_ALLOCATE verb is issued by the invoked TP. It confirms that the invoked TP is ready to begin a conversation with the invoking TP, which issued the [MC_]ALLOCATE verb.

In response to this verb, APPC establishes a conversation between the two TPs, generates a TP identifier for the invoked TP, and generates a conversation identifier. These identifiers are required parameters for subsequent APPC verbs.

UNIX

The Communications Server APPC implementation provides both the standard form of the RECEIVE_ALLOCATE verb, as provided by other APPC implementations, and an extended form that enables the application to receive incoming Attaches from a particular local LU. The two forms are described together in this section, with references to “standard form” and “extended form” where appropriate.

WINDOWS

The extended form of RECEIVE_ALLOCATE is not provided on Windows.

VCB Structure: RECEIVE_ALLOCATE

UNIX

The definition of the VCB structure for the RECEIVE_ALLOCATE verb is as follows:

```
typedef struct receive_allocate
{
  AP_UINT16         opcode;
  unsigned char     opext;                  /* Reserved      */
  unsigned char     format;
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_name[64];
  unsigned char     tp_id[8];
  AP_UINT32         conv_id;
  unsigned char     sync_level;
  unsigned char     conv_type;
  unsigned char     user_id[10];
  unsigned char     lu_alias[8];
  unsigned char     plu_alias[8];
  unsigned char     mode_name[8];
  unsigned char     reserv3[2];
  AP_UINT32         conv_group_id;
  unsigned char     fqplu_name[17];
  unsigned char     pip_incoming;
  unsigned char     duplex_type;
  unsigned char     reserv4[3];
  unsigned char     password[10];
  unsigned char     reserv5[2];
  unsigned char     dload_id[8];
} RECEIVE_ALLOCATE;
```

VCB Structure: RECEIVE_ALLOCATE (Windows)

WINDOWS

The definition of the VCB structure for the RECEIVE_ALLOCATE verb is as follows:

```
typedef struct receive_allocate
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_name[64];
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     sync_level;
  unsigned char     conv_type;
  unsigned char     user_id[10];
  unsigned char     lu_alias[8];
  unsigned char     plu_alias[8];
  unsigned char     mode_name[8];
  unsigned char     reserv3[2];
  unsigned long     conv_group_id;
  unsigned char     fqplu_name[17];
  unsigned char     reserv4[5];
} RECEIVE_ALLOCATE;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode `AP_RECEIVE_ALLOCATE` (standard form)

UNIX

`AP_RECEIVE_ALLOCATE_EX` (extended form)

tp_name

Name of the local TP. APPC matches this name with the TP name specified in the incoming Attach, which is generated by the [MC_]ALLOCATE verb in the invoking TP. If the TP is to be automatically started by Communications Server, this TP name must match a TP name specified in the invokable TP data file.

This parameter is a 64-byte case-sensitive EBCDIC character string. The *tp_name* parameter normally consists of characters from the type-AE EBCDIC character set. These characters are as follows:

- Uppercase and lowercase letters
- Numerals 0–9
- Special characters $, #, @, and period (.)

If the TP name is fewer than 64 bytes, use EBCDIC blanks (`0x40`) to pad it on the right.

The SNA convention for naming a service TP is an exception to the above; the name consists of up to 4 characters, of which the first character is a hexadecimal byte between `0x00` and `0x3F`. The other characters are from the EBCDIC AE character set.

UNIX

The TP can specify that it will accept incoming Attaches for any TP name, by setting this parameter to 64 EBCDIC spaces. For more information about how Communications Server routes incoming Attaches to TPs, see "Routing for Incoming Attaches" on page 77.

lu_alias

For the standard form of RECEIVE_ALLOCATE, this parameter is reserved; set it to a null string. If the TP is automatically started by Communications Server, the entry for this TP in the invokable TP data file must not specify an LU alias.

For the extended form of RECEIVE_ALLOCATE, specify the alias of the local LU from which the TP will accept incoming Attaches. This is an ASCII character string. If the TP is automatically started by Communications Server, this LU alias must match the LU alias specified for the TP in the invokable TP data file.

To indicate that the TP will accept incoming Attaches from any local LU, set this parameter to eight ASCII spaces. If the TP is automatically started by Communications Server, the entry for this TP in the invokable TP data file must not specify an LU alias.

For more information about how Communications Server routes incoming Attaches to TPs, see "Routing for Incoming Attaches" on page 77.

dload_id
The identifier for the Attach provided by a TP server application. If the TP does not cooperate with a TP server, set this field to all zeros.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

UNIX

tp_name
If the application specified a TP name consisting of all spaces, Communications Server returns the TP name that was supplied by the invoking TP on the [MC_]ALLOCATE verb.

tp_id Identifier for the local TP.

conv_id
Conversation identifier.

This value identifies the conversation APPC has established between the two partner TPs.

sync_level
Synchronization level of the conversation.

This parameter determines whether the TPs can request confirmation of receipt of data and confirm receipt of data. Possible values are:

AP_CONFIRM_SYNC_LEVEL
The partner TPs can use confirmation processing in this conversation.

AP_SYNCPT
The TPs can use LU 6.2 Syncpoint functions in this conversation. For more information, see "Syncpoint Support" on page 23.

AP_NONE
Confirmation processing is not used in this conversation.

conv_type
Type of conversation chosen by the partner TP, using the [MC_]ALLOCATE verb. Possible values are:
AP_BASIC_CONVERSATION
AP_MAPPED_CONVERSATION

user_id If the partner TP set [MC_]ALLOCATE's *security* parameter to AP_PGM or AP_SAME, this parameter contains the user ID sent from the partner TP. The user ID is a type-AE EBCDIC character string, padded on the right with EBCDIC spaces to 10 characters if necessary. If the partner TP set [MC_]ALLOCATE's security parameter to AP_NONE, this parameter is set to 10 EBCDIC blanks.

lu_alias
Alias by which the local LU is known to the local TP. This is an ASCII character string.

plu_alias
Alias by which the partner LU (from which the incoming Allocate was received) is known to the local TP. This is an ASCII character string.

mode_name
Mode name specified by the [MC_]ALLOCATE verb in partner TP. This is the name of a set of networking characteristics defined during configuration. This name is a type-A EBCDIC character string.

conv_group_id
The conversation group identifier of the session that the new conversation uses.

fqplu_name
Fully qualified name of the partner LU.

This parameter contains the network name, an EBCDIC period, and the partner LU name. Each of the two names is an 8-byte EBCDIC character string, which can consist of characters from the type-A EBCDIC character set as follows:

- Uppercase letters
- Numerals 0–9
- Special characters $, #, and @

UNIX

pip_incoming
Specifies whether the partner TP supplied program initialization parameter (PIP) data on the [MC_]ALLOCATE request. Possible values are:

AP_YES The partner TP supplied PIP data. The local TP should issue one of the [MC_]RECEIVE verbs to receive the data; the first data record received will be the PIP data.

AP_NO The partner TP did not supply PIP data.

duplex_type
Duplex type of the new conversation. Possible values are:

```
AP_HALF_DUPLEX
AP_FULL_DUPLEX
```

password
If the partner TP set [MC_]ALLOCATE's *security* parameter to AP_PGM, this parameter contains the password specified by the partner TP on the [MC_]ALLOCATE verb. The password is a type-AE EBCDIC character string, padded on the right with EBCDIC spaces to 10 characters if necessary. If the partner TP set [MC_]ALLOCATE's *security* parameter to AP_NONE or AP_SAME, this parameter is set to 10 EBCDIC blanks.

For security reasons, Communications Server does not store the password after returning it on the RECEIVE_ALLOCATE verb. If the application needs to check this parameter, it must use the value returned on RECEIVE_ALLOCATE; the password is not returned on any subsequent verbs. The application can retrieve the user ID at any point during the conversation by issuing the GET_TP_PROPERTIES verb.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

UNIX

AP_BAD_DLOAD_ID
The value specified for the *dload_id* parameter was not recognized.

AP_INVALID_FORMAT
The *format* parameter was set to a value that was not valid.

AP_INVALID_LU_ALIAS
The *lu_alias* parameter contained a character that was not valid. (This value is returned for the extended form of RECEIVE_ALLOCATE, not for the standard form.)

UNIX

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

State Check: If the conversation is in the wrong state when the TP issues this verb, APPC returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc

AP_ALLOCATE_NOT_PENDING
APPC did not find an incoming Allocate (from the invoking TP) to match the combination of TP name, LU alias, or both supplied by the RECEIVE_ALLOCATE verb. The RECEIVE_ALLOCATE verb waited for the incoming Allocate and eventually timed out. For more information, see "Avoiding Waits" and "Routing for Incoming Attaches" on page 77.

This return code also occurs if you attempt to start a TP that is defined in the invokable TP data file as nonqueued. A nonqueued TP is started automatically by Communications Server in response to an incoming Attach; if you attempt to start it manually, the RECEIVE_ALLOCATE verb fails because no incoming Attach is waiting for the TP.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
AP_COMM_SUBSYSTEM_NOT_LOADED
AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT

APPC does not return secondary return codes with these primary return codes.

State When Issued

This must be the first APPC verb issued by the invoked TP. The initial state is Reset.

A single invoked TP can issue multiple RECEIVE_ALLOCATE verbs; each starts a logically different APPC TP, although all of them are executing in the same process.

State Change

If the verb executes successfully (*primary_rc* is AP_OK), the state changes to Receive (for a half-duplex conversation) or Send_Receive (for a full-duplex conversation).

Avoiding Waits

If the invoked TP issues a RECEIVE_ALLOCATE verb and a corresponding incoming Allocate (resulting from the [MC_]ALLOCATE verb issued by the invoking TP) is not present, the invoked TP waits until the incoming Allocate arrives or until the verb times out. The default is to wait indefinitely for an incoming Allocate; this can be overridden by configuring the TP with a timeout of 0 (zero) (RECEIVE_ALLOCATE fails unless an incoming Allocate is already

waiting), or a finite value (RECEIVE_ALLOCATE fails unless an incoming Allocate arrives within the specified time). For more information, refer to the *IBM Communications Server for Data Center Deployment on AIX or Linux Administration Guide*.

Routing for Incoming Attaches

UNIX

If the application does not specify a *dload_id* on the RECEIVE_ALLOCATE, it can use the *tp_name* and *lu_alias* parameters of RECEIVE_ALLOCATE to specify the range of incoming Attaches that it will accept. By specifying a TP name, it indicates that it will accept incoming Attaches from a partner TP only if the partner TP specified this TP name on the [MC_]ALLOCATE verb; by using the extended form of RECEIVE_ALLOCATE and specifying an LU alias, it indicates that it will accept incoming Attaches only if they arrived at a particular Communications Server local LU. In either case, the TP can specify a blank name to indicate that it accepts incoming Attaches for any TP name or from any local LU.

Communications Server matches an incoming Attach to the RECEIVE_ALLOCATE verb in the appropriate TP in the following order of precedence:

1. A TP that specifies a TP name and an LU alias, both of which match the incoming Attach.
2. A TP that specifies a TP name matching the incoming Attach but does not specify an LU alias.
3. A TP that specifies an LU alias matching the LU that received the incoming Attach but does not specify a TP name.
4. A TP that does not specify a TP name or an LU alias. Only one TP on each Communications Server computer should use this feature; if two TPs both issue RECEIVE_ALLOCATE verb with no TP name or LU alias, it is not possible to determine which TP will receive the incoming Attach.

If the TP used a blank TP name, LU alias, or both to accept a range of incoming Attaches, it can check the returned parameters on this verb to determine the TP name specified on the incoming Attach, the LU alias of the LU that received it, or both. This means that you can have a single TP to handle all incoming Attaches, which performs the appropriate processing for each of several TP names, LUs or both. If this TP receives an incoming Attach from an unrecognized or unauthorized partner TP, it can reject the new conversation if necessary by issuing the [MC_]DEALLOCATE verb with an appropriate *dealloc_type* parameter.

The TP accepting the incoming Attach may be an operator-started TP that has already issued RECEIVE_ALLOCATE, or an automatically started TP listed in the Communications Server invokable TP data file. Communications Server uses the TP name, LU alias, or both specified in this file to determine whether to start the TP in order to match the incoming Attach. For more information about the format of this file, refer to the *IBM Communications Server for Data Center Deployment on AIX or Linux Administration Guide*. The TP name, LU alias, or both specified by an automatically started TP on the RECEIVE_ALLOCATE verb must match those specified in the file to ensure that Communications Server can route the incoming Attach correctly.

GET_LU_STATUS

UNIX

This verb is provided for Syncpoint TPs, which need to check whether they have lost communications with their partner TPs so that they can resynchronize if necessary.

Note: If two or more TPs are using the same combination of local LU and partner LU, it is important that only one of them issues this verb. Communications Server maintains the zero sessions indicator for each pair of LUs independently of the TPs using them, and resets it each time this verb is issued. This means that, if the session count drops to `0` (zero) and then sessions are reactivated, and two TPs subsequently issue GET_LU_STATUS, only the first TP will be notified of the zero session count. If multiple TPs using the same LUs need to check LU and session status, they should do so using NOF verbs; refer to the *IBM Communications Server for Data Center Deployment on AIX or Linux NOF Programmer's Guide* for more information.

VCB Structure: GET_LU_STATUS

The definition of the VCB structure for the GET_LU_STATUS verb is as follows:

```
typedef struct get_lu_status
{
  AP_UINT16         opcode;
  unsigned char     opext;                  /* Reserved      */
  unsigned char     format;                 /* Reserved      */
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  unsigned char     plu_alias[8];
  AP_UINT16         active_sess;
  unsigned char     zero_sess;
  unsigned char     reserv3[7];
} GET_LU_STATUS;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode `AP_GET_LU_STATUS`

tp_id Identifier for the local TP.

The value of this parameter is returned by the TP_STARTED verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

plu_alias

Alias by which the partner LU is known to the local TP. This is an 8-byte ASCII character string.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

If the verb executes successfully, APPC returns the following parameters.

primary_rc
: AP_OK

active_sess
: Specifies the number of sessions currently active between the local LU and the specified partner LU.

zero_sess
: Specifies whether the number of active sessions between the two LUs has dropped to 0 (zero) at any time since the last GET_LU_STATUS verb was issued. Possible values are:

  AP_YES The session count has dropped to 0 (zero).

  AP_NO At least one session has been active at all times since the verb was last issued.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
: AP_PARAMETER_CHECK

secondary_rc
: Possible values are:

  AP_BAD_TP_ID
  The value of *tp_id* did not match a TP identifier assigned by APPC.

  AP_INVALID_FORMAT
  The reserved parameter *format* was set to a nonzero value.

  AP_SYNC_NOT_ALLOWED
  The application issued this verb within a callback routine using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

State Check: No state check errors occur for this verb.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible values are:

primary_rc
```
AP_COMM_SUBSYSTEM_ABENDED
AP_INVALID_VERB
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR
```

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation can be in any state except Reset when the TP issues this verb.

State Change

The conversation state does not change for this verb.

GET_TP_PROPERTIES

The GET_TP_PROPERTIES verb returns information about the attributes of the local TP and of the Logical Unit of Work (LUW) in which the TP is participating. A Logical Unit of Work is a transaction between APPC TPs to accomplish a particular task; it may involve two communicating TPs or a sequence of conversations between several TPs.

VCB Structure: GET_TP_PROPERTIES

UNIX

The definition of the VCB structure for the GET_TP_PROPERTIES verb is as follows:

```
typedef struct get_tp_properties
{
  AP_UINT16         opcode;
  unsigned char     opext;                  /* Reserved      */
  unsigned char     format;
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  unsigned char     tp_name[64];
  unsigned char     lu_alias[8];
  LUWID_OVERLAY     luw_id;
  unsigned char     fqlu_name[17];
  unsigned char     reserv3[9];
  unsigned char     verified;
  unsigned char     user_id[10];
  LUWID_OVERLAY     prot_luw_id;
} GET_TP_PROPERTIES;

typedef struct luwid_overlay
{
  unsigned char     fq_length;
  unsigned char     fq_luw_name[17];
  unsigned char     instance[6];
  unsigned char     sequence[2];
} LUWID_OVERLAY;
```

VCB Structure: GET_TP_PROPERTIES (Windows)

WINDOWS

The definition of the VCB structure for the GET_TP_PROPERTIES verb is as follows:

```
typedef struct get_tp_properties
{
  unsigned short    opcode;
  unsigned char     reserv2[2];
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned char     tp_name[64];
  unsigned char     lu_alias[8];
  unsigned char     luw_id[26];
  unsigned char     fqlu_name[17];
  unsigned char     reserv3[10];
  unsigned char     user_id[10];
} GET_TP_PROPERTIES;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode AP_GET_TP_PROPERTIES

tp_id Identifier for the local TP.

The value of this parameter is returned by the TP_STARTED verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

tp_name
TP name of the local TP, as specified on the TP_STARTED or RECEIVE_ALLOCATE verb. This is a 64-byte EBCDIC character string.

lu_alias
Alias by which the local LU is known to the local TP, as specified on the TP_STARTED or RECEIVE_ALLOCATE verb. This is an 8-byte ASCII character string.

UNIX

luw_id The unprotected Logical Unit of Work Identifier (LUWID) for the transaction in which the TP is participating. The LUWID is assigned on behalf of the TP that initiates the transaction, and enables you to correlate the different conversations that make up the transaction. The unprotected LUWID is used to correlate unprotected conversations (those with a *sync_level* of AP_NONE or AP_CONFIRM_SYNC_LEVEL); for TPs that use Syncpoint processing, there is an additional protected LUWID for conversations with a *sync_level* of AP_SYNCPT (for more information, see the *prot_luw_id* parameter).

The LUWID consists of the following parameters:

luw_id.fq_length
The length (1–17 bytes) of the fully qualified LU name associated with the Logical Unit of Work (the LU name itself is specified by the *luw_id.fq_luw_name* parameter).

luw_id.fq_luw_name
The fully qualified LU name associated with the Logical Unit of Work. This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1–8 A-string characters, an EBCDIC dot (period) character, and an LU name of 1–8 A-string characters.

luw_id.instance
The instance number associated with the Logical Unit of Work (a 6-byte binary number).

luw_id.sequence
The sequence number of the current segment of the Logical Unit of Work (a 2-byte binary number).

WINDOWS

luw_id The Logical Unit of Work Identifier (LUWID) for the transaction in which the TP is participating. This is assigned on behalf of the TP that initiates the transaction, and enables you to correlate the different conversations that make up the transaction.

The LUWID is a 26-byte string consisting of the parameters shown in Table 9:

Table 9. LUWID Parameters

Parameter	Length	Description
fq_length	1	The length (1–17 bytes) of the fully-qualified LU name associated with the Logical Unit of Work (the LU name itself is specified by the *fq_luw_name* parameter)
fq_luw_name	1–17	The fully-qualified LU name associated with the Logical Unit of Work. This name is an EBCDIC string, consisting of a network ID of 1–8 A-string characters, an EBCDIC dot (period) character, and an LU name of 1–8 A-string characters. This name is not space-padded; the *fq_length* parameter specifies the number of bytes in the name, and the *instance* parameter follows immediately after this number of bytes.
instance	6	The instance number associated with the Logical Unit of Work (a 6-byte binary number).
sequence	2	The sequence number of the current segment of the Logical Unit of Work (a 2-byte binary number); this is always set to 1.

If the *fq_length* parameter indicates that the LU name is shorter than 17 bytes, the total length of the preceding parameters will be shorter than 26 bytes; the remaining bytes are filled with EBCDIC spaces.

fqlu_name
The fully qualified LU name of the local LU associated with the TP. This

name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1–8 A-string characters, an EBCDIC dot (period) character, and an LU name of 1–8 A-string characters.

UNIX

verified Specifies whether conversation security has been verified for this conversation. Possible values are:

AP_YES Conversation security has been verified. The invoking TP supplied a user ID (returned as the *user_id* parameter on this verb), and either supplied a valid password or indicated that conversation security had already been verified.

AP_NO Conversation security has not been verified. The invoked TP does not require a user ID and password.

user_id The user ID associated with the TP. This is a 10-byte type-AE EBCDIC string, padded on the right with EBCDIC spaces if the ID is shorter than 10 bytes. The password is not returned on this verb; it is returned on the RECEIVE_ALLOCATE verb.

UNIX

prot_luw_id
The protected Logical Unit of Work Identifier (LUWID) for the transaction in which the TP is participating.

The protected LUWID is used to correlate protected conversations (those with a *sync_level* of AP_SYNCPT). It consists of the following parameters:

prot_luw_id.fq_length
The length (1–17 bytes) of the fully qualified LU name associated with the Logical Unit of Work (the LU name itself is specified by the *prot_luw_id.fq_luw_name* parameter)

prot_luw_id.fq_luw_name
The fully qualified LU name associated with the Logical Unit of Work. This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1–8 A-string characters, an EBCDIC dot (period) character, and an LU name of 1–8 A-string characters.

prot_luw_id.instance
The instance number associated with the Logical Unit of Work (a 6-byte binary number).

prot_luw_id.sequence
The sequence number of the current segment of the Logical Unit of Work (a 2-byte binary number).

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

UNIX

AP_INVALID_FORMAT
The *format* parameter was set to a value that was not valid.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

State Check: No state check errors occur for this verb.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible values are:

primary_rc
AP_COMM_SUBSYSTEM_ABENDED
AP_INVALID_VERB
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation can be in any state except Reset when the TP issues this verb.

State Change

The conversation state does not change for this verb.

SET_TP_PROPERTIES

UNIX

The SET_TP_PROPERTIES verb enables the application to set properties of the local TP, which are used when allocating new conversations for the TP. It provides access to the following properties:

- The user ID to be used when allocating a new conversation specifying "already verified" security. In general, a TP uses "already verified" security when it has been invoked by another TP specifying a valid user ID and password, and is now invoking a third TP as part of the same transaction; in this case, APPC sends the user ID from the original TP without requiring a password. Alternatively, if the TP was not invoked by another TP, APPC uses the AIX or Linux user name with which the application is running as the user ID for conversation security.

  However, if the TP obtained and verified the user ID and password by another means (for example, if it requires the user to type in a user ID and password explicitly before allocating the conversation), it needs to provide the user ID to APPC using SET_TP_PROPERTIES before invoking another TP using "already verified" security.
- Identifiers for the Logical Unit of Work in which the TP is participating. A Logical Unit of Work is a transaction between APPC TPs to accomplish a particular task; it may involve two communicating TPs, or a sequence of conversations between several TPs. There are two Logical Unit of Work Identifiers (LUWIDs) associated with the TP: the unprotected LUWID, which is used for conversations with a *sync_level* of `AP_NONE` or `AP_CONFIRM_SYNC_LEVEL`, and the protected LUWID, which is used for conversations with a *sync_level* of `AP_SYNCPT`.

VCB Structure: SET_TP_PROPERTIES

The definition of the VCB structure for the SET_TP_PROPERTIES verb is as follows:

```
typedef struct set_tp_properties
{
  AP_UINT16         opcode;
  unsigned char     opext;                /* Reserved      */
  unsigned char     format;               /* Reserved      */
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  unsigned char     set_prot_id;
  unsigned char     new_prot_id;
  LUWID_OVERLAY     prot_id;
  unsigned char     set_unprot_id;
  unsigned char     new_unprot_id;
  LUWID_OVERLAY     unprot_id;
  unsigned char     set_user_id;
  unsigned char     set_password;
  unsigned char     user_id[10];
  unsigned char     new_password[10];
} SET_TP_PROPERTIES;

typedef struct luwid_overlay
{
  unsigned char    fq_length;
```

```
  unsigned char    fq_luw_name[17];
  unsigned char    instance[6];
  unsigned char    sequence[2];
} LUWID_OVERLAY;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode AP_SET_TP_PROPERTIES

tp_id Identifier for the local TP.

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

set_prot_id
Specifies whether APPC is to modify the protected Logical Unit of Work identifier. Possible values are:

AP_YES Modify the protected LUWID for this TP.

AP_NO Leave the protected LUWID unchanged.

new_prot_id
Specifies whether APPC should generate a new protected Logical Unit of Work identifier, or to use the one specified on this verb. This parameter is reserved if *set_prod_id* is set to AP_NO. Possible values are:

AP_YES Generate a new protected LUWID.

AP_NO Set the TP's protected LUWID to the one supplied on this verb.

prot_id If *set_prot_id* is set to AP_YES and *new_prot_id* is set to AP_NO, this structure specifies the new protected LUWID for the TP; otherwise this structure is reserved. The structure contains the following parameters:

prot_id.fq_length
The length (1–17 bytes) of the fully qualified LU name associated with the Logical Unit of Work (the LU name itself is specified by the following parameter)

prot_id.fq_luw_name
The fully qualified LU name associated with the Logical Unit of Work. This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1–8 A-string characters, an EBCDIC dot (period) character, and an LU name of 1–8 A-string characters.

prot_id.instance
The instance number associated with the Logical Unit of Work (a 6-byte binary number).

prot_id.sequence
The sequence number of the current segment of the Logical Unit of Work (a 2-byte binary number).

set_unprot_id
Specifies whether APPC is to modify the unprotected Logical Unit of Work identifier. Possible values are:

AP_YES Modify the unprotected LUWID for this TP.

AP_NO Leave the unprotected LUWID unchanged.

new_unprot_id
Specifies whether APPC should generate a new unprotected Logical Unit of Work identifier, or to use the one specified on this verb. This parameter is reserved if *set_unprot_id* is set to AP_NO. Possible values are:

AP_YES Generate a new unprotected LUWID.

AP_NO Set the TP's unprotected LUWID to the one supplied on this verb.

unprot_id
If *set_unprot_id* is set to AP_YES and *new_unprot_id* is set to AP_NO, this structure specifies the new unprotected LUWID for the TP; otherwise this structure is reserved. The structure contains the following parameters:

unprot_id.fq_length
The length (1–17 bytes) of the fully qualified LU name associated with the Logical Unit of Work (the LU name itself is specified by the following parameter)

unprot_id.fq_luw_name
The fully qualified LU name associated with the Logical Unit of Work. This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1–8 A-string characters, an EBCDIC dot (period) character, and an LU name of 1–8 A-string characters.

unprot_id.instance
The instance number associated with the Logical Unit of Work (a 6-byte binary number).

unprot_id.sequence
The sequence number of the current segment of the Logical Unit of Work (a 2-byte binary number).

set_user_id
Specifies whether APPC is to modify the user ID. Possible values are:

AP_YES Modify the user ID for this TP.

AP_NO Leave the user ID unchanged.

set_password
Specifies whether APPC should modify the password associated with the *new_password* parameter. Possible values are:

AP_YES APPC should modify the password.

AP_NO APPC should not modify the password.

user_id If *set_user_id* is set to AP_YES, this parameter specifies the new user ID; otherwise it is reserved.

new_password
If *set_password* is set to AP_YES, this parameter specifies the new password; otherwise it is reserved.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

prot_id If *set_prot_id* and *new_prot_id* are both set to AP_YES, this structure specifies the new protected LUWID for the TP, as generated by APPC. The structure contains the following parameters:

prot_id.fq_length
The length (1–17 bytes) of the fully qualified LU name associated with the Logical Unit of Work (the LU name itself is specified by the *prot_id.fq_luw_name* parameter)

prot_id.fq_luw_name
The fully qualified LU name associated with the Logical Unit of Work. This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1–8 A-string characters, an EBCDIC dot (period) character, and an LU name of 1–8 A-string characters.

prot_id.instance
The instance number associated with the Logical Unit of Work (a 6-byte binary number).

prot_id.sequence
The sequence number of the current segment of the Logical Unit of Work (a 2-byte binary number).

unprot_id
If *set_unprot_id* and *new_unprot_id* are both set to AP_YES, this structure specifies the new unprotected LUWID for the TP, as generated by APPC. The structure contains the following parameters:

unprot_id.fq_length
The length (1–17 bytes) of the fully qualified LU name associated with the Logical Unit of Work (the LU name itself is specified by the *unprot_id.fq_luw_name* parameter)

unprot_id.fq_luw_name
The fully qualified LU name associated with the Logical Unit of Work. This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1–8 A-string characters, an EBCDIC dot (period) character, and an LU name of 1–8 A-string characters.

unprot_id.instance
The instance number associated with the Logical Unit of Work (a 6-byte binary number).

unprot_id.sequence
The sequence number of the current segment of the Logical Unit of Work (a 2-byte binary number).

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

AP_INVALID_FORMAT
The reserved parameter *format* was set to a nonzero value.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

State Check: No state check errors occur for this verb.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible values are:

primary_rc

```
AP_COMM_SUBSYSTEM_ABENDED
AP_INVALID_VERB
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR
```

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation can be in any state when the TP issues this verb.

State Change

The conversation state does not change for this verb.

Usage and Restrictions

For TPs that use Syncpoint functions, when the local application changes the protected LUWID, the Syncpoint Manager is responsible for sending the appropriate PS header to the partner application to inform it of the new protected LUWID. Similarly, when the Syncpoint Manager receives a PS header containing a new protected LUWID, it must issue SET_TP_PROPERTIES to inform the local LU of the new LUWID.

Chapter 4. APPC Conversation Verbs

This chapter contains a description of each APPC conversation verb. The following information is provided for each verb:

- Definition of the verb.
- Structure defining the verb control block (VCB) used by the verb. The structure is defined in the APPC header file **/usr/include/sna/appc_c.h**(AIX), **/opt/ibm/sna/include/appc_c.h**(Linux), or **sdk/winappc.h** (Windows). Parameters beginning with *reserv* are reserved.
- Parameters (VCB fields) supplied to and returned by APPC. For each parameter, the following information is provided:
  - Description
  - Possible values
  - Additional information
- Conversation state or states in which the verb can be issued.
- State or states to which the conversation can change upon return from the verb. Conditions that do not cause a state change are not noted. For example, parameter checks and state checks do not cause a state change.
- Additional information describing the use of the verb.

Most parameters supplied to and returned by APPC are hexadecimal values. To simplify coding, these values are represented by meaningful symbolic constants defined in the header file **values_c.h**, which is included by the APPC header file **appc_c.h**. For example, the *opcode* parameter of the MC_SEND_DATA verb is the hexadecimal value represented by the symbolic constant `AP_M_SEND_DATA`.

It is important that you use the symbolic constant and not the hexadecimal value when setting values for supplied parameters, or when testing values of returned parameters. This is because different operating systems store these values differently in memory, so the value shown may not be in the format recognized by your system.

WINDOWS

For Windows, the constants for supplied and returned parameter values are defined in the Windows APPC header file **winappc.h**.

The notation "[MC_]verb" refers to both the mapped and basic form of an APPC verb. For example, [MC_]SEND_DATA refers to the MC_SEND_DATA and SEND_DATA verbs.

Note: The APPC VCBs contain many parameters marked as "reserved"; some of these are used internally by the Communications Server software, and others are not used in this version but may be used in future versions. Your application must not attempt to access any of these reserved parameters; instead, it must set the entire contents of the VCB to zero to ensure that all

of these parameters are zero, before it sets other parameters that are used by the verb. This ensures that Communications Server will not misinterpret any of its internally-used parameters, and also that your application will continue to work with future Communications Server versions in which these parameters may be used to provide new functions.

To set the VCB contents to zero, use `memset`:

memset(vcb, 0, sizeof(vcb));

The conversation verbs are described in the following order:

GET_TYPE

CANCEL_CONVERSATION

[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_]RECEIVE_EXPEDITED_DATA

[MC_]REQUEST_TO_SEND

[MC_]SEND_CONVERSATION

[MC_]SEND_DATA

[MC_]SEND_ERROR

[MC_]SEND_EXPEDITED_DATA

[MC_]TEST_RTS

[MC_]TEST_RTS_AND_POST

GET_TYPE

The GET_TYPE verb returns the conversation type (basic or mapped) of a particular conversation, and whether the conversation operates in full-duplex or half-duplex mode.

With this information, the TP can determine the correct verbs to issue on this conversation.

VCB Structure: GET_TYPE

UNIX

The definition of the VCB structure for the GET_TYPE verb is as follows:

```
typedef struct get_type
{
  AP_UINT16         opcode;
  unsigned char     opext;                  /* Reserved     */
  unsigned char     format;
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  AP_UINT32         conv_id;
  unsigned char     conv_type;
  unsigned char     duplex_type;
} GET_TYPE;
```

VCB Structure: GET_TYPE (Windows)

WINDOWS

The definition of the VCB structure for the GET_TYPE verb is as follows:

```
typedef struct get_type
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     conv_type;
} GET_TYPE;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode AP_GET_TYPE

format If you are building a new APPC application, or recompiling an existing APPC application with the current APPC header file, you must set this parameter to 1. (Existing applications built with earlier versions of the header file, in which this parameter was reserved, will still operate unchanged and there is no need to rebuild them.)

tp_id Identifier for the local TP.

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

conv_id

Identifier for the conversation this TP is inquiring about.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc

AP_OK

conv_type

Conversation type of the conversation identified by *conv_id*.

Possible values are:

AP_BASIC_CONVERSATION
AP_MAPPED_CONVERSATION

duplex_type

Duplex type of the conversation identified by *conv_id*.

Possible values are:

AP_HALF_DUPLEX
AP_FULL_DUPLEX

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_BAD_CONV_ID

The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID

The value of *tp_id* did not match a TP identifier assigned by APPC.

UNIX

AP_INVALID_FORMAT

The *format* parameter was set to a value that was not valid.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

State Check: No state check errors occur for this verb.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
AP_INVALID_VERB
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation can be in any state except Reset when the TP issues this verb.

State Change

The conversation state does not change for this verb.

CANCEL_CONVERSATION

The CANCEL_CONVERSATION verb deallocates a conversation between two TPs. It is equivalent to the MC_DEALLOCATE or DEALLOCATE verb with the *dealloc_type* parameter set to AP_ABEND or one of the AP_ABEND_* values, with the following differences:

- MC_DEALLOCATE or DEALLOCATE cannot be used while another verb is in progress on this conversation; CANCEL_CONVERSATION can be used, and will cancel the outstanding verb.
- DEALLOCATE (but not MC_DEALLOCATE) optionally includes log data that is written to the local error log; CANCEL_CONVERSATION does not.

The results of any outstanding verbs on the conversation are undefined, and will not be returned to the application. For example, if you issue

CANCEL_CONVERSATION while [MC_]SEND_DATA is outstanding, you cannot determine whether any of the data has been sent to the partner TP.

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

VCB Structure: CANCEL_CONVERSATION

UNIX

The definition of the VCB structure for the CANCEL_CONVERSATION verb is as follows:

```
typedef struct cancel_conversation
{
  AP_UINT16         opcode;
  unsigned char     opext;
  unsigned char     format;
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  AP_UINT32         conv_id;
} CANCEL_CONVERSATION;
```

VCB Structure: CANCEL_CONVERSATION (Windows)

WINDOWS

The definition of the VCB structure for the CANCEL_CONVERSATION verb is as follows:

```
typedef struct cancel_conversation
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     format;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
} CANCEL_CONVERSATION;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode AP_CANCEL_CONVERSATION

opext If the verb is being issued on a full-duplex conversation or is being issued as a non-blocking verb, specify one or both of the following values (combined using a logical OR). Otherwise, this parameter is reserved.

AP_FULL_DUPLEX_CONVERSATION
The verb is being issued on a full-duplex conversation.

AP_NON_BLOCKING
The verb is being issued as a non-blocking verb.

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

UNIX

AP_INVALID_FORMAT
The *format* parameter was set to a value that was not valid.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc
AP_ALLOCATION_ERROR

secondary_rc

AP_ALLOCATION_FAILURE_NO_RETRY
AP_ALLOCATION_FAILURE_RETRY
AP_CONVERSATION_TYPE_MISMATCH
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_SECURITY_NOT_VALID
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_TP_NAME_NOT_RECOGNIZED
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_SEC_BAD_PROTOCOL_VIOLATION
AP_SEC_BAD_PASSWORD_EXPIRED
AP_SEC_BAD_PASSWORD_INVALID
AP_SEC_BAD_USERID_REVOKED
AP_SEC_BAD_USERID_INVALID
AP_SEC_BAD_USERID_MISSING
AP_SEC_BAD_PASSWORD_MISSING
AP_SEC_BAD_UID_NOT_DEFD_TO_GRP
AP_SEC_BAD_UNAUTHRZD_AT_RLU
AP_SEC_BAD_UNAUTHRZD_FROM_LLU
AP_SEC_BAD_UNAUTHRZD_TO_TP
AP_SEC_BAD_INSTALL_EXIT_FAILED
AP_SEC_BAD_PROCESSING_FAILURE

UNIX

primary_rc

AP_BACKED_OUT

secondary_rc

AP_BO_NO_RESYNC
AP_BO_RESYNC

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_CONVERSATION_TYPE_MIXED
AP_DEALLOC_ABEND
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_DUPLEX_TYPE_MIXED
AP_PROG_ERROR_PURGING
AP_INVALID_VERB
AP_SVC_ERROR_PURGING
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_COMM_SUBSYSTEM_NOT_LOADED

```
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT
```

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation can be in any state except Reset when the TP issues the CANCEL_CONVERSATION verb.

State Change

After the CANCEL_CONVERSATION verb has completed, the conversation is in Reset state.

MC_ALLOCATE and ALLOCATE

The MC_ALLOCATE or ALLOCATE verb is issued by the invoking TP. 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 TP and the invoked TP.

The MC_ALLOCATE verb establishes a mapped conversation. The ALLOCATE verb can establish either a basic or mapped conversation. The use of the ALLOCATE verb to establish a mapped conversation enables the TP to use basic conversation verbs to communicate with a mapped-conversation partner TP.

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

The [MC_]ALLOCATE request will not usually be sent to the partner LU immediately; it will be queued at the local LU until a full buffer can be sent. This means that errors in allocating a conversation are usually not reported on the [MC_]ALLOCATE verb but on a subsequent verb.

VCB Structure: MC_ALLOCATE

UNIX

The definition of the VCB structure for the MC_ALLOCATE verb is as follows:

```
typedef struct mc_allocate
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  unsigned char    reserv3;
  unsigned char    sync_level;
  unsigned char    reserv4[2];
  unsigned char    rtn_ctl;
  unsigned char    duplex_type;
```

```
  AP_UINT32        conv_group_id;
  AP_UINT32        sense_data;
  unsigned char    plu_alias[8];
  unsigned char    mode_name[8];
  unsigned char    tp_name[64];
  unsigned char    security;
  unsigned char    reserv6[11];
  unsigned char    pwd[10];
  unsigned char    user_id[10];
  AP_UINT16        pip_dlen;
  unsigned char    *pip_dptr;
  unsigned char    reserv6a;
  unsigned char    fqplu_name[17];
  unsigned char    reserv7[8];
} MC_ALLOCATE;
```

VCB Structure: ALLOCATE

The definition of the VCB structure for the ALLOCATE verb is as follows:

```
typedef struct allocate
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  unsigned char    conv_type;
  unsigned char    sync_level;
  unsigned char    reserv3[2];
  unsigned char    rtn_ctl;
  unsigned char    duplex_type;
  AP_UINT32        conv_group_id;
  AP_UINT32        sense_data;
  unsigned char    plu_alias[8];
  unsigned char    mode_name[8];
  unsigned char    tp_name[64];
  unsigned char    security;
  unsigned char    reserv5[11];
  unsigned char    pwd[10];
  unsigned char    user_id[10];
  AP_UINT16        pip_dlen;
  unsigned char    *pip_dptr;
  unsigned char    reserv5a;
  unsigned char    fqplu_name[17];
  unsigned char    reserv6[8];
} ALLOCATE;
```

VCB Structure: MC_ALLOCATE (Windows)

WINDOWS

The definition of the VCB structure for the MC_ALLOCATE verb is as follows:

```
typedef struct mc_allocate
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     reserv3;
```

```
  unsigned char     sync_level;
  unsigned char     reserv4[2];
  unsigned char     rtn_ctl;
  unsigned char     reserv5;
  unsigned long     conv_group_id;
  unsigned long     sense_data;
  unsigned char     plu_alias[8];
  unsigned char     mode_name[8];
  unsigned char     tp_name[64];
  unsigned char     security;
  unsigned char     reserv6[11];
  unsigned char     pwd[10];
  unsigned char     user_id[10];
  unsigned short    pip_dlen;
  unsigned char far *pip_dptr;
  unsigned char     reserv7;
  unsigned char     fqplu_name[17];
  unsigned char     reserv8[8];
} MC_ALLOCATE;
```

VCB Structure: ALLOCATE (Windows)

The definition of the VCB structure for the ALLOCATE verb is as follows:

```
typedef struct allocate
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     conv_type;
  unsigned char     sync_level;
  unsigned char     reserv3[2];
  unsigned char     rtn_ctl;
  unsigned char     reserv4;
  unsigned long     conv_group_id;
  unsigned long     sense_data;
  unsigned char     plu_alias[8];
  unsigned char     mode_name[8];
  unsigned char     tp_name[64];
  unsigned char     security;
  unsigned char     reserv5[11];
  unsigned char     pwd[10];
  unsigned char     user_id[10];
  unsigned short    pip_dlen;
  unsigned char far *pip_dptr;
  unsigned char     reserv7;
  unsigned char     fqplu_name[17];
  unsigned char     reserv8[8];
} ALLOCATE;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_ALLOCATE
For the MC_ALLOCATE verb.

AP_B_ALLOCATE
For the ALLOCATE verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_ALLOCATE verb.

AP_BASIC_CONVERSATION
For the ALLOCATE verb.

If the verb is being issued as a non-blocking verb, combine the value above (using a logical OR) with the value AP_NON_BLOCKING.

tp_id Identifier for the local TP.

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

conv_type
Type of conversation to allocate. This parameter is used only by the ALLOCATE verb.

Possible values are:
```
AP_BASIC_CONVERSATION
AP_MAPPED_CONVERSATION
```

If the ALLOCATE verb establishes a mapped conversation, the local TP 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 TP can issue basic-conversation verbs and provide the mapping layer, or it can use mapped-conversation verbs (if the implementation of APPC the partner TP is using supports mapped-conversation verbs). For further information, refer to the IBM publication *Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2.*

sync_level
Synchronization level of the conversation.

This parameter determines whether the TPs can request confirmation of receipt of data and confirm receipt of data. Possible values are:

AP_NONE
Confirmation processing will not be used in this conversation.

AP_CONFIRM_SYNC_LEVEL
The TPs can use confirmation processing in this conversation. This value can be used only in a half-duplex conversation; confirmation processing is not supported in a full-duplex conversation.

UNIX

AP_SYNCPT
The TPs can use LU 6.2 Syncpoint functions in this conversation. Set this value only if you have a Syncpoint Manager (SPM) and Conversation Protected Resource Manager (C-PRM) in addition to the standard Communications Server product. For more information see “Syncpoint Support” on page 23.

rtn_ctl Specifies when the local LU acting on a session request from the local TP is to return control to the local TP. For information about sessions, see "LU-to-LU Sessions" on page 57. Whatever the value of this parameter, the LU returns control to the TP immediately if it encounters certain errors such as a zero session limit (which mean that a session will never be allocated).

Possible values are:

AP_IMMEDIATE

- If the *auto_act* parameter of the DEFINE_MODE verb or the **define_mode** command is set to `0` (zero), Communications Server does not attempt to activate a session or sessions. If a contention-winner session is immediately available (active and not being used by another conversation), the LU allocates this conversation to it and returns control to the TP immediately. If a contention-winner session is not immediately available, control is returned to the TP immediately with a *primary_rc* of `AP_UNSUCCESSFUL`.
- If the *auto_act* parameter of the DEFINE_MODE verb or the **define_mode** command is set to any other value, Communications Server will attempt to activate a session or sessions.

For more information, refer to the description of the DEFINE_MODE verb in the *IBM Communications Server for Data Center Deployment on AIX or Linux NOF Programmer's Guide* or **define_mode** command in the *IBM Communications Server for Data Center Deployment on AIX or Linux Administration Command Reference*.

AP_WHEN_SESSION_ALLOCATED

If a session is immediately available (active and not being used by another conversation), the LU allocates this conversation to it. If a session is not immediately available but one can be activated, the LU activates it and allocates the conversation to it; if it cannot activate a session, it waits for one to become free.

AP_WHEN_SESSION_FREE

If a session is immediately available (active and not being used by another conversation), the LU allocates this conversation to it. If a session is not immediately available but one can be activated, the LU activates it and allocates the conversation to it. If no active session is free and another session cannot be activated, control is returned to the TP with the primary return code `AP_ALLOCATION_ERROR` and secondary return code `AP_ALLOCATION_FAILURE_RETRY`. This is similar to `AP_WHEN_SESSION_ALLOCATED` except that the LU will not wait for a session to become free.

AP_WHEN_CONWINNER_ALLOC

As for `AP_WHEN_SESSION_ALLOCATED`, except that the LU always allocates the conversation to a contention-winner session; it will not use a contention-loser session.

AP_WHEN_CONLOSER_ALLOC

As for `AP_WHEN_SESSION_ALLOCATED`, except that the LU always allocates the conversation to a contention-loser session; it will not use a contention-winner session.

AP_WHEN_CONV_GROUP_ALLOC
Use this value if you want the new conversation to use the same session as a previous conversation; set the *conv_group_id* parameter to the conversation group ID of the previous conversation, which was returned on the [MC_]ALLOCATE or RECEIVE_ALLOCATE verb.

If the session identified by the *conv_group_id* parameter is immediately available (active and not being used by another conversation), the LU allocates this conversation to it and returns control to the TP immediately. If the session is being used by another conversation, the LU waits for it to become free. If the session is no longer active, control is returned to the TP with the primary return code AP_ALLOCATION_ERROR and secondary return code AP_ALLOCATION_FAILURE_NO_RETRY.

duplex_type
Duplex type of the new conversation. See "Half-Duplex and Full-Duplex Conversations" on page 4 for more details of the differences between full-duplex and half-duplex conversations.

Possible values are:

AP_HALF_DUPLEX
Half-duplex conversation.

AP_FULL_DUPLEX
Full-duplex conversation.

conv_group_id
Conversation group ID of the requested session for the conversation. This parameter is used only if *rtn_ctl* is set to AP_WHEN_CONV_GROUP_ALLOC; set it to binary zeros for any other value of *rtn_ctl*.

plu_alias
Alias by which the partner LU is known to the local TP.

This parameter is an 8-byte ASCII character string, padded on the right with ASCII blanks (0x20) if the alias is shorter than eight characters. It can consist of any of the following characters:

- Uppercase letters
- Numerals 0–9
- Blanks
- Special characters $, #, %, and @

The first character of this string cannot be a blank.

To identify the LU by its LU name instead of its LU alias, set this parameter to eight binary zeros, and specify the LU name in the *fqplu_name* parameter.

mode_name
Name of a set of networking characteristics defined during configuration.

This parameter is an 8-byte EBCDIC character string. It can consist of characters from the type-A EBCDIC character set. These characters are as follows:

- Uppercase letters
- Numerals 0–9
- Special characters $, #, and @

The first character in the string must be an uppercase letter, or can be # for one of the SNA-defined modes such as #INTER. For information about SNA-defined modes, see the *IBM Communications Server for Data Center Deployment on AIX or Linux Administration Guide*. If the mode name is fewer than eight characters long, pad it on the right with EBCDIC blanks (0x40).

A mode name can also be all EBCDIC blanks (0x40).

In a mapped conversation, the name cannot be SNASVCMG (a reserved mode name used internally by APPC). Using this name in a basic conversation is not recommended.

If the specified mode name does not match either an SNA-defined mode or a mode defined in the Communications Server configuration, Communications Server creates a new mode based on the default specified in the configuration (or on the SNA-defined mode with a blank mode name, if no default mode is defined).

tp_name

Name of the invoked TP.

The value of *tp_name* specified by the [MC_]ALLOCATE verb in the invoking TP must match the value of *tp_name* specified by the RECEIVE_ALLOCATE verb in the invoked TP.

This parameter is a 64-byte EBCDIC character string; it is case-sensitive. The *tp_name* parameter normally consists of characters from the type-AE EBCDIC character set (except when naming a service TP). These characters are as follows:

- Uppercase and lowercase letters
- Numerals 0–9
- Special characters $, #, @, and period (.)

If the TP name is fewer than 64 bytes, use EBCDIC blanks (0x40) to pad it on the right.

The SNA convention for naming a service TP is an exception to the above; the name consists of up to four characters, of which the first character is a hexadecimal byte between 0x00 and 0x3F. The other characters are from the EBCDIC AE character set.

security

Specifies the information the partner LU requires in order to validate access to the invoked TP.

Based on the conversation security established for the invoked TP during configuration, use one of the following values:

AP_NONE
The invoked TP does not use conversation security. (If you use this value, the invoked TP must be configured not to use conversation security.)

AP_PGM The invoked TP uses conversation security and thus requires a user ID and password. Supply this information through the *user_id* and *pwd* parameters.

AP_PGM_STRONG
The invoked TP uses conversation security and thus requires a user ID and password. In addition, setting AP_PGM_STRONG stipulates that

Communications Server encrypts the password when sending it across the network. Supply the user ID and password through the *user_id* and *pwd* parameters.

AP_SAME
Use this value when your TP was invoked by another TP, using a valid user ID and password, and is now invoking a third TP that also requires conversation security. (The situation in which one TP invokes a second TP which then invokes a third TP is illustrated in "Multiple Conversations" on page 3.) This value tells the third TP (the invoked TP) that conversation security has already been verified for the first invoking TP.

If you use this value, the *tp_id* supplied on this [MC_]ALLOCATE verb must be the same as the one that was returned on the RECEIVE_ALLOCATE verb when this TP was invoked.

UNIX

This value can also be used if your TP was not invoked by another TP, but has obtained and verified the appropriate security information by another means (for example from the AIX or Linux user name and password supplied during logon). In this case, APPC uses the AIX or Linux user name with which the application is running, truncated to 10 characters if necessary, as the user ID for conversation security; ensure that this name consists of valid AE-string characters (see the description of the *user_id* parameter for more information) and is a valid user name for the TP being invoked.

If the TP has obtained the security information by another means (for example by requesting the user to type in a valid user ID and password before allocating the conversation), it should use SET_TP_PROPERTIES to specify this user ID to APPC before issuing [MC_]ALLOCATE.

pwd Password associated with *user_id*.

This parameter is required only if the *security* parameter is set to `AP_PGM` or `AP_PGM_STRONG`; otherwise it is reserved.

The *pwd* and *user_id* parameters must match a user ID/password pair configured on the computer where the invoked TP is located.

This parameter is a 10-byte EBCDIC character string; it is case-sensitive. The *pwd* parameter can consist of characters from the type-AE EBCDIC character set. These characters are as follows:

- Uppercase and lowercase letters
- Numerals 0–9
- Special characters $, #, @, and period (.)

If the password is fewer than 10 bytes, use EBCDIC blanks (`0x40`) to pad it on the right.

user_id User ID required to access the partner TP.

This parameter is required only if the *security* parameter is set to `AP_PGM` or `AP_PGM_STRONG`; otherwise it is reserved.

The *pwd* and *user_id* parameters must match a user ID/password pair configured on the computer where the invoked TP is located.

This parameter is a 10-byte EBCDIC character string; it is case-sensitive. The *user_id* parameter can consist of characters from the type-AE EBCDIC character set. These characters are as follows:

- Uppercase and lowercase letters
- Numerals 0–9
- Special characters $, #, @, and period (.)

If the user ID is fewer than 10 bytes, use EBCDIC blanks (`0x40`) to pad it on the right.

pip_dlen

Length of the program initialization parameters (PIP) to be passed to the partner TP. The range for this value is `0`–`32,767`.

Not all APPC implementations support PIP data. Set *pip_dlen* to `0` (zero) if the partner TP is using an implementation of APPC that does not support PIP data, or if the partner is a CPI-C application.

pip_dptr

Address of buffer containing PIP data.

Use this parameter only if *pip_dlen* is greater than `0` (zero).

PIP data can consist of initialization parameters or environment setup information required by a partner TP or remote operating system. The PIP data must follow the General Data Stream format. For further information, refer to the IBM publication *Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*.

WINDOWS

The PIP data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

fqplu_name

Fully qualified LU name of the partner LU. This parameter is used only if *plu_alias* is set to zeros.

This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces, containing one of the following:

- A network ID of 1–8 A-string characters, an EBCDIC dot (period) character, and an LU name of 1–8 A-string characters
- An LU name of 1–8 A-string characters (without the network ID or the EBCDIC dot)

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

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

conv_group_id
Conversation group ID of the session used by the conversation.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_CONV_TYPE
(Returned for basic-conversation ALLOCATE only) The value specified for *conv_type* was not valid.

AP_BAD_DUPLEX_TYPE
The value specified for *duplex_type* was not valid.

AP_BAD_PARTNER_LU_ALIAS
One of the following has occurred:

- The *plu_alias* parameter did not match any defined partner LU alias.
- The value specified for *fqplu_name* was not valid.

AP_BAD_RETURN_CONTROL
The value specified for *rtn_ctl* was not valid.

AP_BAD_SECURITY
The value specified for security was not valid.

AP_BAD_SYNC_LEVEL
The value specified for *sync_level* was not valid.

AP_BAD_TP_ID
The value specified for *tp_id* was not valid.

WINDOWS

AP_INVALID_DATA_SEGMENT
The PIP data was longer than the allocated data segment, or the address of the PIP data buffer was incorrect.

AP_CONFIRM_INVALID_FOR_FDX
The *sync_level* parameter was set to AP_CONFIRM_SYNC_LEVEL in a full-duplex conversation. This value can be used only in a half-duplex conversation.

AP_NO_USE_OF_SNASVCMG
(Returned for MC_ALLOCATE only) SNASVCMG is not a valid value for *mode_name*.

AP_PIP_LEN_INCORRECT
The value of *pip_dlen* was greater than 32,767.

AP_UNKNOWN_PARTNER_MODE
The value specified for *mode_name* was not valid.

UNIX

AP_INVALID_FORMAT
The *format* parameter was set to a value that was not valid.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

State Check: No state check errors occur for this verb.

Session Not Available: Depending on the value specified for *rtn_ctl*, APPC may return the following parameter:

primary_rc

AP_UNSUCCESSFUL
The supplied parameter *rtn_ctl* specified immediate (AP_IMMEDIATE) return of control to the TP, and the local LU did not have an available contention-winner session.

Allocation Error: If Communications Server cannot allocate the conversation, APPC returns the following parameters:

primary_rc
AP_ALLOCATION_ERROR

secondary_rc
Possible values are:

AP_ALLOCATION_FAILURE_NO_RETRY
The conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. 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.

This value is also returned if the session corresponding to the requested conversation group ID is no longer active.

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 allow the condition to clear.

AP_FDX_NOT_SUPPORTED_BY_LU
The *duplex_type* parameter was set to AP_FULL_DUPLEX, but the LU used by this TP does not support full-duplex operation.

For information about these secondary return codes, see Appendix B, "Common Return Codes," on page 273.

AP_SEC_BAD_PROTOCOL_VIOLATION
AP_SEC_BAD_PASSWORD_EXPIRED
AP_SEC_BAD_PASSWORD_INVALID
AP_SEC_BAD_USERID_REVOKED
AP_SEC_BAD_USERID_INVALID
AP_SEC_BAD_USERID_MISSING
AP_SEC_BAD_PASSWORD_MISSING
AP_SEC_BAD_UID_NOT_DEFD_TO_GRP
AP_SEC_BAD_UNAUTHRZD_AT_RLU
AP_SEC_BAD_UNAUTHRZD_FROM_LLU
AP_SEC_BAD_UNAUTHRZD_TO_TP
AP_SEC_BAD_INSTALL_EXIT_FAILED
AP_SEC_BAD_PROCESSING_FAILURE

sense_data
SNA sense data giving more information about the cause of the allocation failure.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
AP_INVALID_VERB
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation state is Reset when the TP issues this verb. It can be issued during an existing conversation which is in any state, since it always implies the start of a new conversation which is in Reset state.

State Change

Upon successful execution of this verb (*primary_rc* is AP_OK), the state of the new conversation is Send (for a half-duplex conversation) or Send_Receive (for a full-duplex conversation). If the verb fails, the state remains unchanged.

EBCDIC-ASCII, ASCII-EBCDIC Translation

Several parameters of the [MC_]ALLOCATE verb are EBCDIC or ASCII strings. A TP can use the Common Service Verb CONVERT to translate a string from one character set to the other. For further information, refer to the *IBM Communications Server for Data Center Deployment on AIX or Linux CSV Programmer's Guide*.

Immediate Allocation

To ensure that the conversation with the partner is started immediately, the invoking TP can issue the [MC_]FLUSH or [MC_]CONFIRM verb immediately after the [MC_]ALLOCATE verb. ([MC_]CONFIRM applies to half-duplex conversations only.) Otherwise, the [MC_]ALLOCATE request accumulates with other data in the local LU's send buffer until the buffer is full.

Confirming the Allocation (half-duplex conversation only)

By issuing the [MC_]CONFIRM verb after [MC_]ALLOCATE, the invoking TP can immediately determine whether the allocation was successful (if *sync_level* is set to AP_CONFIRM_SYNC_LEVEL).

MC_CONFIRM and CONFIRM

The MC_CONFIRM or CONFIRM verb sends the contents of the local LU's send buffer and a confirmation request to the partner TP.

Note: This verb can be used only in a half-duplex conversation; it is not valid in a full-duplex conversation.

In response to the [MC_]CONFIRM verb, the partner TP normally issues the [MC_]CONFIRMED verb to confirm that it has received the data without error. (If the partner TP encounters an error, it issues the [MC_]SEND_ERROR verb or abnormally deallocates the conversation.)

The TP 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: MC_CONFIRM

UNIX

The definition of the VCB structure for the MC_CONFIRM verb is as follows:

```
typedef struct mc_confirm
{
  AP_UINT16         opcode;
  unsigned char     opext;
  unsigned char     format;
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
```

```
  AP_UINT32         conv_id;
  unsigned char     rts_rcvd;
  unsigned char     expd_rcvd;
} MC_CONFIRM;
```

VCB Structure: CONFIRM

The definition of the VCB structure for the CONFIRM verb is as follows:

```
typedef struct confirm
{
  AP_UINT16         opcode;
  unsigned char     opext;
  unsigned char     format;
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  AP_UINT32         conv_id;
  unsigned char     rts_rcvd;
  unsigned char     expd_rcvd;
} CONFIRM;
```

VCB Structure: MC_CONFIRM (Windows)

WINDOWS

The definition of the VCB structure for the MC_CONFIRM verb is as follows:

```
typedef struct mc_confirm
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     rts_rcvd;
} MC_CONFIRM;
```

VCB Structure: CONFIRM (Windows)

The definition of the VCB structure for the CONFIRM verb is as follows:

```
typedef struct confirm
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     rts_rcvd;
} CONFIRM;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_CONFIRM
For the MC_CONFIRM verb.

AP_B_CONFIRM
For the CONFIRM verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_CONFIRM verb.

AP_BASIC_CONVERSATION
For the CONFIRM verb.

If the verb is being issued as a non-blocking verb, combine the value above (using a logical OR) with the value AP_NON_BLOCKING.

format If you are building a new APPC application, or recompiling an existing APPC application with the current APPC header file, you must set this parameter to 1. (Existing applications built with earlier versions of the header file, in which this parameter was reserved, will still operate unchanged and there is no need to rebuild them.)

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

rts_rcvd
Request-to-send-received indicator. This parameter applies only in a half-duplex conversation; it is not used in a full-duplex conversation.

Possible values are:

AP_YES The partner TP has issued the [MC_]REQUEST_TO_SEND verb, which requests that the local TP change the conversation to Receive state. To change to Receive state, the local TP can use the [MC_]PREPARE_TO_RECEIVE, [MC_]RECEIVE_AND_WAIT, or [MC_]RECEIVE_AND_POST verb.

AP_NO The partner TP has not issued the [MC_]REQUEST_TO_SEND verb.

expd_rcvd
Expedited data indicator.

Possible values are:

AP_YES The partner TP has sent expedited data that the local TP has not yet received. To receive this data, the local TP can use the [MC_]RECEIVE_EXPEDITED_DATA verb.

This indicator can be set on a number of APPC verbs. It continues to be set on subsequent verbs until the local TP issues the [MC_]RECEIVE_EXPEDITED_DATA verb to receive the data.

AP_NO There is no expedited data waiting to be received.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

AP_CONFIRM_INVALID_FOR_FDX
The local TP attempted to use the [MC_]CONFIRM verb in a full-duplex conversation. This verb can be used only in a half-duplex conversation.

AP_CONFIRM_ON_SYNC_LEVEL_NONE
The local TP attempted to use the [MC_]CONFIRM verb in a conversation with a synchronization level of AP_NONE. The synchronization level, established by the [MC_]ALLOCATE verb, must be AP_CONFIRM_SYNC_LEVEL.

UNIX

AP_INVALID_FORMAT
The *format* parameter was set to a value that was not valid.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

State Check: If the conversation is in the wrong state when the TP issues this verb, APPC returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
Possible values are:

AP_CONFIRM_BAD_STATE
The conversation was not in Send or Send_Pending state.

AP_CONFIRM_NOT_LL_BDY
(Returned for basic-conversation CONFIRM only) The conversation for the local TP was in Send state, and the local TP did not finish sending a logical record.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc
AP_ALLOCATION_ERROR

secondary_rc
AP_ALLOCATION_FAILURE_NO_RETRY
AP_ALLOCATION_FAILURE_RETRY
AP_CONVERSATION_TYPE_MISMATCH
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_SECURITY_NOT_VALID
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_TP_NAME_NOT_RECOGNIZED
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_SEC_BAD_PROTOCOL_VIOLATION
AP_SEC_BAD_PASSWORD_EXPIRED
AP_SEC_BAD_PASSWORD_INVALID
AP_SEC_BAD_USERID_REVOKED
AP_SEC_BAD_USERID_INVALID
AP_SEC_BAD_USERID_MISSING
AP_SEC_BAD_PASSWORD_MISSING
AP_SEC_BAD_UID_NOT_DEFD_TO_GRP
AP_SEC_BAD_UNAUTHRZD_AT_RLU
AP_SEC_BAD_UNAUTHRZD_FROM_LLU
AP_SEC_BAD_UNAUTHRZD_TO_TP
AP_SEC_BAD_INSTALL_EXIT_FAILED
AP_SEC_BAD_PROCESSING_FAILURE

UNIX

primary_rc
AP_BACKED_OUT

secondary_rc
AP_BO_NO_RESYNC
AP_BO_RESYNC

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_CONVERSATION_TYPE_MIXED
AP_PROG_ERROR_PURGING
AP_INVALID_VERB
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT

APPC does not return secondary return codes with these primary return codes.

The following primary return code is returned by the MC_CONFIRM verb:

primary_rc

AP_DEALLOC_ABEND

APPC does not return a secondary return code with this primary return code.

The following primary return codes are returned by the CONFIRM verb:

primary_rc

AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_SVC_ERROR_PURGING

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation must be in Send or Send_Pending state when the TP issues this verb.

State Change

State changes, summarized in the following table, are based on the value of the *primary_rc* parameter.

primary_rc	**New state**
AP_OK	Send
AP_PARAMETER_CHECK	No change
AP_STATE_CHECK	
AP_CONVERSATION_TYPE_MIXED	
AP_INVALID_VERB	
AP_INVALID_VERB_SEGMENT	
AP_STACK_TOO_SMALL	
AP_TP_BUSY	
AP_UNEXPECTED_DOS_ERROR	

primary_rc	**New state**
AP_PROG_ERROR-PURGING AP_SVC_ERROR_PURGING	Receive
AP_ALLOCATION_ERROR AP_COMM_SUBSYSTEM_ABENDED AP_COMM_SUBSYSTEM_NOT_LOADED	Reset
AP_CONV_FAILURE_RETRY AP_CONV_FAILURE_NO_RETRY	Reset
AP_DEALLOC_ABEND AP_DEALLOC_ABEND_PROG AP_DEALLOC_ABEND_SVC AP_DEALLOC_ABEND_TIMER	Reset

Synchronizing with Partner TP

The [MC_]CONFIRM verb waits for a response from the partner TP. A response is generated by one of the following verbs in the partner TP:

- [MC_]CONFIRMED
- [MC_]SEND_ERROR
- MC_DEALLOCATE with *dealloc_type* set to AP_ABEND
- DEALLOCATE with *dealloc_type* set to AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER
- TP_ENDED

MC_CONFIRMED and CONFIRMED

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

Note: This verb can be used only in a half-duplex conversation; it is not valid in a full-duplex conversation.

Because the TP issuing the confirmation request waits for a confirmation, the [MC_]CONFIRMED verb synchronizes the processing of the two TPs.

Sources of Confirmation Requests

A confirmation request is issued by one of the following verbs in the partner TP:

- [MC_]CONFIRM
- [MC_]PREPARE_TO_RECEIVE if *ptr_type* is set to AP_SYNC_LEVEL and the conversation's synchronization level (established by the [MC_]ALLOCATE verb) is AP_CONFIRM_SYNC_LEVEL
- [MC_]DEALLOCATE if *dealloc_type* is set to AP_SYNC_LEVEL and the conversation's synchronization level (established by the [MC_]ALLOCATE verb) is AP_CONFIRM_SYNC_LEVEL
- [MC_]SEND_DATA if *type* is set to AP_SEND_DATA_CONFIRM and the conversation's synchronization level (established by the [MC_]ALLOCATE verb) is AP_CONFIRM_SYNC_LEVEL

Receiving Confirmation Requests

A confirmation request is received by the local TP through the *what_rcvd* parameter of one of the following verbs:

- [MC_]RECEIVE_IMMEDIATE
- [MC_]RECEIVE_AND_WAIT
- [MC_]RECEIVE_AND_POST

The local TP can issue the MC_CONFIRMED or CONFIRMED verb only if the *what_rcvd* field contains one of the following values:

- AP_CONFIRM_WHAT_RECEIVED, AP_DATA_CONFIRM, or AP_DATA_COMPLETE_CONFIRM
- AP_CONFIRM_SEND, AP_DATA_CONFIRM_SEND, or AP_DATA_COMPLETE_CONFIRM_SEND
- AP_CONFIRM_DEALLOCATE, AP_DATA_CONFIRM_DEALLOCATE, or AP_DATA_COMPLETE_CONFIRM_DEALL

VCB Structure: MC_CONFIRMED

UNIX

The definition of the VCB structure for the MC_CONFIRMED verb is as follows:

```
typedef struct mc_confirmed
{
  AP_UINT16         opcode;
  unsigned char     opext;
  unsigned char     format;                 /* Reserved            */
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  AP_UINT32         conv_id;
} MC_CONFIRMED;
```

VCB Structure: CONFIRMED

The definition of the VCB structure for the CONFIRMED verb is as follows:

```
typedef struct confirmed
{
  AP_UINT16         opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  AP_UINT32         conv_id;
} CONFIRMED;
```

VCB Structure: MC_CONFIRMED (Windows)

WINDOWS

The definition of the VCB structure for the MC_CONFIRMED verb is as follows:

```
typedef struct mc_confirmed
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
} MC_CONFIRMED;
```

VCB Structure: CONFIRMED (Windows)

The definition of the VCB structure for the CONFIRMED verb is as follows:

```
typedef struct confirmed
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
} CONFIRMED;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_CONFIRMED
For the MC_CONFIRMED verb.

AP_B_CONFIRMED
For the CONFIRMED verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_CONFIRMED verb.

AP_BASIC_CONVERSATION
For the CONFIRMED verb.

If the verb is being issued as a non-blocking verb, combine the value above (using a logical OR) with the value AP_NON_BLOCKING.

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

UNIX

AP_INVALID_FORMAT
The reserved field *format* was set to a nonzero value.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

AP_CONFIRMED_INVALID_FOR_FDX
The local TP attempted to use the [MC_]CONFIRMED verb in a full-duplex conversation. This verb can be used only in a half-duplex conversation.

State Check: If the conversation is in the wrong state when the TP issues this verb, APPC returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc

AP_CONFIRMED_BAD_STATE
The conversation was not in Confirm, Confirm_Send, or Confirm_Deallocate state.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc
AP_COMM_SUBSYSTEM_ABENDED
AP_CONVERSATION_TYPE_MIXED

```
AP_INVALID_VERB
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR
```

WINDOWS

```
AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT
```

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation must be in one of the following states when the TP issues this verb:

- Confirm
- Confirm_Send
- Confirm_Deallocate

State Change

The new state is determined by the old state—the state of the conversation when the local TP issued the [MC_]CONFIRMED verb. The old state is indicated by the value of the *what_rcvd* parameter of the preceding receive verb. The possible state changes are summarized in the following table.

Old State	New State
Confirm	Receive
Confirm_Send	Send
Confirm_Deallocate	Reset

MC_DEALLOCATE and DEALLOCATE

The MC_DEALLOCATE or DEALLOCATE verb deallocates a conversation between two TPs.

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

- The [MC_]FLUSH verb, sending the contents of the local LU's send buffer to the partner LU (and TP).
- The [MC_]CONFIRM verb, sending the contents of the local LU's send buffer and a confirmation request to the partner TP. ([MC_]CONFIRM applies to half-duplex conversations only.)

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

VCB Structure: MC_DEALLOCATE

UNIX

The definition of the VCB structure for the MC_DEALLOCATE verb is as follows:

```
typedef struct mc_deallocate
{
  AP_UINT16         opcode;
  unsigned char     opext;
  unsigned char     format;
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  AP_UINT32         conv_id;
  unsigned char     expd_rcvd;
  unsigned char     dealloc_type;
  unsigned char     reserv4[2];
  unsigned char     reserv5[4];
  void              (*callback)();
  void *            correlator;
  unsigned char     reserv6[4];
} MC_DEALLOCATE;
```

VCB Structure: DEALLOCATE

The definition of the VCB structure for the DEALLOCATE verb is as follows:

```
typedef struct deallocate
{
  AP_UINT16         opcode;
  unsigned char     opext;
  unsigned char     format;
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  AP_UINT32         conv_id;
  unsigned char     expd_rcvd;
  unsigned char     dealloc_type;
  AP_UINT16         log_dlen;
  unsigned char     *log_dptr;
  void              (*callback)();
  void *            correlator;
  unsigned char     reserv6[4];
} DEALLOCATE;
```

VCB Structure: MC_DEALLOCATE (Windows)

WINDOWS

The definition of the VCB structure for the MC_DEALLOCATE verb is as follows:

```
typedef struct mc_deallocate
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     reserv3;
  unsigned char     dealloc_type;
  unsigned char     reserv4[2];
  unsigned char     reserv5[4];
} MC_DEALLOCATE;
```

VCB Structure: DEALLOCATE (Windows)

The definition of the VCB structure for the DEALLOCATE verb is as follows:

```
typedef struct deallocate
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     reserv3;
  unsigned char     dealloc_type;
  unsigned short    log_dlen;
  unsigned char far *log_dptr;
} DEALLOCATE;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_DEALLOCATE
For the MC_DEALLOCATE verb.

AP_B_DEALLOCATE
For the DEALLOCATE verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_DEALLOCATE verb.

AP_BASIC_CONVERSATION
For the DEALLOCATE verb.

If the verb is being issued on a full-duplex conversation or is being issued as a non-blocking verb, combine the value above (using a logical OR) with one or both of the following values:

AP_FULL_DUPLEX_CONVERSATION
The verb is being issued on a full-duplex conversation.

AP_NON_BLOCKING
The verb is being issued as a non-blocking verb.

tp_id Identifier for the local TP.

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

conv_id

Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

dealloc_type

Specifies how to perform the deallocation. The possible values are listed below.

Using AP_ABEND or any of the AP_ABEND_* values deallocates the conversation abnormally. If the conversation is in Send state when the local TP issues the [MC_]DEALLOCATE verb, APPC sends the contents of the

local LU's send buffer to the partner TP before deallocating the conversation. If the conversation is in Receive or Pending-Post state, APPC purges any incoming data before deallocating the conversation.

AP_ABEND
This value is valid only for the MC_DEALLOCATE verb. A TP should specify AP_ABEND when it encounters an error preventing the successful completion of a transaction.

AP_ABEND_PROG
This value is valid only for the DEALLOCATE verb. An application or service TP should specify AP_ABEND_PROG when it encounters an error preventing the successful completion of a transaction.

AP_ABEND_SVC
A service TP should specify AP_ABEND_SVC when it encounters an error caused by its partner service TP (for example, a format error in control information sent by the partner service TP).

AP_ABEND_TIMER
A service TP should specify AP_ABEND_TIMER when it encounters an error requiring immediate deallocation (for example an operator ending the program prematurely).

AP_FLUSH
Sends the contents of the local LU's send buffer to the partner TP before deallocating the conversation. This value is allowed only if the conversation is in Send or Send_Pending state.

AP_CONFIRM_TYPE
Use this value only if the conversation's synchronization level is AP_SYNCPT. It indicates that confirmation from the partner TP (but not syncpoint processing) is required before deallocating the conversation.

APPC sends the contents of the local LU's send buffer and a confirmation request to the partner TP. Upon receiving confirmation from the partner TP, APPC deallocates the conversation. If, however, the partner TP reports an error, the conversation remains allocated.

AP_SYNC_LEVEL
Uses the conversation's synchronization level (established by the [MC_]ALLOCATE verb) to determine how to deallocate the conversation. This value is allowed only if the conversation is in Send or Send_Pending state.

If the synchronization level of the conversation is AP_NONE, APPC sends the contents of the local LU's send buffer to the partner TP before deallocating the conversation.

If the synchronization level is AP_CONFIRM_SYNC_LEVEL, APPC sends the contents of the local LU's send buffer and a confirmation request to the partner TP. Upon receiving confirmation from the partner TP, APPC deallocates the conversation. If, however, the partner TP reports an error, the conversation remains allocated.

UNIX

If the synchronization level of the conversation is AP_SYNCPT, APPC sends the contents of the local LU's send buffer to the partner TP before deallocating the conversation. The Syncpoint Manager is responsible for the following:

- Intercepting the [MC_]DEALLOCATE verb when a *dealloc_type* of AP_SYNC_LEVEL is specified
- Performing the required syncpoint processing
- Passing the original [MC_]DEALLOCATE verb through to Communications Server when syncpoint processing has completed

When Communications Server receives the [MC_]DEALLOCATE verb with a *dealloc_type* of AP_SYNC_LEVEL on a conversation with *sync_level* of AP_SYNCPT, it assumes that the Syncpoint Manager has already performed all the necessary syncpoint processing, and processes the verb as for a *sync_level* of AP_NONE.

AP_TP_NOT_AVAIL_RETRY
This value should be used only by a TP that issued RECEIVE_ALLOCATE with a blank TP name (to accept incoming conversations for any TP name). It indicates that the TP identified by the TP name specified on the incoming Attach is unavailable because of a temporary condition. The error will be reported to the partner TP with the return codes AP_ALLOCATION_FAILURE and AP_TRANS_PGM_NOT_AVAIL_RETRY; the partner TP can retry the allocation request.

AP_TP_NOT_AVAIL_NO_RETRY
This value should be used only by a TP that issued RECEIVE_ALLOCATE with a blank TP name (to accept incoming conversations for any TP name). It indicates that the TP identified by the TP name specified on the incoming Attach is unavailable because of a condition that requires correction (such as a configuration problem). The error will be reported to the partner TP with the return codes AP_ALLOCATION_FAILURE and AP_TRANS_PGM_NOT_AVAIL_NO_RETRY; the partner TP should not retry the allocation request until the condition that caused this deallocation has been corrected.

AP_TPN_NOT_RECOGNIZED
This value should be used only by a TP that issued RECEIVE_ALLOCATE with a blank TP name (to accept incoming conversations for any TP name). It indicates that the TP name specified on the incoming Attach was not recognized as a valid TP name. The error will be reported to the partner TP with the return codes AP_ALLOCATION_FAILURE and AP_TP_NAME_NOT_RECOGNIZED.

AP_PIP_DATA_NOT_ALLOWED
This value indicates that the local TP is deallocating the conversation because the partner TP supplied PIP data on the [MC_]ALLOCATE verb but the local TP did not expect to receive it. The error will be reported to the partner TP with the return codes AP_ALLOCATION_FAILURE and AP_PIP_NOT_ALLOWED.

AP_PIP_DATA_INCORRECT
This value indicates that the local TP is deallocating the conversation because it expected to receive PIP data from the partner TP, but the partner TP supplied incorrect PIP data or no

PIP data. The error will be reported to the partner TP with the return codes AP_ALLOCATION_FAILURE and AP_PIP_NOT_SPECIFIED_CORRECTLY.

AP_RESOURCE_FAILURE_NO_RETRY
This value indicates that the local TP is deallocating the conversation because a resource required for the TP to operate has failed.

AP_CONV_TYPE_MISMATCH
This value indicates that the local TP is deallocating the conversation because it does not support the conversation type (mapped or basic) specified by the partner TP on [MC_]ALLOCATE. The error will be reported to the partner TP with the return codes AP_ALLOCATION_FAILURE and AP_CONVERSATION_TYPE_MISMATCH.

AP_SYNC_LVL_NOT_SUPPORTED
This value indicates that the local TP is deallocating the conversation because it does not support the synchronization level specified by the partner TP on [MC_]ALLOCATE. The error will be reported to the partner TP with the return codes AP_ALLOCATION_FAILURE and AP_SYNC_LEVEL_NOT_SUPPORTED.

AP_SECURITY_PARAMS_INVALID
This value indicates that the local TP is deallocating the conversation because it did not accept the *security* parameters specified by the partner TP on [MC_]ALLOCATE. The error will be reported to the partner TP with the return codes AP_ALLOCATION_FAILURE and AP_SECURITY_NOT_VALID.

log_dlen
Number of bytes of data to be sent to the error log file. The range for this value is 0–32,767.

This parameter is used only by the DEALLOCATE verb, with the *dealloc_type* parameter set to AP_ABEND_PGM, AP_ABEND_SVC, or AP_ABEND_TIMER. For MC_DEALLOCATE, or for other values of *dealloc_type*, this parameter must be 0 (zero).

log_dptr
Address of data buffer containing error information. This data is sent to the local error log and to the partner LU.

This parameter is used by the DEALLOCATE verb if *log_dlen* is greater than 0 (zero); otherwise it is reserved.

The TP must format the error data as a General Data Stream (GDS) error log variable. For further information, refer to the IBM publication *Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2.*

The log data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

UNIX

The following parameters are used if the conversation's synchronization level is AP_SYNCPT; they are reserved otherwise.

callback
: If the TP requires "implied forget" notification, this parameter specifies a pointer to a callback routine that Communications Server will call to provide this notification. If the TP does not require this notification, it does not use this parameter. For more information, see "Implied Forget Notification" on page 131.

correlator
: An optional correlator for use by the application. This parameter is used only if the *callback* parameter is specified; it is reserved otherwise.

  Communications Server does not use this value, but passes it as a parameter to the callback routine with the "implied forget" notification. This value enables the application to correlate the returned information with its other processing.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
: AP_OK

expd_rcvd
: Expedited data indicator. This parameter is used only in full-duplex conversations, where the TP can continue to receive expedited data after successfully issuing [MC_]DEALLOCATE.

  Possible values are:

  AP_YES The partner TP has sent expedited data that the local TP has not yet received. To receive this data, the local TP can use the [MC_]RECEIVE_EXPEDITED_DATA verb.

  This indicator can be set on a number of APPC verbs. It continues to be set on subsequent verbs until the local TP issues the [MC_]RECEIVE_EXPEDITED_DATA verb to receive the data.

  AP_NO There is no expedited data waiting to be received.

  In a half-duplex conversation, this parameter is always set to AP_NO, because the conversation ends when the [MC_]DEALLOCATE verb completes successfully and so the local TP cannot receive any further expedited data.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

AP_DEALLOC_BAD_TYPE
The *dealloc_type* parameter was not set to a valid value.

AP_DEALLOC_LOG_LL_WRONG
(Returned for basic-conversation DEALLOCATE only) The LL field of the GDS error log variable did not match the actual length of the log data, or the value of the *log_dlen* parameter was incorrect.

UNIX

AP_INVALID_FORMAT
The *format* parameter was set to a value that was not valid.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

WINDOWS

AP_INVALID_DATA_SEGMENT
(Returned for basic-conversation DEALLOCATE only) The log data was longer than the allocated data segment, or the address of the log data buffer was incorrect.

State Check: If the conversation is in the wrong state when the TP issues this verb, APPC returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
Possible values are:

AP_DEALLOC_CONFIRM_BAD_STATE
The conversation was not in Send or Send_Pending state, and the

TP attempted to flush the send buffer and send a confirmation request. This attempt occurred because the value of the *dealloc_type* parameter was AP_SYNC_LEVEL and the synchronization level of the conversation was AP_CONFIRM_SYNC_LEVEL.

AP_DEALLOC_FLUSH_BAD_STATE
The conversation was not in Send or Send_Pending state, and the TP attempted to flush the send buffer. This attempt occurred because the value of the *dealloc_type* parameter was AP_FLUSH or because the value of the *dealloc_type* parameter was AP_SYNC_LEVEL and the synchronization level of the conversation was AP_NONE.

AP_DEALLOC_NOT_LL_BDY
(Returned for basic-conversation DEALLOCATE only) The conversation was in Send state, and the TP did not finish sending a logical record. The *dealloc_type* parameter was set to AP_SYNC_LEVEL or AP_FLUSH.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc

AP_ALLOCATION_ERROR

secondary_rc

AP_ALLOCATION_FAILURE_NO_RETRY
AP_ALLOCATION_FAILURE_RETRY
AP_CONVERSATION_TYPE_MISMATCH
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_SECURITY_NOT_VALID
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_TP_NAME_NOT_RECOGNIZED
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_SEC_BAD_PROTOCOL_VIOLATION
AP_SEC_BAD_PASSWORD_EXPIRED
AP_SEC_BAD_PASSWORD_INVALID
AP_SEC_BAD_USERID_REVOKED
AP_SEC_BAD_USERID_INVALID
AP_SEC_BAD_USERID_MISSING
AP_SEC_BAD_PASSWORD_MISSING
AP_SEC_BAD_UID_NOT_DEFD_TO_GRP
AP_SEC_BAD_UNAUTHRZD_AT_RLU
AP_SEC_BAD_UNAUTHRZD_FROM_LLU
AP_SEC_BAD_UNAUTHRZD_TO_TP
AP_SEC_BAD_INSTALL_EXIT_FAILED
AP_SEC_BAD_PROCESSING_FAILURE

UNIX

primary_rc

AP_BACKED_OUT

secondary_rc
AP_BO_NO_RESYNC
AP_BO_RESYNC

primary_rc
AP_COMM_SUBSYSTEM_ABENDED
AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_CONVERSATION_TYPE_MIXED
AP_DUPLEX_TYPE_MIXED
AP_PROG_ERROR_PURGING
AP_INVALID_VERB
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT

APPC does not return secondary return codes with these primary return codes.

The following primary return code is returned by the MC_DEALLOCATE verb:

primary_rc
AP_DEALLOC_ABEND

APPC does not return a secondary return code with this primary return code.

The following primary return codes are returned by the DEALLOCATE verb:

primary_rc
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_SVC_ERROR_PURGING

APPC does not return secondary return codes with these primary return codes.

State When Issued

Depending on the value of the *dealloc_type* parameter, the conversation can be in one of the states indicated in the following table when the TP issues the [MC_]DEALLOCATE verb.

dealloc_type	**Allowed state**
AP_FLUSH	Send_Receive (full-duplex conversation only), Send or Send_Pending
AP_SYNC_LEVEL	Send_Receive (full-duplex conversation only), Send or Send_Pending

dealloc_type	**Allowed state**
AP_ABEND AP_ABEND_PROG AP_ABEND_SVC AP_ABEND_TIMER	Any except Reset

State Change

State changes, summarized in the following table, are based on the value of the *primary_rc* parameter.

primary_rc	**New state**
AP_OK	Receive_Only (full-duplex conversation with *dealloc_type* set to AP_FLUSH or AP_SYNC_LEVEL), or Reset (all other cases)
AP_PARAMETER_CHECK AP_STATE_CHECK AP_CONVERSATION_TYPE_MIXED AP_INVALID_VERB AP_INVALID_VERB_SEGMENT AP_STACK_TOO_SMALL AP_TP_BUSY AP_UNEXPECTED_DOS_ERROR	No change
AP_ALLOCATION_ERROR AP_CONV_FAILURE_RETRY AP_CONV_FAILURE_NO_RETRY	Reset
AP_DEALLOC_ABEND AP_DEALLOC_ABEND_PROG AP_DEALLOC_ABEND_SVC AP_DEALLOC_ABEND_TIMER	Reset
AP_PROG_ERROR_PURGING AP_SVC_ERROR_PURGING	Receive

Implied Forget Notification

UNIX

The Syncpoint protocols include a feature known as “implied forget”, which means that the FORGET PS Header (the last message in a Syncpoint exchange) is not always required. When the protocol requires a FORGET as the next message to be received on a session, the next data flow received on that session implies that the FORGET has been received.

However, if the message that precedes the FORGET indicates that the conversation is being deallocated, the application no longer has access to the session, and so cannot tell when the next data flow occurs. To provide this information, Communications Server enables the application to specify a callback routine on the [MC_]DEALLOCATE verb; Communications Server then calls this routine when the next data flow occurs on the session, or when the session ends (either normally or abnormally).

If an application uses this feature, it should wait for the callback routine to be called before issuing the TP_ENDED verb for this TP. Communications Server will not call the callback routine after TP_ENDED has been issued.

The callback routine is defined as follows:

```
void (*AP_CALLBACK) (
                       void *           vcb,
                       unsigned char    tp_id[8],
                       AP_UINT32        conv_id,
                       AP_UINT16        type,
                       AP_CORR          corr
                     );
typedef union ap_corr {
                        void *          corr_p;
                        AP_UINT32       corr_l;
                        AP_INT32        corr_i;
                      } AP_CORR;
```

Communications Server calls the routine with the following parameters:

vcb Pointer to the original [MC_]DEALLOCATE VCB supplied by the application. If the application needs to use the VCB parameters in the callback routine, it should not free or reuse the memory associated with the VCB until the callback routine has been called.

tp_id The 8-byte TP identifier of the TP in which the verb was issued.

conv_id
The conversation identifier of the conversation in which the verb was issued. The application cannot issue further verbs using this conversation identifier, because it is no longer valid after the [MC_]DEALLOCATE verb has completed.

type The type of message flow that Communications Server is reporting. Possible values are:

AP_DATA_FLOW
Normal data flow on the session.

AP_UNBIND
The session ended normally.

AP_FAILURE
The session ended abnormally. The Syncpoint manager may need to perform resynchronization.

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

The callback routine need not use all of these parameters. It can perform all the necessary processing on the returned VCB, or it can simply set a variable to inform the main program that the notification has been received.

If the application is using scheduling by signals, the callback routine runs in the context of a signal-catcher. This means that there are limitations on the operating system calls you can use within the routine; refer to your operating system documentation for more information.

The application can issue further APPC verbs from within the callback routine, if required. However, these must be asynchronous verbs. Any synchronous verbs issued from within a callback routine will be rejected with the return codes AP_PARAMETER_CHECK and AP_SYNC_NOT_ALLOWED.

MC_FLUSH and FLUSH

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

Sources of Buffered Data

Data processed by the [MC_]SEND_DATA verb and allocation requests generated by the [MC_]ALLOCATE verb accumulate in the local LU's send buffer until one of the following happens:

- The local TP issues the [MC_]FLUSH verb (or other verb that flushes the LU's send buffer)
- The buffer is full

VCB Structure: MC_FLUSH

UNIX

The definition of the VCB structure for the MC_FLUSH verb is as follows:

```
typedef struct mc_flush
{
  AP_UINT16         opcode;
  unsigned char     opext;
  unsigned char     format;                 /* Reserved            */
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  AP_UINT32         conv_id;
} MC_FLUSH;
```

VCB Structure: FLUSH

The definition of the VCB structure for the FLUSH verb is as follows:

```
typedef struct flush
{
  AP_UINT16         opcode;
  unsigned char     opext;
  unsigned char     format;                 /* Reserved            */
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  AP_UINT32         conv_id;
} FLUSH;
```

VCB Structure: MC_FLUSH (Windows)

WINDOWS

The definition of the VCB structure for the MC_FLUSH verb is as follows:

```
typedef struct mc_flush
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
} MC_FLUSH;
```

VCB Structure: FLUSH (Windows)

The definition of the VCB structure for the FLUSH verb is as follows:

```
typedef struct flush
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
} FLUSH;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_FLUSH
For the MC_FLUSH verb.

AP_B_FLUSH
For the FLUSH verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_FLUSH verb.

AP_BASIC_CONVERSATION
For the FLUSH verb.

If the verb is being issued on a full-duplex conversation or is being issued as a non-blocking verb, combine the value above (using a logical OR) with one or both of the following values:

AP_FULL_DUPLEX_CONVERSATION
The verb is being issued on a full-duplex conversation.

AP_NON_BLOCKING
The verb is being issued as a non-blocking verb.

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

UNIX

AP_INVALID_FORMAT
The reserved field *format* was set to a nonzero value.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

State Check: If the conversation is in the wrong state when the TP issues this verb, APPC returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc

AP_FLUSH_NOT_SEND_STATE
The conversation was not in Send or Send_Pending state.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc
AP_COMM_SUBSYSTEM_ABENDED
AP_CONVERSATION_TYPE_MIXED

AP_DUPLEX_TYPE_MIXED
AP_INVALID_VERB
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation must be in Send_Receive (full-duplex conversation only), Send or Send_Pending state when the TP issues this verb.

State Change

After successful execution, there is no state change.

MC_GET_ATTRIBUTES and GET_ATTRIBUTES

The MC_GET_ATTRIBUTES or GET_ATTRIBUTES verb returns the attributes of the conversation. For more details on these attributes, see Chapter 1, "Concepts," on page 1 of this manual, or the *IBM Communications Server for Data Center Deployment on AIX or Linux Administration Guide*.

VCB Structure: MC_GET_ATTRIBUTES

UNIX

The definition of the VCB structure for the MC_GET_ATTRIBUTES verb is as follows:

```
typedef struct mc_get_attributes
{
  AP_UINT16         opcode;
  unsigned char     opext;
  unsigned char     format;                 /* Reserved             */
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  AP_UINT32         conv_id;
  unsigned char     reserv3;
  unsigned char     sync_level;
  unsigned char     mode_name[8];
  unsigned char     net_name[8];
  unsigned char     lu_name[8];
  unsigned char     lu_alias[8];
  unsigned char     plu_alias[8];
  unsigned char     plu_un_name[8];
  unsigned char     reserv4[2];
  unsigned char     fqplu_name[17];
  unsigned char     reserv5;
  unsigned char     user_id[10];
  AP_UINT32         conv_group_id;
```

```
  unsigned char     conv_corr_len;
  unsigned char     conv_corr[8];
  unsigned char     reserv6[13];
  LUWID_OVERLAY     luw_id;
  unsigned char     sess_id[8];
} MC_GET_ATTRIBUTES;

typedef struct luwid_overlay
{
  unsigned char     fq_length;
  unsigned char     fq_luw_name[17];
  unsigned char     instance[6];
  unsigned char     sequence[2];
} LUWID_OVERLAY;
```

VCB Structure: GET_ATTRIBUTES

The definition of the VCB structure for the GET_ATTRIBUTES verb is as follows:

```
typedef struct get_attributes
{
  AP_UINT16         opcode;
  unsigned char     opext;
  unsigned char     format;                 /* Reserved             */
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  AP_UINT32         conv_id;
  unsigned char     reserv3;
  unsigned char     sync_level;
  unsigned char     mode_name[8];
  unsigned char     net_name[8];
  unsigned char     lu_name[8];
  unsigned char     lu_alias[8];
  unsigned char     plu_alias[8];
  unsigned char     plu_un_name[8];
  unsigned char     reserv4[2];
  unsigned char     fqplu_name[17];
  unsigned char     reserv5;
  unsigned char     user_id[10];
  AP_UINT32         conv_group_id;
  unsigned char     conv_corr_len;
  unsigned char     conv_corr[8];
  unsigned char     reserv6[13];
  LUWID_OVERLAY     luw_id;
  unsigned char     sess_id[8];
} GET_ATTRIBUTES;

typedef struct luwid_overlay
{
  unsigned char     fq_length;
  unsigned char     fq_luw_name[17];
  unsigned char     instance[6];
  unsigned char     sequence[2];
} LUWID_OVERLAY;
```

VCB Structure: MC_GET_ATTRIBUTES (Windows)

WINDOWS

The definition of the VCB structure for the MC_GET_ATTRIBUTES verb is as follows:

```
typedef struct mc_get_attributes
{
  unsigned short    opcode;
  unsigned char     opext;
```

```
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     reserv3;
  unsigned char     sync_level;
  unsigned char     mode_name[8];
  unsigned char     net_name[8];
  unsigned char     lu_name[8];
  unsigned char     lu_alias[8];
  unsigned char     plu_alias[8];
  unsigned char     plu_un_name[8];
  unsigned char     reserv4[2];
  unsigned char     fqplu_name[17];
  unsigned char     reserv5;
  unsigned char     user_id[10];
  unsigned long     conv_group_id;
  unsigned char     conv_corr_len;
  unsigned char     conv_corr[8];
  unsigned char     reserv6[13];
} MC_GET_ATTRIBUTES;
```

VCB Structure: GET_ATTRIBUTES (Windows)

The definition of the VCB structure for the GET_ATTRIBUTES verb is as follows:

```
typedef struct get_attributes
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     reserv3;
  unsigned char     sync_level;
  unsigned char     mode_name[8];
  unsigned char     net_name[8];
  unsigned char     lu_name[8];
  unsigned char     lu_alias[8];
  unsigned char     plu_alias[8];
  unsigned char     plu_un_name[8];
  unsigned char     reserv4[2];
  unsigned char     fqplu_name[17];
  unsigned char     reserv5;
  unsigned char     user_id[10];
  unsigned long     conv_group_id;
  unsigned char     conv_corr_len;
  unsigned char     conv_corr[8];
  unsigned char     reserv6[13];
} GET_ATTRIBUTES;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_GET_ATTRIBUTES
For the MC_GET_ATTRIBUTES verb.

AP_B_GET_ATTRIBUTES
For the GET_ATTRIBUTES verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_GET_ATTRIBUTES verb.

AP_BASIC_CONVERSATION
For the GET_ATTRIBUTES verb.

If the verb is being issued on a full-duplex conversation, combine the value above (using a logical OR) with the value AP_FULL_DUPLEX_CONVERSATION.

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

If the verb executes successfully, APPC returns the following parameters. For more information about the meaning and usage of these parameters, refer to the *IBM Communications Server for Data Center Deployment on AIX or Linux Administration Guide*.

primary_rc
AP_OK

sync_level
Synchronization level of the conversation. This parameter determines whether the TPs can request confirmation of receipt of data and confirm receipt of data.

Possible values are:

AP_CONFIRM_SYNC_LEVEL
The TPs can use confirmation processing in this conversation.

AP_SYNCPT
The TPs can use LU 6.2 Syncpoint functions in this conversation. For more information, see "Syncpoint Support" on page 23.

AP_NONE
Confirmation processing will not be used in this conversation.

mode_name
Name of a set of networking characteristics.

This parameter is an 8-byte EBCDIC character string. It can consist of characters from the type-A EBCDIC character set. These characters are as follows:

- Uppercase letters

- Numerals 0–9
- Special characters $, #, and @

net_name
Name of the network containing the local LU.

This parameter is an 8-byte EBCDIC character string. It can consist of characters from the type-A EBCDIC character set. These characters are as follows:

- Uppercase letters
- Numerals 0–9
- Special characters $, #, and @

lu_name
Name of the local LU.

This parameter is an 8-byte EBCDIC character string. It can consist of characters from the type-A EBCDIC character set. These characters are as follows:

- Uppercase letters
- Numerals 0–9
- Special characters $, #, and @

lu_alias
Alias by which the local LU is known to the local TP. This is an 8-byte ASCII character string.

plu_alias
Alias by which the partner LU is known to the local TP. This is an 8-byte ASCII character string.

plu_un_name
Uninterpreted name of partner LU—the name of the partner LU as defined at the System Services Control Point (SSCP). This is taken from the configuration of the remote LU in the Communications Server configuration file. This parameter is required in the configuration only if the local LU is dependent, so the name returned for an independent LU may be blank or null.

This parameter is an 8-byte EBCDIC character string; it is case-sensitive. It can consist of characters from the type-AE EBCDIC character set. These characters are as follows:

- Uppercase and lowercase letters
- Numerals 0–9
- Special characters $, #, @, and period (.)

fqplu_name
Fully qualified name of the partner LU.

This field contains the network name, an EBCDIC period, and the partner LU name. Each of the two names is an 8-byte EBCDIC character string, which can consist of characters from the type-A EBCDIC character set. These characters are as follows:

- Uppercase letters
- Numerals 0–9
- Special characters $, #, and @

user_id User ID sent by the invoking TP through the [MC_]ALLOCATE verb to access the invoked TP (if applicable).

This parameter is a 10-byte EBCDIC character string; it is case-sensitive. It can consist of characters from the type-AE EBCDIC character set. These characters are as follows:

- Uppercase and lowercase letters
- Numerals 0–9
- Special characters $, #, @, and period (.)

This field contains the user ID if the following conditions are true:

- The invoked TP requires conversation security.
- This verb was issued by the invoked TP.

Otherwise, this field contains blanks.

conv_group_id
The conversation group identifier of the session that this conversation uses.

conv_corr_len
The length (0–8 bytes) of the conversation correlator (see the description of the *conv_corr* parameter for more information).

conv_corr
The conversation correlator assigned by the invoking TP's node when the conversation was allocated.

UNIX

For TPs that use Syncpoint processing, the Syncpoint Manager uses this parameter to identify the conversation during resynchronization processing.

luw_id The Logical Unit of Work Identifier (LUWID) for the transaction in which this conversation is participating. This is assigned on behalf of the TP that initiates the transaction, and enables you to correlate the different conversations that make up the transaction.

A TP that uses Syncpoint processing has two LUWIDs associated with it; the unprotected LUWID is used for conversations with a *sync_level* of `AP_NONE` or `AP_CONFIRM_SYNC_LEVEL`, and the protected LUWID is used for conversations with a *sync_level* of `AP_SYNCPT`. A TP that does not use Syncpoint processing has only one, the unprotected LUWID. This verb returns the LUWID that is associated with this conversation; this is the protected LUWID if the conversation has a *sync_level* of `AP_SYNCPT`, and the unprotected LUWID otherwise. The application can use the GET_TP_PROPERTIES verb to get both LUWIDs for the TP.

The LUWID consists of the following parameters:

luw_id.fq_length
The length (1–17 bytes) of the fully qualified LU name associated with the Logical Unit of Work (the LU name itself is specified by the following parameter).

luw_id.fq_luw_name
The fully qualified LU name associated with the Logical Unit of Work. This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces. It consists of a network ID of 1–8 A-string characters, an EBCDIC dot (period) character, and an LU name of 1–8 A-string characters.

luw_id.instance
The instance number associated with the Logical Unit of Work (a 6-byte binary number).

luw_id.sequence
The sequence number of the current segment of the Logical Unit of Work (a 2-byte binary number).

sess_id The session ID of the session used by this conversation.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

UNIX

AP_INVALID_FORMAT
The *format* parameter was set to a value that was not valid.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

State Check: No state check errors occur for this verb.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc
AP_COMM_SUBSYSTEM_ABENDED
AP_CONVERSATION_TYPE_MIXED
AP_DUPLEX_TYPE_MIXED

```
AP_INVALID_VERB
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR
```

WINDOWS

```
AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT
```

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation can be in any state except Reset when the TP issues this verb.

State Change

The conversation state does not change for this verb.

MC_PREPARE_TO_RECEIVE and PREPARE_TO_RECEIVE

The MC_PREPARE_TO_RECEIVE or PREPARE_TO_RECEIVE verb changes the state of the conversation for the local TP from Send or Send_Pending to Receive.

Note: This verb can be used only in a half-duplex conversation; it is not valid in a full-duplex conversation.

Before changing the conversation state, this verb performs the equivalent of one of the following, depending on the *ptr_type* (prepare-to-receive type) parameter as described below:

- The [MC_]FLUSH verb, sending the contents of the local LU's send buffer to the partner LU (and TP)
- The [MC_]CONFIRM verb, sending the contents of the local LU's send buffer and a confirmation request to the partner TP

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

VCB Structure: MC_PREPARE_TO_RECEIVE

UNIX

The definition of the VCB structure for the MC_PREPARE_TO_RECEIVE verb is as follows:

```
typedef struct mc_prepare_to_receive
{
  AP_UINT16       opcode;
  unsigned char   opext;
  unsigned char   format;                 /* Reserved            */
  AP_UINT16       primary_rc;
  AP_UINT32       secondary_rc;
  unsigned char   tp_id[8];
```

```
  AP_UINT32       conv_id;
  unsigned char   ptr_type;
  unsigned char   locks;
} MC_PREPARE_TO_RECEIVE;
```

VCB Structure: PREPARE_TO_RECEIVE

The definition of the VCB structure for the PREPARE_TO_RECEIVE verb is as follows:

```
typedef struct prepare_to_receive
{
  AP_UINT16       opcode;
  unsigned char   opext;
  unsigned char   format;                  /* Reserved             */
  AP_UINT16       primary_rc;
  AP_UINT32       secondary_rc;
  unsigned char   tp_id[8];
  AP_UINT32       conv_id;
  unsigned char   ptr_type;
  unsigned char   locks;
} PREPARE_TO_RECEIVE;
```

VCB Structure: MC_PREPARE_TO_RECEIVE (Windows)

WINDOWS

The definition of the VCB structure for the MC_PREPARE_TO_RECEIVE verb is as follows:

```
typedef struct mc_prepare_to_receive
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     ptr_type;
  unsigned char     locks;
} MC_PREPARE_TO_RECEIVE;
```

VCB Structure: PREPARE_TO_RECEIVE (Windows)

The definition of the VCB structure for the PREPARE_TO_RECEIVE verb is as follows:

```
typedef struct prepare_to_receive
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     ptr_type;
  unsigned char     locks;
} PREPARE_TO_RECEIVE;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_PREPARE_TO_RECEIVE
For the MC_PREPARE_TO_RECEIVE verb.

AP_B_PREPARE_TO_RECEIVE
For the PREPARE_TO_RECEIVE verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_PREPARE_TO_RECEIVE verb.

AP_BASIC_CONVERSATION
For the PREPARE_TO_RECEIVE verb.

If the verb is being issued as a non-blocking verb, combine the value above (using a logical OR) with the value AP_NON_BLOCKING.

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

ptr_type
Specifies how to perform the state change.

Possible values are:

AP_FLUSH
Sends the contents of the local LU's send buffer to the partner LU (and TP) before changing the conversation's state to Receive.

UNIX

AP_CONFIRM_TYPE
Use this value only if the conversation's synchronization level is AP_SYNCPT. It indicates that confirmation from the partner TP (but not syncpoint processing) is required before changing the conversation's state to Receive.

APPC sends the contents of the local LU's send buffer and a confirmation request to the partner TP. The conversation state does not change to Receive until the partner TP sends the requested confirmation (or reports an error).

AP_SYNC_LEVEL
Uses the conversation's synchronization level (established by the [MC_]ALLOCATE verb) to determine how to perform the state change.

If the synchronization level of the conversation is AP_NONE, APPC sends the contents of the local LU's send buffer to the partner TP before changing the conversation's state to Receive.

If the synchronization level is AP_CONFIRM_SYNC_LEVEL, APPC sends the contents of the local LU's send buffer and a confirmation request to the partner TP. Upon receiving confirmation from the partner TP, APPC changes the conversation's state to Receive. If, however, the partner TP reports an error, the state changes to Receive or Reset; see "State Change" on page 149.

UNIX

If the synchronization level of the conversation is AP_SYNCPT, APPC sends the contents of the local LU's send buffer to the partner TP before changing the conversation state. The Syncpoint Manager is responsible for the following:

- Intercepting the [MC_]PREPARE_TO_RECEIVE verb when a *ptr_type* of AP_SYNC_LEVEL is specified
- Performing the required syncpoint processing
- Passing the original [MC_]PREPARE_TO_RECEIVE verb through to Communications Server when syncpoint processing has completed

When Communications Server receives the [MC_]PREPARE_TO_RECEIVE verb with a *ptr_type* of AP_SYNC_LEVEL on a conversation with *sync_level* of AP_SYNCPT, it assumes that the Syncpoint Manager has already performed all the necessary syncpoint processing, and processes the verb as for a *sync_level* of AP_NONE.

locks Specifies when APPC is to return control to the local TP.

Use this parameter only if *ptr_type* is set to AP_SYNC_LEVEL and the synchronization level of the conversation, established by the [MC_]ALLOCATE verb is AP_CONFIRM_SYNC_LEVEL. (Otherwise, the parameter is ignored.)

Possible values are:

AP_LONG
APPC returns control to the local TP when the confirmation and subsequent data from the partner TP arrive at the local LU. (This method results in more efficient use of the network but requires longer to return control to the local TP.)

AP_SHORT
APPC returns control to the local TP when the confirmation from the partner TP arrives at the local LU.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

AP_P_TO_R_INVALID_FOR_FDX
The local TP attempted to use the [MC_]PREPARE_TO_RECEIVE verb in a full-duplex conversation. This verb can be used only in a half-duplex conversation.

AP_P_TO_R_INVALID_TYPE
The *ptr_type* parameter was not set to a valid value.

UNIX

AP_INVALID_FORMAT
The reserved field *format* was set to a nonzero value.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

State Check: If the conversation is in the wrong state when the TP issues this verb, APPC returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
Possible values are:

AP_P_TO_R_NOT_LL_BDY
(Returned for basic-conversation PREPARE_TO_RECEIVE only) The local TP did not finish sending a logical record.

AP_P_TO_R_NOT_SEND_STATE
The conversation was not in Send or Send_Pending state.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc
AP_ALLOCATION_ERROR

secondary_rc
AP_ALLOCATION_FAILURE_NO_RETRY
AP_ALLOCATION_FAILURE_RETRY
AP_CONVERSATION_TYPE_MISMATCH
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_SECURITY_NOT_VALID
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_TP_NAME_NOT_RECOGNIZED
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_SEC_BAD_PROTOCOL_VIOLATION
AP_SEC_BAD_PASSWORD_EXPIRED
AP_SEC_BAD_PASSWORD_INVALID
AP_SEC_BAD_USERID_REVOKED
AP_SEC_BAD_USERID_INVALID
AP_SEC_BAD_USERID_MISSING
AP_SEC_BAD_PASSWORD_MISSING
AP_SEC_BAD_UID_NOT_DEFD_TO_GRP
AP_SEC_BAD_UNAUTHRZD_AT_RLU
AP_SEC_BAD_UNAUTHRZD_FROM_LLU
AP_SEC_BAD_UNAUTHRZD_TO_TP
AP_SEC_BAD_INSTALL_EXIT_FAILED
AP_SEC_BAD_PROCESSING_FAILURE

UNIX

primary_rc
AP_BACKED_OUT

secondary_rc
AP_BO_NO_RESYNC
AP_BO_RESYNC

primary_rc
AP_COMM_SUBSYSTEM_ABENDED
AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_CONVERSATION_TYPE_MIXED
AP_PROG_ERROR_PURGING
AP_INVALID_VERB

AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT

APPC does not return secondary return codes with these primary return codes.

The following primary return code is returned by the MC_PREPARE_TO_RECEIVE verb:

primary_rc
AP_DEALLOC_ABEND

The following primary return codes are returned by the PREPARE_TO_RECEIVE verb:

primary_rc
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_SVC_ERROR_PURGING

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation must be in Send or Send_Pending state when the TP issues this verb.

State Change

State changes, summarized in the following table, are based on the value of the *primary_rc* parameter.

primary_rc	**New state**
AP_OK	Receive
AP_PARAMETER_CHECK	No change
AP_STATE_CHECK	
AP_CONVERSATION_TYPE_MIXED	
AP_INVALID_VERB	
AP_INVALID_VERB_SEGMENT	
AP_STACK_TOO_SMALL	
AP_TP_BUSY	
AP_UNEXPECTED_DOS_ERROR	
AP_ALLOCATION_ERROR	Reset
AP_CONV_FAILURE_RETRY	Reset
AP_CONV_FAILURE_NO_RETRY	
AP_DEALLOC_ABEND_RESET	Reset
AP_DEALLOC_ABEND_PROG	
AP_DEALLOC_ABEND_SVC	
AP_DEALLOC_ABEND_TIMER	

primary_rc	**New state**
AP_PROG_ERROR_PURGING_RECEIVE AP_SVC_ERROR_PURGING	Receive

Usage Note

The conversation does not change to Send or Send_Pending state for the partner TP until the partner TP receives one of the following values through the *what_rcvd* parameter of a subsequent receive verb:

- AP_SEND, AP_DATA_SEND, AP_DATA_COMPLETE_SEND
- AP_CONFIRM_SEND, AP_DATA_CONFIRM_SEND, or AP_DATA_COMPLETE_CONFIRM_SEND (and replies with [MC_]CONFIRMED)

The RECEIVE verbs are [MC_]RECEIVE_AND_WAIT, [MC_]RECEIVE_IMMEDIATE, and [MC_]RECEIVE_AND_POST.

MC_RECEIVE and RECEIVE Verbs

APPC provides three different verbs which are used to receive data from the partner TP. Most of the parameters and return codes are the same for all three verbs, but each operates in a different way and provides a different function. Common information which applies to all three verbs are explained together in this section; each verb is then explained in detail.

The three RECEIVE verbs are:

- [MC_]RECEIVE_IMMEDIATE
- [MC_]RECEIVE_AND_WAIT
- [MC_]RECEIVE_AND_POST

Note: The [MC_]RECEIVE_EXPEDITED_DATA verb also receives data from the partner TP, but it receives data that was sent as expedited flow data rather than normal flow data. This verb is described separately after the other RECEIVE verbs.

How a TP Receives Data

The process through which the local TP receives data is as follows:

1. The local TP issues a receive verb until it finishes receiving a complete unit of data. The data received can be any of the following:
   - One data record transmitted in a mapped conversation
   - One logical record transmitted in a basic conversation
   - A buffer of data received independent of its logical-record format in a basic conversation

   The local TP may need to issue several RECEIVE verbs in order to receive a complete unit of data. Once a complete unit of data has been received, the local TP can manipulate it.
2. The local TP issues another receive verb. This has one of the following effects:
   - If the partner TP has sent more data, the local TP begins to receive a new unit of data.

- If the partner TP has finished sending data or is waiting for confirmation, status information (available through the *what_rcvd* parameter) indicates the next action the local TP normally takes. For more information, see "The what_rcvd Parameter."

Alternatively, the local TP can set a parameter *rtn_status* when issuing the receive verb; this indicates that any status information available is to be returned with the data. In this case, the receive verb that returns the last part of the data also returns the status information, and the local TP does not need to issue a separate receive verb for it. For more information, see "The what_rcvd Parameter."

The what_rcvd Parameter

After issuing one of the [MC_]RECEIVE verbs, a TP will normally use the *what_rcvd* parameter to determine its next action. The values referring to a data type of "User Control" will be returned on a mapped conversation on the AIX or Linux system, and the values referring to a data type of "PS Header" will be returned on a mapped conversation on the AIX or Linux system with a synchronization level of `AP_SYNCPT`.

The following list describes the possible values of the *what_rcvd* parameter, with the action normally taken by the TP for each of them:

AP_DATA AP_DATA_COMPLETE AP_DATA_INCOMPLETE AP_USER_CONTROL_DATA_COMPLETE AP_USER_CONTROL_DATA_INCMP AP_PS_HEADER_COMPLETE AP_PS_HEADER_INCOMPLETE
: The local TP received data from the partner TP. It will normally continue to issue RECEIVE verbs until it receives one of the other *what_rcvd* parameters in this list.

AP_SEND (half-duplex conversation only)
: The partner TP issued the [MC_]PREPARE_TO_RECEIVE verb without requesting confirmation, or issued the [MC_]SEND_DATA verb with a send type of PREPARE_TO_RECEIVE. The local TP is now in Send state, so it will normally begin to send data.

AP_CONFIRM_DEALLOCATE (half-duplex conversation only)
: The partner TP issued the [MC_]DEALLOCATE verb with a *dealloc_type* parameter indicating that confirmation was required, or issued the [MC_]SEND_DATA verb with a send type of DEALLOCATE. The local TP is now in Confirm_Deallocate state, so it will normally issue the [MC_]CONFIRMED verb to confirm deallocation of the conversation.

AP_CONFIRM_SEND (half-duplex conversation only)
: The partner TP issued the [MC_]PREPARE_TO_RECEIVE verb with *ptr_type* and *dealloc_type* parameters indicating that confirmation was required, or issued the [MC_]SEND_DATA verb with a send type of PREPARE_TO_RECEIVE_CONFIRM. The local TP is now in Confirm_Send state, so it will normally issue the [MC_]CONFIRMED verb to confirm the state change and then begin to send data.

AP_CONFIRM_WHAT_RECEIVED (half-duplex conversation only)
: The partner TP issued the [MC_]CONFIRM verb, or issued the [MC_]SEND_DATA verb with a send type of CONFIRM. The local TP is now in Confirm state, so it will normally issue the [MC_]CONFIRMED verb.

The following values will only be returned if the local TP specified AP_YES for the *rtn_status* (return status with data) parameter:

AP_DATA_SEND AP_DATA_COMPLETE_SEND AP_UC_DATA_COMPLETE_SEND AP_PS_HDR_COMPLETE_SEND
: The partner TP sent data and then issued the [MC_]PREPARE_TO_RECEIVE verb without requesting confirmation, or issued the [MC_]SEND_DATA verb with a send type of PREPARE_TO_RECEIVE. The local TP is now in Send_Pending state, so it will normally begin to send data.

AP_DATA_CONFIRM_DEALLOCATE AP_DATA_COMPLETE_CONFIRM_DEALL AP_UC_DATA_COMPLETE_CNFM_DEALL AP_PS_HDR_COMLETE_CNFM_DEALL
: All of these values apply only to half-duplex conversations.

  The partner TP sent data and then issued the [MC_]DEALLOCATE verb with a *dealloc_type* parameter indicating that confirmation was required, or issued the [MC_]SEND_DATA verb with a send type of DEALLOCATE. The local TP is now in Confirm_Deallocate state, so it will normally issue the [MC_]CONFIRMED verb to confirm deallocation of the conversation.

AP_DATA_CONFIRM_SEND, AP_DATA_COMPLETE_CONFIRM_SEND, AP_UC_DATA_COMPLETE_CNFM_SEND, AP_PS_HDR_COMPLETE_CNFM_SEND
: All of these values apply only to half-duplex conversations.

  The partner TP sent data and then issued the [MC_]PREPARE_TO_RECEIVE verb with *ptr_type* and *dealloc_type* parameters indicating that confirmation was required, or issued the [MC_]SEND_DATA verb with a send type of PREPARE_TO_RECEIVE_CONFIRM. The local TP is now in Confirm_Send state, so it will normally issue the [MC_]CONFIRMED verb to confirm the state change and then begin to send data.

AP_DATA_CONFIRM, AP_DATA_COMPLETE_CONFIRM, AP_UC_DATA_COMPLETE_CONFIRM, AP_PS_HDR_COMPLETE_CONFIRM
: All of these values apply only to half-duplex conversations.

  The partner TP sent data and then issued the [MC_]CONFIRM verb, or issued the [MC_]SEND_DATA verb with a send type of CONFIRM. The local TP is now in Confirm state, so it will normally issue the [MC_]CONFIRMED verb.

  In all CONFIRM cases above, the TP may issue the [MC_]SEND_ERROR verb instead of the [MC_]CONFIRMED verb, to indicate that an error was detected in the supplied data or in processing. If it issues [MC_]SEND_ERROR in Send_Pending state (after receiving AP_DATA_SEND, AP_DATA_COMPLETE_SEND, AP_UC_DATA_COMPLETE_SEND, or AP_PS_HDR_COMPLETE_SEND), it can specify whether the error was detected in the supplied data, or in its own data or processing. For more information, see the description of the [MC_]SEND_ERROR verb in "MC_SEND_ERROR and SEND_ERROR" on page 220.

End of Data

If the local TP issues one of the basic-conversation RECEIVE verbs and sets the *fill* parameter to AP_BUFFER, the receipt of data ends when *max_len* or end of data is reached. End of data is indicated by either of the following:

- A *primary_rc* parameter with a value other than AP_OK (for example, AP_DEALLOC_NORMAL)
- A *what_rcvd* parameter that includes SEND, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE

To determine if end of data has been reached, the local TP reissues one of the RECEIVE verbs. If the new *primary_rc* parameter contains AP_OK and *what_rcvd* contains AP_DATA or AP_DATA_INCOMPLETE, end of data has not been reached. If, however, end of data has been reached, the *primary_rc* or *what_rcvd* parameter will indicate the cause of the end of data.

Testing the what_rcvd Parameter

The local TP can use any of the [MC_]RECEIVE verbs to determine whether the partner TP has data to send, seeks confirmation, or has changed the conversation state, without receiving any data. To do this, it issues the [MC_]RECEIVE verb with the *max_len* parameter set to 0 (zero), and then (if the verb returns with a *primary_rc* of AP_OK) tests the *what_rcvd* parameter.

MC_RECEIVE_AND_POST and RECEIVE_AND_POST

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

Note: This verb can be used only in a half-duplex conversation; it is not valid in a full-duplex conversation.

VCB Structure: MC_RECEIVE_AND_POST

UNIX

The definition of the VCB structure for the MC_RECEIVE_AND_POST verb is as follows:

```
typedef struct mc_receive_and_post
{
  AP_UINT16       opcode;
  unsigned char   opext;
  unsigned char   format;                  /* Reserved            */
  AP_UINT16       primary_rc;
  AP_UINT32       secondary_rc;
  unsigned char   tp_id[8];
  AP_UINT32       conv_id;
  AP_UINT16       what_rcvd;
  unsigned char   rtn_status;
  unsigned char   reserv4;
  unsigned char   rts_rcvd;
  unsigned char   expd_rcvd;
  AP_UINT16       max_len;
  AP_UINT16       dlen;
  unsigned char   *dptr;
  void            (*callback)();
  unsigned char   reserv6;
} MC_RECEIVE_AND_POST;
```

VCB Structure: RECEIVE_AND_POST

The definition of the VCB structure for the RECEIVE_AND_POST verb is as follows:

```
typedef struct receive_and_post
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                  /* Reserved             */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  AP_UINT16        what_rcvd;
  unsigned char    rtn_status;
  unsigned char    fill;
  unsigned char    rts_rcvd;
  unsigned char    expd_rcvd;
  AP_UINT16        max_len;
  AP_UINT16        dlen;
  unsigned char    *dptr;
  void             (*callback)();
  unsigned char    reserv5;
} RECEIVE_AND_POST;
```

VCB Structure: MC_RECEIVE_AND_POST (Windows)

WINDOWS

The definition of the VCB structure for the MC_RECEIVE_AND_POST verb is as follows:

```
typedef struct mc_receive_and_post
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned short    what_rcvd;
  unsigned char     rtn_status;
  unsigned char     reserv4;
  unsigned char     rts_rcvd;
  unsigned char     reserv5;
  unsigned short    max_len;
  unsigned short    dlen;
  unsigned char far *dptr;
  unsigned char far *sema;
  unsigned char     reserv6;
} MC_RECEIVE_AND_POST;
```

VCB Structure: RECEIVE_AND_POST (Windows)

The definition of the VCB structure for the RECEIVE_AND_POST verb is as follows:

```
typedef struct receive_and_post
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned short    what_rcvd;
  unsigned char     rtn_status;
```

```
  unsigned char     fill;
  unsigned char     rts_rcvd;
  unsigned char     reserv4;
  unsigned short    max_len;
  unsigned short    dlen;
  unsigned char far *dptr;
  unsigned char far *sema;
  unsigned char     reserv5;
} RECEIVE_AND_POST;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_RECEIVE_AND_POST
For the MC_RECEIVE_AND_POST verb.

AP_B_RECEIVE_AND_POST
For the RECEIVE_AND_POST verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_RECEIVE_AND_POST verb.

AP_BASIC_CONVERSATION
For the RECEIVE_AND_POST verb.

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

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

Possible values are:

AP_YES Status information, if available, is returned with the last part of a data record.

AP_NO Status information is not returned with data. After receiving the end of a data record, the local TP must issue another [MC_]RECEIVE verb to obtain the status information.

fill Indicates the manner in which the local TP receives data.

This parameter is used only by the basic-conversation RECEIVE_AND_POST verb.

Possible values are:

AP_BUFFER
The local TP receives data until the number of bytes specified by

the *max_len* parameter is reached or until end of data. Data is received without regard for the logical-record format.

AP_LL Data is received in logical-record format. The data received can be any of the following:

- A complete logical record
- A *max_len*-byte portion of a logical record
- The end of a logical record

max_len
Maximum number of bytes of data the local TP can receive.

The range for this value is `0–65,535`.

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.

UNIX

callback
Address of the callback routine which APPC is to call when the asynchronous receiving operation is finished. For more information, see "Usage Notes®" on page 164.

WINDOWS

sema A Windows event handle, obtained by calling one of the two Windows functions `CreateEvent` or `OpenEvent`. APPC signals this event handle to inform the TP when the asynchronous receiving operation is finished.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

When this verb is issued, it returns immediately with a *primary_rc* which indicates whether or not the verb was issued successfully. The only returned parameters which are valid at this stage are *primary_rc*, *secondary_rc* (if the *primary_rc* is not `AP_OK`), and *rts_rcvd*. The possible *primary_rc* and *secondary_rc* values are as described later in this section.

If this *primary_rc* is `AP_OK`, the verb has successfully begun to receive data asynchronously. When the verb has completed (either because it has successfully received data or because it was terminated by a conversation error), APPC calls the supplied callback routine. At this point, the returned parameters are as shown below. The *primary_rc* and *secondary_rc* parameters will now have new values indicating whether or not data was received successfully, and should be examined again.

Successful Execution

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

primary_rc
AP_OK

what_rcvd
Status information received with the incoming data.

The next action taken by the TP will usually depend on the value of this parameter. For more information, see "The what_rcvd Parameter" on page 151.

Possible values are:

AP_CONFIRM_DEALLOCATE
The partner TP has issued the [MC_]DEALLOCATE verb with *dealloc_type* set to AP_SYNC_LEVEL, and the conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_CONFIRM_SEND
The partner TP has issued the [MC_]PREPARE_TO_RECEIVE verb with *ptr_type* set to AP_SYNC_LEVEL, and the conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_CONFIRM_WHAT_RECEIVED
The partner TP has issued the [MC_]CONFIRM verb.

AP_DATA
This value can be returned by the basic-conversation RECEIVE_AND_POST if the *fill* parameter is set to AP_BUFFER; it is not applicable to MC_RECEIVE_AND_POST.

The local TP received data until *max_len* or end of data was reached.

AP_DATA_COMPLETE
For MC_RECEIVE_AND_POST, this value indicates that the local TP has received a complete data record or the last part of a data record. For RECEIVE_AND_POST with the *fill* parameter set to AP_LL, this value indicates that the local TP has received a complete logical record or the end of a logical record.

AP_DATA_INCOMPLETE
For MC_RECEIVE_AND_POST, this value indicates that the local TP has received an incomplete data record. The *max_len* parameter specified a value less than the length of the data record (or less than the remainder of the data record if this is not the first receive verb to read the record).

For RECEIVE_AND_POST with the *fill* parameter set to AP_LL, this value indicates that the local TP has received an incomplete logical record.

AP_SEND
For the partner TP, the conversation has entered Receive state. For the local TP, the conversation is now in Send state.

The following values will only be returned if *rtn_status* was set to AP_YES:

AP_DATA_CONFIRM
This is a combination of AP_DATA and AP_CONFIRM_WHAT_RECEIVED. The partner TP sent data and then issued the [MC_]CONFIRM verb, or issued the [MC_]SEND_DATA verb with a send type of CONFIRM.

AP_DATA_COMPLETE_CONFIRM
This is a combination of AP_DATA_COMPLETE and AP_CONFIRM_WHAT_RECEIVED. The partner TP sent a complete data record (or the end of a data record) and then issued the [MC_]CONFIRM verb, or issued the [MC_]SEND_DATA verb with a send type of CONFIRM.

AP_DATA_CONFIRM_DEALLOCATE
This is a combination of AP_DATA and AP_CONFIRM_DEALLOCATE. The partner TP sent data and then issued the [MC_]DEALLOCATE verb with *dealloc_type* set to AP_SYNC_LEVEL, or issued the [MC_]SEND_DATA verb with a send type of DEALLOC_SYNC_LEVEL. The conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_DATA_COMPLETE_CONFIRM_DEALL
This is a combination of AP_DATA_COMPLETE and AP_CONFIRM_DEALLOCATE. The partner TP sent a complete data record (or the end of a data record) and then issued the [MC_]DEALLOCATE verb with *dealloc_type* set to AP_SYNC_LEVEL, or issued the [MC_]SEND_DATA verb with a send type of DEALLOC_SYNC_LEVEL. The conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_DATA_CONFIRM_SEND
This is a combination of AP_DATA and AP_CONFIRM_SEND. The partner TP sent data and then issued the [MC_]PREPARE_TO_RECEIVE verb with *ptr_type* set to AP_SYNC_LEVEL, or issued the [MC_]SEND_DATA verb with a send type of P_TO_R_SYNC_LEVEL. The conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_DATA_COMPLETE_CONFIRM_SEND
This is a combination of AP_DATA_COMPLETE and AP_CONFIRM_SEND. The partner TP sent a complete data record (or the end of a data record) and then issued the [MC_]PREPARE_TO_RECEIVE verb with *ptr_type* set to AP_SYNC_LEVEL, or issued the [MC_]SEND_DATA verb with a send type of P_TO_R_SYNC_LEVEL. The conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_DATA_SEND
The partner TP sent data and then entered Receive state. For the local TP, the conversation is now in Send_Pending state.

AP_DATA_COMPLETE_SEND
The partner TP sent a complete data record (or the end of a data record) and then entered Receive state. For the local TP, the conversation is now in Send_Pending state.

The following values will be returned on the MC_RECEIVE_AND_POST verb:

AP_USER_CONTROL_DATA_INCMP
As for AP_DATA_INCOMPLETE, except that the received data was in User Control Data format.

AP_USER_CONTROL_DATA_COMPLETE
As for AP_DATA_COMPLETE, except that the received data was in User Control Data format.

AP_UC_DATA_COMPLETE_SEND
As for AP_DATA_COMPLETE_SEND, except that the received data was in User Control Data format.

AP_UC_DATA_COMPLETE_CONFIRM
As for AP_DATA_COMPLETE_CONFIRM, except that the received data was in User Control Data format.

AP_UC_DATA_COMPLETE_CNFM_DEALL
As for AP_DATA_COMPLETE_CONFIRM_DEALL, except that the received data was in User Control Data format.

AP_UC_DATA_COMPLETE_CNFM_SEND
As for AP_DATA_COMPLETE_CONFIRM_SEND, except that the received data was in User Control Data format.

The following values will be returned on the MC_RECEIVE_AND_POST verb with *sync_level* set to AP_SYNCPT:

AP_PS_HEADER_INCOMPLETE
As for AP_DATA_INCOMPLETE, except that the received data was in PS Header format.

AP_PS_HEADER_COMPLETE
As for AP_DATA_COMPLETE, except that the received data was in PS Header format.

AP_PS_HDR_COMPLETE_SEND
As for AP_DATA_COMPLETE_SEND, except that the received data was in PS Header format.

AP_PS_HDR_COMPLETE_CONFIRM
As for AP_DATA_COMPLETE_CONFIRM, except that the received data was in PS Header format.

AP_PS_HDR_COMPLETE_CNFM_DEALL
As for AP_DATA_COMPLETE_CONFIRM_DEALL, except that the received data was in PS Header format.

AP_PS_HDR_COMPLETE_CNFM_SEND
As for AP_DATA_COMPLETE_CONFIRM_SEND, except that the received data was in PS Header format.

rts_rcvd
Request-to-send-received indicator. This parameter applies only in a half-duplex conversation; it is not used in a full-duplex conversation.

Possible values are:

AP_YES The partner TP has issued the [MC_]REQUEST_TO_SEND verb, which requests that the local TP change the conversation to Receive state.

AP_NO The partner TP has not issued the [MC_]REQUEST_TO_SEND verb.

For an explanation of why this indicator can be received by receive verbs, see "MC_REQUEST_TO_SEND and REQUEST_TO_SEND" on page 196.

expd_rcvd
Expedited data indicator.

Possible values are:

AP_YES The partner TP has sent expedited data that the local TP has not yet received. To receive this data, the local TP can use the [MC_]RECEIVE_EXPEDITED_DATA verb.

This indicator can be set on a number of APPC verbs. It continues to be set on subsequent verbs until the local TP issues the [MC_]RECEIVE_EXPEDITED_DATA verb to receive the data.

AP_NO There is no expedited data waiting to be received.

dlen Number of bytes of data received (the data is stored in the buffer specified by the *dptr* parameter). A length of `0` (zero) indicates that no data was received. This parameter is only used if the *what_rcvd* parameter indicates that data was received.

Conversation Deallocated: If the partner TP has deallocated the conversation without requesting confirmation, APPC returns the following parameters:

primary_rc

AP_DEALLOC_NORMAL
The partner TP issued the [MC_]DEALLOCATE verb with *dealloc_type* set to one of the following:

- `AP_FLUSH`
- `AP_SYNC_LEVEL` with the synchronization level of the conversation specified as `AP_NONE`

dlen Number of bytes of data received (the data is stored in the buffer specified by the *dptr* parameter). A length of `0` (zero) indicates that no data was received. This parameter is only used if *rtn_status* was set to `AP_YES`.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
`AP_PARAMETER_CHECK`

secondary_rc
Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_RETURN_STATUS_WITH_DATA
The *rtn_status* parameter was set to a value that was not valid.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

AP_INVALID_FORMAT
The reserved field *format* was set to a nonzero value.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

AP_INVALID_CALLBACK_HANDLE
The *callback* parameter was set to a null pointer, and the verb was issued using the synchronous entry point (or using the asynchronous entry point with a null pointer to a callback routine). For more information, see "Usage Notes®" on page 164.

AP_RCV_AND_POST_BAD_FILL
This return code applies only to the basic-conversation RECEIVE_AND_POST verb. The *fill* parameter was set to a value that was not valid.

State Check: If the conversation is in the wrong state when the TP issues this verb, APPC returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
Possible values are:

AP_RCV_AND_POST_BAD_STATE
The conversation was not in Receive, Send, or Send_Pending state when the TP issued this verb.

AP_RCV_AND_POST_NOT_LL_BDY
This return code applies only to the basic-conversation RECEIVE_AND_POST verb. The conversation was in Send state; the TP began but did not finish sending a logical record.

Verb Canceled: This return code cannot be returned as the initial return code, but only as the subsequent return code if the initial return code is AP_OK.

If the verb did not execute because it was canceled by another verb issued by the TP, APPC returns the following parameter:

primary_rc

AP_CANCELLED
The local TP issued one of the following verbs while in Pending_Post state:

- DEALLOCATE with *dealloc_type* set to AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER
- MC_DEALLOCATE with *dealloc_type* set to AP_ABEND
- [MC_]SEND_ERROR
- TP_ENDED

Issuing one of these verbs while in Pending_Post state causes the [MC_]RECEIVE_AND_POST verb to be canceled. The callback routine is not called. The local TP is no longer receiving data asynchronously from the partner TP.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc
AP_ALLOCATION_ERROR

secondary_rc
AP_ALLOCATION_FAILURE_NO_RETRY
AP_ALLOCATION_FAILURE_RETRY
AP_CONVERSATION_TYPE_MISMATCH
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_SECURITY_NOT_VALID
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_TP_NAME_NOT_RECOGNIZED
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_SEC_BAD_PROTOCOL_VIOLATION
AP_SEC_BAD_PASSWORD_EXPIRED
AP_SEC_BAD_PASSWORD_INVALID
AP_SEC_BAD_USERID_REVOKED
AP_SEC_BAD_USERID_INVALID
AP_SEC_BAD_USERID_MISSING
AP_SEC_BAD_PASSWORD_MISSING
AP_SEC_BAD_UID_NOT_DEFD_TO_GRP
AP_SEC_BAD_UNAUTHRZD_AT_RLU
AP_SEC_BAD_UNAUTHRZD_FROM_LLU
AP_SEC_BAD_UNAUTHRZD_TO_TP
AP_SEC_BAD_INSTALL_EXIT_FAILED
AP_SEC_BAD_PROCESSING_FAILURE

primary_rc
AP_BACKED_OUT

secondary_rc
AP_BO_NO_RESYNC
AP_BO_RESYNC

primary_rc
AP_COMM_SUBSYSTEM_ABENDED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_CONVERSATION_TYPE_MIXED
AP_PROG_ERROR_NO_TRUNC
AP_PROG_ERROR_PURGING
AP_PROG_ERROR_TRUNC
AP_INVALID_VERB
AP_TP_BUSY

APPC does not return secondary return codes with these primary return codes.

The following primary return code is returned by MC_RECEIVE_AND_POST:

primary_rc

AP_DEALLOC_ABEND

APPC does not return a secondary return code with this primary return code.

The following primary return codes are returned by RECEIVE_AND_POST:

primary_rc

AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_SVC_ERROR_NO_TRUNC
AP_SVC_ERROR_PURGING
AP_SVC_ERROR_TRUNC

APPC does not return secondary return codes with these primary return codes.

State When Issued

The TP can issue [MC_]RECEIVE_AND_POST when the conversation is in Receive, Send, or Send_Pending state.

Issuing the Verb in Send State

Issuing the [MC_]RECEIVE_AND_POST verb while the conversation is in Send state has the following effects:

- The local LU sends the information in its send buffer and a SEND indicator to the partner TP.
- The conversation changes to Pending_Post state; the local TP is ready to receive information from the partner TP asynchronously.

State Change

The conversation changes state twice. On the initial return of the verb, if the *primary_rc* is AP_OK, the conversation changes to Pending_Post state. After APPC calls the callback routine or clears the semaphore to indicate completion of the verb, the state changes as described in this section.

The state change on completion of [MC_]RECEIVE_AND_POST depends on the value of the following:

- The *primary_rc* parameter
- The *what_rcvd* parameter if *primary_rc* is AP_OK

The table that follows summarizes the possible state changes that can occur when *primary_rc* is AP_OK:

what_rcvd **parameter**	**New state**
AP_CONFIRM_WHAT_RECEIVED AP_DATA_CONFIRM AP_DATA_COMPLETE_CONFIRM	Confirm
AP_CONFIRM_DEALLOCATE AP_DATA_CONFIRM_DEALLOCATE AP_DATA_COMPLETE_CONFIRM_DEALL	Confirm_Deallocate
AP_CONFIRM_SEND AP_DATA_CONFIRM_SEND AP_DATA_COMPLETE_CONFIRM_SEND	Confirm_Send

what_rcvd **parameter**	**New state**
AP_DATA AP_DATA_COMPLETE AP_DATA_INCOMPLETE	Receive
AP_SEND	Send
AP_DATA_SEND AP_DATA_COMPLETE_SEND	Send_ Pending

The table that follows summarizes the possible state changes that can occur when *primary_rc* is not AP_OK:

primary_rc	**New state**
AP_PARAMETER_CHECK AP_STATE_CHECK AP_CONVERSATION_TYPE_MIXED	No change (these return codes can only occur as the first return code, not as the second return code)
AP_INVALID_VERB AP_INVALID_VERB_SEGMENT	No change
AP_STACK_TOO_SMALL AP_TP_BUSY AP_UNEXPECTED_DOS_ERROR	No change
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	Reset
AP_PROG_ERROR_PURGING AP_PROG_ERROR_NO_TRUNC AP_SVC_ERROR_PURGING AP_SVC_ERROR_NO_TRUNC AP_PROG_ERROR_TRUNC AP_SVC_ERROR_TRUNC	Receive
AP_CANCELLED	The conversation returns to the state (Send or Receive) in which the [MC_]RECEIVE_AND_POST verb was issued. Since the AP_CANCELLED return code is caused by another verb issued by the same TP, the conversation state will then change again when this later verb completes.

Usage Notes®

This section provides additional usage information about the following topics:

- PS Header data
- Callback routine
- Processing while the verb is pending
- Compatibility with other APPC implementations
- How the TP uses the verb
- Avoiding indefinite waits

PS Header Data

In a conversation with a synchronization level of AP_SYNCPT, the received data may be in PS Header format. In a mapped conversation, this is indicated by the value of the *what_rcvd* parameter; in a basic conversation, this is indicated by an LL field of 0x0001 (see "Logical Records" on page 58 for more information). The Syncpoint Manager is responsible for converting the data into the appropriate Syncpoint commands.

Callback Routine

UNIX

The application supplies a pointer to a callback routine as one of the parameters to the VCB. This section describes how Communications Server uses this routine, and the functions that it must perform.

The callback routine is defined as follows:

```
void (*callback) (
                    void *          vcb,
                    unsigned char   tp_id[8],
                    AP_UINT32       conv_id
                  );
```

Communications Server calls the routine with the following parameters:

vcb Pointer to the VCB supplied by the application, including the returned parameters set by Communications Server.

tp_id The 8-byte TP identifier of the TP in which the verb was issued.

conv_id
The conversation identifier of the conversation in which the verb was issued.

The callback routine need not use all of these parameters. It may perform all the necessary processing on the returned VCB, or may simply set a variable to inform the main program that the verb has completed.

The application can issue further APPC verbs from within the callback routine, if required. However, these must be asynchronous verbs. Any synchronous verbs issued from within a callback routine will be rejected with the return codes AP_PARAMETER_CHECK and AP_SYNC_NOT_ALLOWED.

If the application issues the [MC_]RECEIVE_AND_POST verb using the asynchronous APPC entry point, there are two callback routines specified: one in the VCB, the other supplied as a parameter to the entry point. In general, APPC uses the callback routine specified in the VCB and ignores the one on the entry point; however, if the application supplies a null pointer for the callback routine in the VCB, APPC uses the callback routine on the entry point.

Continuing with Other Processing While the Verb Is Pending

Because the [MC_]RECEIVE_AND_POST verb returns immediately without waiting for data to arrive, the TP can continue other processing while waiting for it to complete. However, the following points should be noted:

- The VCB supplied to the [MC_]RECEIVE_AND_POST verb continues to be used until the callback routine returns. The TP must not change any fields in the VCB during this time. If it issues any other APPC verb while in Pending_Post state, it must use another VCB for it.
- Only one [MC_]RECEIVE_AND_POST verb per conversation can be active at any time.

Compatibility with Other APPC Implementations

UNIX

The AIX or Linux implementation of the [MC_]RECEIVE_AND_POST verb is different from Windows APPC implementations. In addition, this verb is not available in any DOS implementations of APPC. Because of this, TPs using [MC_]RECEIVE_AND_POST are not totally portable to other operating systems; if your TP uses this verb, you will need to rewrite the sections of the TP that use it if you want the TP to run on other operating systems.

How the TP Uses the Verb

To use the [MC_]RECEIVE_AND_POST verb, the local TP performs the following steps:

1. Issues the [MC_]RECEIVE_AND_POST verb.
2. Checks the value of the primary return code *primary_rc*.

   If the primary return code is `AP_OK`, the receive buffer (pointed to by the *dptr* parameter) is asynchronously receiving data from the partner TP. While receiving data asynchronously, the local TP can do the following:
   - Perform tasks not related to this conversation
   - Issue the [MC_]REQUEST_TO_SEND verb
   - Gather information about this conversation by issuing the following verbs:
     - GET_TYPE
     - [MC_]GET_ATTRIBUTES
     - [MC_]TEST_RTS
   - Prematurely cancel the [MC_]RECEIVE_AND_POST verb by issuing one of the following verbs:
     - DEALLOCATE with *dealloc_type* set to `AP_ABEND_PROG`, `AP_ABEND_SVC`, or `AP_ABEND_TIMER`
     - MC_DEALLOCATE with *dealloc_type* set to `AP_ABEND`
     - SEND_ERROR
     - TP_ENDED
3. Checks that the callback routine (supplied as a parameter on this verb) has been called by APPC. When the TP finishes receiving data asynchronously, APPC calls this routine.
4. Checks the new value of the primary return code *primary_rc*.

   If the primary return code is `AP_OK`, the local TP can examine the other returned parameters and manipulate the asynchronously received data.

   If the primary return code is not `AP_OK`, only the *secondary_rc* and *rts_rcvd* (request-to-send received) parameters are meaningful.

Avoiding Indefinite Waits

UNIX

If the local TP issues the [MC_]RECEIVE_AND_POST verb and subsequently waits for the callback routine to be called, it will be suspended until information is received from the partner TP. It could wait indefinitely if the partner TP does not send any information, or does not issue a verb causing the partner LU to flush its send buffer. If you need to have the TP operating continuously, avoid waiting on the callback routine, or use the [MC_]RECEIVE_IMMEDIATE verb.

MC_RECEIVE_AND_WAIT and RECEIVE_AND_WAIT

The MC_RECEIVE_AND_WAIT or RECEIVE_AND_WAIT verb receives any data that is currently available from the partner TP. If no data is currently available, the local TP waits for data to arrive.

While an asynchronous [MC_]RECEIVE_AND_WAIT is outstanding, the application can issue the following verbs on the same conversation:

- GET_TYPE
- [MC_]DEALLOCATE with a deallocate type of AP_ABEND, AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER
- [MC_]GET_ATTRIBUTES
- Additional [MC_]RECEIVE verbs, provided that they are issued in non-blocking mode
- [MC_]RECEIVE_EXPEDITED_DATA
- [MC_]REQUEST_TO_SEND
- [MC_]SEND_DATA (full-duplex conversations only)
- [MC_]SEND_EXPEDITED_DATA
- [MC_]SEND_ERROR
- [MC_]TEST_RTS
- TP_ENDED

VCB Structure: MC_RECEIVE_AND_WAIT

UNIX

The definition of the VCB structure for the MC_RECEIVE_AND_WAIT verb is as follows:

```
typedef struct mc_receive_and_wait
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                 /* Reserved             */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  AP_UINT16        what_rcvd;
  unsigned char    rtn_status;
```

```
  unsigned char    reserv4;
  unsigned char    rts_rcvd;
  unsigned char    expd_rcvd;
  AP_UINT16        max_len;
  AP_UINT16        dlen;
  unsigned char    *dptr;
  unsigned char    reserv6[5];
} MC_RECEIVE_AND_WAIT;
```

VCB Structure: RECEIVE_AND_WAIT

The definition of the VCB structure for the RECEIVE_AND_WAIT verb is as follows:

```
typedef struct receive_and_wait
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;               /* Reserved            */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  AP_UINT16        what_rcvd;
  unsigned char    rtn_status;
  unsigned char    fill;
  unsigned char    rts_rcvd;
  unsigned char    expd_rcvd;
    AP_UINT16        max_len;
  AP_UINT16        dlen;
  unsigned char    *dptr;
  unsigned char    reserv5[5];
} RECEIVE_AND_WAIT;
```

VCB Structure: MC_RECEIVE_AND_WAIT (Windows)

WINDOWS

The definition of the VCB structure for the MC_RECEIVE_AND_WAIT verb is as follows:

```
typedef struct mc_receive_and_wait
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned short    what_rcvd;
  unsigned char     rtn_status;
  unsigned char     reserv4;
  unsigned char     rts_rcvd;
  unsigned char     reserv5;
  unsigned short    max_len;
  unsigned short    dlen;
  unsigned char far *dptr;
  unsigned char     reserv6[5];
} MC_RECEIVE_AND_WAIT;
```

VCB Structure: RECEIVE_AND_WAIT (Windows)

The definition of the VCB structure for the RECEIVE_AND_WAIT verb is as follows:

```
typedef struct receive_and_wait
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned short    what_rcvd;
  unsigned char     rtn_status;
  unsigned char     fill;
  unsigned char     rts_rcvd;
  unsigned char     reserv4;
  unsigned short    max_len;
  unsigned short    dlen;
  unsigned char far *dptr;
  unsigned char     reserv5[5];
} RECEIVE_AND_WAIT;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_RECEIVE_AND_WAIT
For the MC_RECEIVE_AND_WAIT verb.

AP_B_RECEIVE_AND_WAIT
For the RECEIVE_AND_WAIT verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_RECEIVE_AND_WAIT verb.

AP_BASIC_CONVERSATION
For the RECEIVE_AND_WAIT verb.

If the verb is being issued on a full-duplex conversation or is being issued as a non-blocking verb, combine the value above (using a logical OR) with one or both of the following values:

AP_FULL_DUPLEX_CONVERSATION
The verb is being issued on a full-duplex conversation.

AP_NON_BLOCKING
The verb is being issued as a non-blocking verb.

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

rtn_status
Indicates whether status information and data can be returned on the same verb. Possible values are:

AP_YES Status information, if available, is returned with the last part of a data record.

AP_NO Status information is not returned with data. After receiving the end of a data record, the local TP must issue another [MC_]RECEIVE verb to obtain the status information.

fill Indicates the manner in which the local TP receives data.

This parameter is used only by the basic-conversation RECEIVE_AND_WAIT verb. Possible values are:

AP_BUFFER
The local TP receives data until the number of bytes specified by the *max_len* parameter is reached or until end of data. Data is received without regard for the logical-record format.

AP_LL Data is received in logical-record format. The data received can be any of the following:

- A complete logical record
- A *max_len*-byte portion of a logical record
- The end of a logical record

max_len
Maximum number of bytes of data the local TP can receive.

The range for this value is `0–65,535`.

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.

WINDOWS

The data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
`AP_OK`

what_rcvd
Status information received with the incoming data.

The next action taken by the TP will usually depend on the value of this parameter. For more information, see "The what_rcvd Parameter" on page 151.

Possible values are:

AP_CONFIRM_DEALLOCATE
This value can be returned only in a half-duplex conversation.

The partner TP has issued the [MC_]DEALLOCATE verb with *dealloc_type* set to AP_SYNC_LEVEL, and the conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_CONFIRM_SEND
This value can be returned only in a half-duplex conversation.

The partner TP has issued the [MC_]PREPARE_TO_RECEIVE verb with *ptr_type* set to AP_SYNC_LEVEL, and the conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_CONFIRM_WHAT_RECEIVED
This value can be returned only in a half-duplex conversation.

The partner TP has issued the [MC_]CONFIRM verb.

AP_DATA
This value can be returned by the basic-conversation RECEIVE_AND_WAIT verb if the *fill* parameter is set to AP_BUFFER; it is not applicable to MC_RECEIVE_AND_WAIT.

The local TP received data until *max_len* or end of data was reached.

AP_DATA_COMPLETE
For the mapped-conversation MC_RECEIVE_AND_WAIT verb, this value indicates that the local TP has received a complete data record or the last part of a data record.

For the basic-conversation RECEIVE_AND_WAIT verb with the *fill* parameter set to AP_LL, this value indicates that the local TP has received a complete logical record or the end of a logical record.

AP_DATA_INCOMPLETE
For the mapped-conversation MC_RECEIVE_AND_WAIT verb, this value indicates that the local TP has received an incomplete data record. The *max_len* parameter specified a value less than the length of the data record (or less than the remainder of the data record if this is not the first receive verb to read the record).

For the basic-conversation RECEIVE_AND_WAIT verb with the *fill* parameter set to AP_LL, this value indicates that the local TP has received an incomplete logical record.

AP_SEND
This value can be returned only in a half-duplex conversation.

For the partner TP, the conversation has entered Receive state. For the local TP, the conversation is now in Send state.

The following values will be returned only in a half-duplex conversation, and only if *rtn_status* was set to AP_YES:

AP_DATA_CONFIRM
This is a combination of AP_DATA and AP_CONFIRM_WHAT_RECEIVED. The partner TP sent data and then issued the [MC_]CONFIRM verb, or issued the [MC_]SEND_DATA verb with a send type of CONFIRM.

AP_DATA_COMPLETE_CONFIRM
This is a combination of AP_DATA_COMPLETE and AP_CONFIRM_WHAT_RECEIVED. The partner TP sent a complete data record (or the end of a data record) and then issued the [MC_]CONFIRM verb, or issued the [MC_]SEND_DATA verb with a send type of CONFIRM.

AP_DATA_CONFIRM_DEALLOCATE
This is a combination of AP_DATA and AP_CONFIRM_DEALLOCATE. The partner TP sent data and then issued the [MC_]DEALLOCATE verb with *dealloc_type* set to AP_SYNC_LEVEL or issued the [MC_]SEND_DATA verb with a send type of DEALLOC_SYNC_LEVEL. The conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_DATA_COMPLETE_CONFIRM_DEALL
This is a combination of AP_DATA_COMPLETE and AP_CONFIRM_DEALLOCATE. The partner TP sent a complete data record (or the end of a data record) and then issued the [MC_]DEALLOCATE verb with *dealloc_type* set to AP_SYNC_LEVEL, or issued the [MC_]SEND_DATA verb with a send type of DEALLOC_SYNC_LEVEL. The conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_DATA_CONFIRM_SEND
This is a combination of AP_DATA and AP_CONFIRM_SEND. The partner TP sent data and then issued the [MC_]PREPARE_TO_RECEIVE verb with *ptr_type* set to AP_SYNC_LEVEL, or issued the [MC_]SEND_DATA verb with a send type of P_TO_R_SYNC_LEVEL. The conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_DATA_COMPLETE_CONFIRM_SEND
This is a combination of AP_DATA_COMPLETE and AP_CONFIRM_SEND. The partner TP sent a complete data record (or the end of a data record) and then issued the [MC_]PREPARE_TO_RECEIVE verb with *ptr_type* set to AP_SYNC_LEVEL, or issued the [MC_]SEND_DATA verb with a send type of P_TO_R_SYNC_LEVEL. The conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_DATA_SEND
The partner TP sent data and then entered Receive state. For the local TP, the conversation is now in Send_Pending state.

AP_DATA_COMPLETE_SEND
The partner TP sent a complete data record (or the end of a data record) and then entered Receive state. For the local TP, the conversation is now in Send_Pending state.

UNIX

The following values will be returned on the MC_RECEIVE_AND_WAIT verb:

AP_USER_CONTROL_DATA_INCMP
As for AP_DATA_INCOMPLETE, except that the received data was in User Control Data format.

AP_USER_CONTROL_DATA_COMPLETE
As for AP_DATA_COMPLETE, except that the received data was in User Control Data format.

AP_UC_DATA_COMPLETE_SEND
As for AP_DATA_COMPLETE_SEND, except that the received data was in User Control Data format.

AP_UC_DATA_COMPLETE_CONFIRM
As for AP_DATA_COMPLETE_CONFIRM, except that the received data was in User Control Data format.

AP_UC_DATA_COMPLETE_CNFM_DEALL
As for AP_DATA_COMPLETE_CONFIRM_DEALL, except that the received data was in User Control Data format.

AP_UC_DATA_COMPLETE_CNFM_SEND
As for AP_DATA_COMPLETE_CONFIRM_SEND, except that the received data was in User Control Data format.

The following values will be returned on the MC_RECEIVE_AND_WAIT verb with *sync_level* set to AP_SYNCPT:

AP_PS_HEADER_INCOMPLETE
As for AP_DATA_INCOMPLETE, except that the received data was in PS Header format.

AP_PS_HEADER_COMPLETE
As for AP_DATA_COMPLETE, except that the received data was in PS Header format.

AP_PS_HDR_COMPLETE_SEND
As for AP_DATA_COMPLETE_SEND, except that the received data was in PS Header format.

AP_PS_HDR_COMPLETE_CONFIRM
As for AP_DATA_COMPLETE_CONFIRM, except that the received data was in PS Header format.

AP_PS_HDR_COMPLETE_CNFM_DEALL
As for AP_DATA_COMPLETE_CONFIRM_DEALL, except that the received data was in PS Header format.

AP_PS_HDR_COMPLETE_CNFM_SEND
As for AP_DATA_COMPLETE_CONFIRM_SEND, except that the received data was in PS Header format.

rts_rcvd
Request-to-send-received indicator. This parameter applies only in a half-duplex conversation; it is not used in a full-duplex conversation.

Possible values are:

AP_YES The partner TP has issued the [MC_]REQUEST_TO_SEND verb, which requests that the local TP change the conversation to Receive state.

AP_NO The partner TP has not issued the [MC_]REQUEST_TO_SEND verb.

For an explanation of why this indicator can be received by RECEIVE verbs, see "MC_REQUEST_TO_SEND and REQUEST_TO_SEND" on page 196.

expd_rcvd

Expedited data indicator.

Possible values are:

AP_YES The partner TP has sent expedited data that the local TP has not yet received. To receive this data, the local TP can use the [MC_]RECEIVE_EXPEDITED_DATA verb.

This indicator can be set on a number of APPC verbs. It continues to be set on subsequent verbs until the local TP issues the [MC_]RECEIVE_EXPEDITED_DATA verb to receive the data.

AP_NO There is no expedited data waiting to be received.

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 `0` (zero) indicates that no data was received.

Conversation Deallocated: If the partner TP has deallocated the conversation without requesting confirmation, APPC returns the following parameters:

primary_rc

AP_DEALLOC_NORMAL
The partner TP issued the [MC_]DEALLOCATE verb with *dealloc_type* set to one of the following:

- `AP_FLUSH`
- `AP_SYNC_LEVEL` with the synchronization level of the conversation specified as `AP_NONE`

dlen Number of bytes of data received (the data is stored in the buffer specified by the *dptr* parameter). A length of `0` (zero) indicates that no data was received. This parameter is only used if *rtn_status* was set to `AP_YES`.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc

`AP_PARAMETER_CHECK`

secondary_rc
Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_RETURN_STATUS_WITH_DATA
The *rtn_status* parameter was set to a value that was not valid.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

UNIX

AP_INVALID_FORMAT
The reserved field *format* was set to a nonzero value.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

WINDOWS

AP_INVALID_DATA_SEGMENT
The data was longer than the allocated data segment, or the address of the data buffer was incorrect.

AP_RCV_AND_WAIT_BAD_FILL
This return code applies only to the basic-conversation RECEIVE_AND_WAIT verb. The *fill* parameter was set to a value that was not valid.

State Check: If the conversation is in the wrong state when the TP issues this verb, APPC returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
Possible values are:

AP_RCV_AND_WAIT_BAD_STATE
The conversation was not in Receive, Send, or Send_Pending state when the TP issued this verb.

AP_RCV_AND_WAIT_NOT_LL_BDY
This return code applies only to the basic-conversation RECEIVE_AND_WAIT verb. The conversation was in Send state; the TP began but did not finish sending a logical record.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc

AP_ALLOCATION_ERROR

secondary_rc

AP_ALLOCATION_FAILURE_NO_RETRY
AP_ALLOCATION_FAILURE_RETRY
AP_CONVERSATION_TYPE_MISMATCH
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_SECURITY_NOT_VALID
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_TP_NAME_NOT_RECOGNIZED
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_SEC_BAD_PROTOCOL_VIOLATION
AP_SEC_BAD_PASSWORD_EXPIRED
AP_SEC_BAD_PASSWORD_INVALID
AP_SEC_BAD_USERID_REVOKED
AP_SEC_BAD_USERID_INVALID
AP_SEC_BAD_USERID_MISSING
AP_SEC_BAD_PASSWORD_MISSING
AP_SEC_BAD_UID_NOT_DEFD_TO_GRP
AP_SEC_BAD_UNAUTHRZD_AT_RLU
AP_SEC_BAD_UNAUTHRZD_FROM_LLU
AP_SEC_BAD_UNAUTHRZD_TO_TP
AP_SEC_BAD_INSTALL_EXIT_FAILED
AP_SEC_BAD_PROCESSING_FAILURE

UNIX

primary_rc

AP_BACKED_OUT

secondary_rc

AP_BO_NO_RESYNC
AP_BO_RESYNC

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT

AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_CONVERSATION_TYPE_MIXED

```
AP_DUPLEX_TYPE_MIXED
AP_PROG_ERROR_NO_TRUNC
AP_PROG_ERROR_PURGING
AP_PROG_ERROR_TRUNC
AP_INVALID_VERB
AP_TP_BUSY
```

APPC does not return secondary return codes with these primary return codes.

The following primary return code is returned by the MC_RECEIVE_AND_WAIT verb:

primary_rc

```
AP_DEALLOC_ABEND
```

APPC does not return a secondary return code with this primary return code.

The following primary return codes are returned by the RECEIVE_AND_WAIT verb:

primary_rc

```
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_SVC_ERROR_NO_TRUNC
AP_SVC_ERROR_PURGING
AP_SVC_ERROR_TRUNC
```

APPC does not return secondary return codes with these primary return codes.

State When Issued

The TP can issue the [MC_]RECEIVE_AND_WAIT verb when the conversation is in Receive, Send, or Send_Pending state.

Issuing the Verb in Send State (half-duplex conversation only)

Issuing the [MC_]RECEIVE_AND_WAIT verb while the conversation is in Send state has the following effects:

- The local LU sends the information in its send buffer and a SEND indicator to the partner TP.
- The conversation changes to Receive state; the local TP waits to receive information from the partner TP.

State Change

WINDOWS

When the verb is issued to the asynchronous entry point, the conversation changes state twice. On the initial return of the verb, if the *primary_rc* is AP_OK, the conversation changes to Pending_Post state. After APPC indicates completion of the verb, the state changes as described below. For more information about the actions that the application can take in Pending_Post state, see "MC_RECEIVE_AND_POST and RECEIVE_AND_POST" on page 153.

The state change after the [MC_]RECEIVE_AND_WAIT verb depends on the value of the following:

- The *primary_rc* parameter
- The *what_rcvd* parameter

The possible state changes are summarized in the following tables.

what_rcvd **parameter**	**New state**
AP_CONFIRM_WHAT_RECEIVED	Confirm
AP_DATA_CONFIRM	
AP_DATA_COMPLETE_CONFIRM	
AP_CONFIRM_DEALLOCATE	Confirm_Deallocate
AP_DATA_CONFIRM_DEALLOCATE	
AP_DATA_COMPLETE_CONFIRM_DEALL	
AP_CONFIRM_SEND	Confirm_Send
AP_DATA_CONFIRM_SEND	
AP_DATA_COMPLETE_CONFIRM_SEND	
AP_DATA	Receive (half-duplex conversation) or no change (full-duplex conversation)
AP_DATA_COMPLETE	
AP_DATA_INCOMPLETE	
AP_SEND	Send
AP_DATA_SEND	Send_Pending
AP_DATA_COMPLETE_SEND	

primary_rc	**New state**
AP_PARAMETER_CHECK	No change
AP_STATE_CHECK	
AP_CONVERSATION_TYPE_MIXED	
AP_INVALID_VERB	
AP_INVALID_VERB_SEGMENT	
AP_STACK_TOO_SMALL	
AP_TP_BUSY	
AP_UNEXPECTED_DOS_ERROR	
AP_CONV_FAILURE_RETRY	Reset
AP_CONV_FAILURE_NO_RETRY	
AP_DEALLOC_ABEND	Reset
AP_DEALLOC_ABEND_PROG	
AP_DEALLOC_ABEND_SVC	
AP_DEALLOC_ABEND_TIMER	
AP_DEALLOC_NORMAL	Reset
AP_PROG_ERROR_PURGING	Receive (half-duplex conversation) or no change (full-duplex conversation)
AP_PROG_ERROR_NO_TRUNC	
AP_SVC_ERROR_PURGING	
AP_SVC_ERROR_NO_TRUNC	
AP_PROG_ERROR_TRUNC	
AP_SVC_ERROR_TRUNC	

Usage Notes

This section provides additional usage information about the following topics:

- PS header data
- Avoiding indefinite waits

PS Header Data

UNIX

In a conversation with a synchronization level of AP_SYNCPT, the received data may be in PS Header format. In a mapped conversation, this is indicated by the value of the *what_rcvd* parameter; in a basic conversation, this is indicated by an LL field of 0x0001 (for more information, see "Logical Records" on page 58). The Syncpoint Manager is responsible for converting the data into the appropriate Syncpoint commands.

Avoiding Indefinite Waits

If the local TP issues the [MC_]RECEIVE_AND_WAIT verb, it will be suspended until information is received from the partner TP. It could wait indefinitely if the partner TP does not send any information, or does not issue a verb causing the partner LU to flush its send buffer. If you need to have the TP operating continuously, use the [MC_]RECEIVE_AND_POST verb but avoid waiting on the callback routine, or use the [MC_]RECEIVE_IMMEDIATE verb.

MC_RECEIVE_IMMEDIATE and RECEIVE_IMMEDIATE

The MC_RECEIVE_IMMEDIATE or RECEIVE_IMMEDIATE verb receives any data and/or status information which is currently available from the partner TP. If none is currently available, the local TP returns immediately and does not wait.

WINDOWS

Although this verb does not wait to receive information, it is still possible that the Windows APPC library will yield to allow other processing to continue. Do not assume that the verb will return without yielding.

VCB Structure: MC_RECEIVE_IMMEDIATE

UNIX

The definition of the VCB structure for the MC_RECEIVE_IMMEDIATE verb is as follows:

```
typedef struct mc_receive_immediate
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                 /* Reserved            */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  AP_UINT16        what_rcvd;
  unsigned char    rtn_status;
```

```
  unsigned char    reserv4;
  unsigned char    rts_rcvd;
  unsigned char    expd_rcvd;
  AP_UINT16        max_len;
  AP_UINT16        dlen;
  unsigned char    *dptr;
  unsigned char    reserv6[5];
} MC_RECEIVE_IMMEDIATE;
```

VCB Structure: RECEIVE_IMMEDIATE

The definition of the VCB structure for the RECEIVE_IMMEDIATE verb is as follows:

```
typedef struct receive_immediate
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                  /* Reserved             */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  AP_UINT16        what_rcvd;
  unsigned char    rtn_status;
  unsigned char    fill;
  unsigned char    rts_rcvd;
  unsigned char    expd_rcvd;
  AP_UINT16        max_len;
  AP_UINT16        dlen;
  unsigned char    *dptr;
  unsigned char    reserv5[5];
} RECEIVE_IMMEDIATE;
```

VCB Structure: MC_RECEIVE_IMMEDIATE (Windows)

WINDOWS

The definition of the VCB structure for the MC_RECEIVE_IMMEDIATE verb is as follows:

```
typedef struct mc_receive_immediate
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned short    what_rcvd;
  unsigned char     rtn_status;
  unsigned char     reserv4;
  unsigned char     rts_rcvd;
  unsigned char     reserv5;
  unsigned short    max_len;
  unsigned short    dlen;
  unsigned char far *dptr;
  unsigned char     reserv6[5];
} MC_RECEIVE_IMMEDIATE;
```

VCB Structure: RECEIVE_IMMEDIATE (Windows)

The definition of the VCB structure for the RECEIVE_IMMEDIATE verb is as follows:

```
typedef struct receive_immediate
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned short    what_rcvd;
  unsigned char     rtn_status;
  unsigned char     fill;
  unsigned char     rts_rcvd;
  unsigned char     reserv4;
  unsigned short    max_len;
  unsigned short    dlen;
  unsigned char far *dptr;
  unsigned char     reserv5[5];
} RECEIVE_IMMEDIATE;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_RECEIVE_IMMEDIATE
For the MC_RECEIVE_IMMEDIATE verb.

AP_B_RECEIVE_IMMEDIATE
For the RECEIVE_IMMEDIATE verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_RECEIVE_IMMEDIATE verb.

AP_BASIC_CONVERSATION
For the RECEIVE_IMMEDIATE verb.

If the verb is being issued on a full-duplex conversation or is being issued as a non-blocking verb, combine the value above (using a logical OR) with one or both of the following values:

AP_FULL_DUPLEX_CONVERSATION
The verb is being issued on a full-duplex conversation.

AP_NON_BLOCKING
The verb is being issued as a non-blocking verb.

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

rtn_status
Indicates whether status information and data can be returned on the same verb. Possible values are:

AP_YES Status information, if available, is returned with the last part of a data record.

AP_NO Status information is not returned with data. After receiving the end of a data record, the local TP must issue another [MC_]RECEIVE verb to obtain the status information.

fill Indicates the manner in which the local TP receives data.

This parameter is used only by the basic-conversation RECEIVE_IMMEDIATE verb. Possible values are:

AP_BUFFER
The local TP receives data until the number of bytes specified by the *max_len* parameter is reached or until end of data. Data is received without regard for the logical-record format.

AP_LL Data is received in logical-record format. The data received can be any of the following:

- A complete logical record
- A *max_len*-byte portion of a logical record
- The end of a logical record

max_len
Maximum number of bytes of data the local TP can receive.

The range for this value is `0–65,535`.

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.

WINDOWS

The data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
`AP_OK`

what_rcvd
Status information received with the incoming data.

The next action taken by the TP will usually depend on the value of this parameter. For more information, see "The what_rcvd Parameter" on page 151.

Possible values are:

AP_CONFIRM_DEALLOCATE
The partner TP has issued the [MC_]DEALLOCATE verb with *dealloc_type* set to AP_SYNC_LEVEL, and the conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_CONFIRM_SEND
The partner TP has issued the [MC_]PREPARE_TO_RECEIVE verb with *ptr_type* set to AP_SYNC_LEVEL, and the conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_CONFIRM_WHAT_RECEIVED
The partner TP has issued the [MC_]CONFIRM verb.

AP_DATA
This value can be returned by the basic-conversation RECEIVE_IMMEDIATE verb if the *fill* parameter is set to AP_BUFFER; it is not applicable to MC_RECEIVE_IMMEDIATE.

The local TP received data until *max_len* or end of data was reached.

AP_DATA_COMPLETE
For the mapped-conversation MC_RECEIVE_IMMEDIATE verb, this value indicates that the local TP has received a complete data record or the last part of a data record.

For the basic-conversation RECEIVE_IMMEDIATE verb with the *fill* parameter set to AP_LL, this value indicates that the local TP has received a complete logical record or the end of a logical record.

AP_DATA_INCOMPLETE
For the mapped-conversation MC_RECEIVE_IMMEDIATE verb, this value indicates that the local TP has received an incomplete data record. The *max_len* parameter specified a value less than the length of the data record (or less than the remainder of the data record if this is not the first receive verb to read the record).

For the basic-conversation RECEIVE_IMMEDIATE verb with the *fill* parameter set to AP_LL, this value indicates that the local TP has received an incomplete logical record.

AP_SEND
For the partner TP, the conversation has entered Receive state. For the local TP, the conversation is now in Send state.

The following values will only be returned if *rtn_status* was set to AP_YES:

AP_DATA_CONFIRM
This is a combination of AP_DATA and AP_CONFIRM_WHAT_RECEIVED. The partner TP sent data and then issued the [MC_]CONFIRM verb, or issued the [MC_]SEND_DATA verb with a send type of CONFIRM.

AP_DATA_COMPLETE_CONFIRM
This is a combination of AP_DATA_COMPLETE and

AP_CONFIRM_WHAT_RECEIVED. The partner TP sent a complete data record (or the end of a data record) and then issued the [MC_]CONFIRM verb, or issued the [MC_]SEND_DATA verb with a send type of CONFIRM.

AP_DATA_CONFIRM_DEALLOCATE
This is a combination of AP_DATA and AP_CONFIRM_DEALLOCATE. The partner TP sent data and then issued the [MC_]DEALLOCATE verb with *dealloc_type* set to AP_SYNC_LEVEL, or issued the [MC_]SEND_DATA verb with a send type of DEALLOC_SYNC_LEVEL. The conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_DATA_COMPLETE_CONFIRM_DEALL
This is a combination of AP_DATA_COMPLETE and AP_CONFIRM_DEALLOCATE. The partner TP sent a complete data record (or the end of a data record) and then issued the [MC_]DEALLOCATE verb with *dealloc_type* set to AP_SYNC_LEVEL, or issued the [MC_]SEND_DATA verb with a send type of DEALLOC_SYNC_LEVEL. The conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_DATA_CONFIRM_SEND
This is a combination of AP_DATA and AP_CONFIRM_SEND. The partner TP sent data and then issued the [MC_]PREPARE_TO_RECEIVE verb with *ptr_type* set to AP_SYNC_LEVEL, or issued the [MC_]SEND_DATA verb with a send type of P_TO_R_SYNC_LEVEL. The conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_DATA_COMPLETE_CONFIRM_SEND
This is a combination of AP_DATA_COMPLETE and AP_CONFIRM_SEND. The partner TP sent a complete data record (or the end of a data record) and then issued the [MC_]PREPARE_TO_RECEIVE verb with *ptr_type* set to AP_SYNC_LEVEL, or issued the [MC_]SEND_DATA verb with a send type of P_TO_R_SYNC_LEVEL. The conversation's synchronization level, established by the [MC_]ALLOCATE verb, is AP_CONFIRM_SYNC_LEVEL.

AP_DATA_SEND
The partner TP sent data and then entered Receive state. For the local TP, the conversation is now in Send_Pending state.

AP_DATA_COMPLETE_SEND
The partner TP sent a complete data record (or the end of a data record) and then entered Receive state. For the local TP, the conversation is now in Send_Pending state.

UNIX

The following values will be returned on the MC_RECEIVE_IMMEDIATE verb:

AP_USER_CONTROL_DATA_INCMP
As for AP_DATA_INCOMPLETE, except that the received data was in User Control Data format.

AP_USER_CONTROL_DATA_COMPLETE
As for AP_DATA_COMPLETE, except that the received data was in User Control Data format.

AP_UC_DATA_COMPLETE_SEND
As for AP_DATA_COMPLETE_SEND, except that the received data was in User Control Data format.

AP_UC_DATA_COMPLETE_CONFIRM
As for AP_DATA_COMPLETE_CONFIRM, except that the received data was in User Control Data format.

AP_UC_DATA_COMPLETE_CNFM_DEALL
As for AP_DATA_COMPLETE_CONFIRM_DEALL, except that the received data was in User Control Data format.

AP_UC_DATA_COMPLETE_CNFM_SEND
As for AP_DATA_COMPLETE_CONFIRM_SEND, except that the received data was in User Control Data format.

The following values will be returned on the MC_RECEIVE_IMMEDIATE verb with *sync_level* set to AP_SYNCPT:

AP_PS_HEADER_INCOMPLETE
As for AP_DATA_INCOMPLETE, except that the received data was in PS Header format.

AP_PS_HEADER_COMPLETE
As for AP_DATA_COMPLETE, except that the received data was in PS Header format.

AP_PS_HDR_COMPLETE_SEND
As for AP_DATA_COMPLETE_SEND, except that the received data was in PS Header format.

AP_PS_HDR_COMPLETE_CONFIRM
As for AP_DATA_COMPLETE_CONFIRM, except that the received data was in PS Header format.

AP_PS_HDR_COMPLETE_CNFM_DEALL
As for AP_DATA_COMPLETE_CONFIRM_DEALL, except that the received data was in PS Header format.

AP_PS_HDR_COMPLETE_CNFM_SEND
As for AP_DATA_COMPLETE_CONFIRM_SEND, except that the received data was in PS Header format.

rts_rcvd
Request-to-send-received indicator. This parameter applies only in a half-duplex conversation; it is not used in a full-duplex conversation.

Possible values are:

AP_YES The partner TP has issued the [MC_]REQUEST_TO_SEND verb, which requests that the local TP change the conversation to Receive state.

AP_NO The partner TP has not issued the [MC_]REQUEST_TO_SEND verb.

For an explanation of why this indicator can be received by RECEIVE verbs, see "MC_REQUEST_TO_SEND and REQUEST_TO_SEND" on page 196.

expd_rcvd
Expedited data indicator.

Possible values are:

AP_YES The partner TP has sent expedited data that the local TP has not yet received. To receive this data, the local TP can use the [MC_]RECEIVE_EXPEDITED_DATA verb.

This indicator can be set on a number of APPC verbs. It continues to be set on subsequent verbs until the local TP issues the [MC_]RECEIVE_EXPEDITED_DATA verb to receive the data.

AP_NO There is no expedited data waiting to be received.

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 `0` (zero) indicates that no data was received.

Conversation Deallocated: If the partner TP has deallocated the conversation without requesting confirmation, APPC returns the following parameters:

primary_rc

AP_DEALLOC_NORMAL
The partner TP issued the [MC_]DEALLOCATE verb with *dealloc_type* set to one of the following:

- `AP_FLUSH`
- `AP_SYNC_LEVEL` with the synchronization level of the conversation specified as `AP_NONE`.

dlen Number of bytes of data received (the data is stored in the buffer specified by the *dptr* parameter). A length of `0` (zero) indicates that no data was received. This parameter is only used if *rtn_status* was set to `AP_YES`.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
`AP_PARAMETER_CHECK`

secondary_rc
Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_RETURN_STATUS_WITH_DATA
The *rtn_status* parameter was set to a value that was not valid.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

UNIX

AP_INVALID_FORMAT
The reserved field *format* was set to a nonzero value.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

WINDOWS

AP_INVALID_DATA_SEGMENT
The data was longer than the allocated data segment, or the address of the data buffer was incorrect.

AP_RCV_IMMD_BAD_FILL
This return code applies only to the basic-conversation RECEIVE_IMMEDIATE verb. The *fill* parameter was set to a value that was not valid.

State Check: If the conversation is in the wrong state when the TP issues this verb, APPC returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
Possible values are:

AP_RCV_IMMD_BAD_STATE
The conversation was not in Receive state when the TP issued this verb.

No Data Available: If no data is immediately available from the partner TP, APPC returns the following parameter:

primary_rc
AP_UNSUCCESSFUL

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc
AP_ALLOCATION_ERROR

secondary_rc

AP_ALLOCATION_FAILURE_NO_RETRY
AP_ALLOCATION_FAILURE_RETRY
AP_CONVERSATION_TYPE_MISMATCH
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_SECURITY_NOT_VALID
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_TP_NAME_NOT_RECOGNIZED
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_SEC_BAD_PROTOCOL_VIOLATION
AP_SEC_BAD_PASSWORD_EXPIRED
AP_SEC_BAD_PASSWORD_INVALID
AP_SEC_BAD_USERID_REVOKED
AP_SEC_BAD_USERID_INVALID
AP_SEC_BAD_USERID_MISSING
AP_SEC_BAD_PASSWORD_MISSING
AP_SEC_BAD_UID_NOT_DEFD_TO_GRP
AP_SEC_BAD_UNAUTHRZD_AT_RLU
AP_SEC_BAD_UNAUTHRZD_FROM_LLU
AP_SEC_BAD_UNAUTHRZD_TO_TP
AP_SEC_BAD_INSTALL_EXIT_FAILED
AP_SEC_BAD_PROCESSING_FAILURE

UNIX

primary_rc

AP_BACKED_OUT

secondary_rc

AP_BO_NO_RESYNC
AP_BO_RESYNC

[redacted]

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_CONVERSATION_TYPE_MIXED
AP_DUPLEX_TYPE_MIXED
AP_PROG_ERROR_NO_TRUNC
AP_PROG_ERROR_PURGING
AP_PROG_ERROR_TRUNC
AP_INVALID_VERB
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT

APPC does not return secondary return codes with these primary return codes.

The following primary return code is returned by the MC_RECEIVE_IMMEDIATE verb:

primary_rc
AP_DEALLOC_ABEND

APPC does not return a secondary return code with this primary return code.

The following primary return codes are returned by the RECEIVE_IMMEDIATE verb:

primary_rc
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_SVC_ERROR_NO_TRUNC
AP_SVC_ERROR_PURGING
AP_SVC_ERROR_TRUNC

APPC does not return secondary return codes with these primary return codes.

State When Issued

The TP can issue the [MC_]RECEIVE_IMMEDIATE verb only when the conversation is in Send_Receive (full-duplex conversation only) or Receive state.

State Change

The state change after the [MC_]RECEIVE_IMMEDIATE verb depends on the value of the following:

- The *primary_rc* parameter
- The *what_rcvd* parameter if *primary_rc* is AP_OK

The possible state changes are summarized in the following tables.

what_rcvd **parameter**	**New state**
AP_CONFIRM_WHAT_RECEIVED AP_DATA_CONFIRM AP_DATA_COMPLETE_CONFIRM	Confirm
AP_CONFIRM_DEALLOCATE AP_DATA_CONFIRM_DEALLOCATE AP_DATA_COMPLETE_CONFIRM_DEALL	Confirm_Deallocate
AP_CONFIRM_SEND AP_DATA_CONFIRM_SEND AP_DATA_COMPLETE_CONFIRM_SEND	Confirm_Send
AP_DATA AP_DATA_COMPLETE AP_DATA_INCOMPLETE	Receive (half-duplex conversation) or no change (full-duplex conversation)
AP_SEND	Send
AP_DATA_SEND AP_DATA_COMPLETE_SEND	Send_Pending

primary_rc	**New state**
AP_PARAMETER_CHECK AP_STATE_CHECK AP_CONVERSATION_TYPE_MIXED AP_INVALID_VERB AP_INVALID_VERB_SEGMENT AP_STACK_TOO_SMALL AP_TP_BUSY AP_UNEXPECTED_DOS_ERROR	No change
AP_CONV_FAILURE_RETRY AP_CONV_FAILURE_NO_RETRY	Reset
AP_DEALLOC_ABEND AP_DEALLOC_ABEND_PROG AP_DEALLOC_ABEND_SVC AP_DEALLOC_ABEND_TIMER AP_DEALLOC_NORMAL	Reset (half-duplex conversation) or Send_Only (full-duplex conversation)
AP_PROG_ERROR_PURGING AP_PROG_ERROR_NO_TRUNC AP_SVC_ERROR_PURGING AP_SVC_ERROR_NO_TRUNC AP_PROG_ERROR_TRUNC AP_SVC_ERROR_TRUNC	Receive (half-duplex conversation) or no change (full-duplex conversation)

PS Header Data

UNIX

In a conversation with a synchronization level of AP_SYNCPT, the received data may be in PS Header format. In a mapped conversation, this is indicated by the value of the *what_rcvd* parameter; in a basic conversation, this is indicated by an LL field of 0x0001 (see "Logical Records" on page 58 for more information). The Syncpoint Manager is responsible for converting the data into the appropriate Syncpoint commands.

MC_RECEIVE_EXPEDITED_DATA and RECEIVE_EXPEDITED_DATA

The MC_RECEIVE_EXPEDITED_DATA or RECEIVE_EXPEDITED_DATA verb receives any expedited data that is currently available from the partner TP. If no data is currently available, the verb can either return immediately or wait for data to arrive.

VCB Structure: MC_RECEIVE_EXPEDITED_DATA

The definition of the VCB structure for the MC_RECEIVE_EXPEDITED_DATA verb is as follows:

```
typedef struct mc_receive_expedited_data
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                 /* Reserved             */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
```

```
  AP_UINT32        conv_id;
  unsigned char    rtn_ctl;
  unsigned char    reserv1[3];
  unsigned char    rts_rcvd;
  unsigned char    expd_rcvd;
  AP_UINT16        max_len;
  AP_UINT16        dlen;
  unsigned char    *dptr;
} MC_RECEIVE_EXPEDITED_DATA;
```

VCB Structure: RECEIVE_EXPEDITED_DATA

The definition of the VCB structure for the RECEIVE_EXPEDITED_DATA verb is as follows:

```
typedef struct receive_expedited_data
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                 /* Reserved             */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  unsigned char    rtn_ctl;
  unsigned char    reserv1[3];
  unsigned char    rts_rcvd;
  unsigned char    expd_rcvd;
  AP_UINT16        max_len;
  AP_UINT16        dlen;
  unsigned char    *dptr;
} RECEIVE_EXPEDITED_DATA;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_RECEIVE_EXPEDITED_DATA
For the MC_RECEIVE_EXPEDITED_DATA verb.

AP_B_RECEIVE_EXPEDITED_DATA
For the RECEIVE_EXPEDITED_DATA verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_RECEIVE_EXPEDITED_DATA verb.

AP_BASIC_CONVERSATION
For the RECEIVE_EXPEDITED_DATA verb.

If the verb is being issued on a full-duplex conversation or is being issued as a non-blocking verb, combine the value above (using a logical OR) with one or both of the following values:

AP_FULL_DUPLEX_CONVERSATION
The verb is being issued on a full-duplex conversation.

AP_NON_BLOCKING
The verb is being issued as a non-blocking verb.

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

rtn_ctl Indicates when the verb should return control to the TP if no expedited data is available at the time it is issued. Possible values are:

AP_IMMEDIATE
If no expedited data is available, the verb returns immediately with a return code indicating this.

AP_WHEN_EXPD_RCVD
If no expedited data is available, the verb waits for data to arrive. It may wait indefinitely if the partner TP does not send any expedited data.

max_len
Maximum number of bytes of data the local TP can receive.

The range for this value is 0–86.

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.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

expd_rcvd
Expedited data indicator.

Possible values are:

AP_YES The partner TP has sent further expedited data that the local TP has not yet received, in addition to the data returned on this verb. To receive this data, the local TP can issue the [MC_]RECEIVE_EXPEDITED_DATA verb again.

This indicator can be set on a number of APPC verbs. It continues to be set on subsequent verbs until the local TP issues the [MC_]RECEIVE_EXPEDITED_DATA verb to receive the data.

AP_NO There is no expedited data waiting to be received.

dlen Number of bytes of data received (the data is stored in the buffer specified by the *dptr* parameter). A length of 0 (zero) indicates that no data was received.

Any data received is unformatted, and does not contain a two-byte length field (LL).

No Data Available: If the *rtn_ctl* parameter was set to AP_IMMEDIATE and no expedited data was available, APPC returns the following parameter:

primary_rc

AP_UNSUCCESSFUL

Conversation Deallocated: If the partner TP has deallocated the conversation, APPC returns one of the following values:

primary_rc

AP_DEALLOC_NORMAL
The partner TP issued the [MC_]DEALLOCATE verb with *dealloc_type* set to one of the following:

- AP_FLUSH
- AP_SYNC_LEVEL with the synchronization level of the conversation specified as AP_NONE

primary_rc

AP_CONVERSATION_ENDED
This verb was issued as a non-blocking verb and was queued behind an earlier verb. The partner TP issued the [MC_]DEALLOCATE verb as for AP_DEALLOC_NORMAL above, and the first verb in the queue returned with *primary_rc* set to AP_DEALLOC_NORMAL, indicating the end of the conversation. Any subsequent verbs in the queue then return with *primary_rc* set to AP_CONVERSATION_ENDED, indicating that the conversation had already ended before the verb could be processed.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

Expedited Data Not Supported: If the verb does not execute because the remote LU does not support expedited data, APPC returns the following parameter:

primary_rc

AP_EXPD_NOT_SUPPORTED_BY_LU

Data Buffer Too Small: If the verb does not execute because the TP's data buffer is too small to contain all of the available expedited data, APPC returns the following parameter:

primary_rc

AP_BUFFER_TOO_SMALL

dlen Number of bytes of expedited data available at the LU.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

AP_INVALID_FORMAT
The reserved field *format* was set to a nonzero value.

AP_EXPD_BAD_RETURN_CONTROL
The *rtn_ctl* parameter was set to a value that was not valid.

AP_RCV_EXPD_INVALID_LENGTH
The *max_len* parameter was set to a value that was not valid.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

State Check: If the conversation is in the wrong state when the TP issues this verb, APPC returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
Possible values are:

AP_EXPD_DATA_BAD_CONV_STATE
The conversation was in Reset state when the TP issued this verb.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc
AP_ALLOCATION_ERROR

secondary_rc
AP_CONVERSATION_TYPE_MISMATCH
AP_DUPLEX_TYPE_MIXED
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_SECURITY_NOT_VALID
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_TP_NAME_NOT_RECOGNIZED
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
AP_TRANS_PGM_NOT_AVAIL_RETRY

UNIX

primary_rc
AP_BACKED_OUT

secondary_rc
AP_BO_NO_RESYNC
AP_BO_RESYNC

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
AP_UNEXPECTED_SYSTEM_ERROR
AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_CONVERSATION_TYPE_MIXED
AP_PROG_ERROR_NO_TRUNC
AP_PROG_ERROR_PURGING
AP_PROG_ERROR_TRUNC
AP_INVALID_VERB
AP_TP_BUSY

APPC does not return secondary return codes with these primary return codes.

The following primary return code is returned by the MC_RECEIVE_EXPEDITED_DATA verb:

primary_rc

AP_DEALLOC_ABEND

APPC does not return a secondary return code with this primary return code.

The following primary return codes are returned by the RECEIVE_EXPEDITED_DATA verb:

primary_rc

AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_SVC_ERROR_NO_TRUNC
AP_SVC_ERROR_PURGING
AP_SVC_ERROR_TRUNC

APPC does not return secondary return codes with these primary return codes.

State When Issued

The TP can issue the [MC_]RECEIVE_EXPEDITED_DATA verb when the conversation is in any state except Reset.

State Change

The state change after the [MC_]RECEIVE_EXPEDITED_DATA verb depends on the *primary_rc* parameter. The possible state changes are summarized in the following table.

primary_rc	**New state**
AP_OK	No change

primary_rc	**New state**
AP_PARAMETER_CHECK	No change
AP_STATE_CHECK	
AP_CONVERSATION_TYPE_MIXED	
AP_INVALID_VERB	
AP_INVALID_VERB_SEGMENT	
AP_STACK_TOO_SMALL	
AP_TP_BUSY	
AP_UNEXPECTED_DOS_ERROR	
AP_CONV_FAILURE_RETRY	Reset
AP_CONV_FAILURE_NO_RETRY	
AP_CONVERSATION_ENDED	
AP_DEALLOC_ABEND	Reset
AP_DEALLOC_ABEND_PROG	
AP_DEALLOC_ABEND_SVC	
AP_DEALLOC_ABEND_TIMER	
AP_DEALLOC_NORMAL	Reset

MC_REQUEST_TO_SEND and REQUEST_TO_SEND

The MC_REQUEST_TO_SEND or REQUEST_TO_SEND verb notifies the partner TP that the local TP wants to send data.

Note: This verb can be used only in a half-duplex conversation; it is not valid in a full-duplex conversation.

Action of the Partner TP

In response to this request, the partner TP can change the conversation to one of the following states:

- Receive state by issuing the [MC_]PREPARE_TO_RECEIVE or [MC_]RECEIVE_AND_WAIT verb
- Pending_Post state by issuing the [MC_]RECEIVE_AND_POST verb

The partner TP can also ignore the request to send.

When the Local TP Can Send Data

The conversation state changes to Send for the local TP when it receives one of the following values through the *what_rcvd* parameter of a subsequent receive verb:

- AP_CONFIRM_SEND, AP_DATA_CONFIRM_SEND, or AP_DATA_COMPLETE_CONFIRM_SEND (and replies with [MC_]CONFIRMED)
- AP_SEND

The conversation state changes to Send_Pending for the local TP when the local TP receives one of the following values through the *what_rcvd* parameter of a subsequent receive verb:

- AP_DATA_SEND
- AP_DATA_COMPLETE_SEND

The RECEIVE verbs are [MC_]RECEIVE_AND_WAIT, [MC_]RECEIVE_IMMEDIATE, and [MC_]RECEIVE_AND_POST.

VCB Structure: MC_REQUEST_TO_SEND

UNIX

The definition of the VCB structure for the MC_REQUEST_TO_SEND verb is as follows:

```
typedef struct mc_request_to_send
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                 /* Reserved            */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
} MC_REQUEST_TO_SEND;
```

VCB Structure: REQUEST_TO_SEND

The definition of the VCB structure for the REQUEST_TO_SEND verb is as follows:

```
typedef struct request_to_send
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                 /* Reserved            */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
} REQUEST_TO_SEND;
```

VCB Structure: MC_REQUEST_TO_SEND (Windows)

WINDOWS

The definition of the VCB structure for the MC_REQUEST_TO_SEND verb is as follows:

```
typedef struct mc_request_to_send
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
} MC_REQUEST_TO_SEND;
```

VCB Structure: REQUEST_TO_SEND (Windows)

The definition of the VCB structure for the REQUEST_TO_SEND verb is as follows:

```
typedef struct request_to_send
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
} REQUEST_TO_SEND;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_REQUEST_TO_SEND
For the MC_REQUEST_TO_SEND verb.

AP_B_REQUEST_TO_SEND
For the REQUEST_TO_SEND verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_REQUEST_TO_SEND verb.

AP_BASIC_CONVERSATION
For the REQUEST_TO_SEND verb.

If the verb is being issued as a non-blocking verb, combine the value above (using a logical OR) with the value AP_NON_BLOCKING.

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

Conversation Deallocated: If the partner TP has deallocated the conversation, APPC returns the following value:

primary_rc

AP_CONVERSATION_ENDED
This verb was issued as a non-blocking verb and was queued behind an earlier verb. The partner TP issued the [MC_]DEALLOCATE verb as for AP_DEALLOC_NORMAL above, and the first verb in the queue returned with *primary_rc* set to

AP_DEALLOC_NORMAL, indicating the end of the conversation. Any subsequent verbs in the queue then return with *primary_rc* set to AP_CONVERSATION_ENDED, indicating that the conversation had already ended before the verb could be processed.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

UNIX

AP_INVALID_FORMAT
The reserved field *format* was set to a nonzero value.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

AP_R_TO_S_INVALID_FOR_FDX
The local TP attempted to use the [MC_]REQUEST_TO_SEND verb in a full-duplex conversation. This verb can be used only in a half-duplex conversation.

State Check: If the conversation is in the wrong state when the TP issues this verb, APPC returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc

AP_R_T_S_BAD_STATE
The conversation was not in an allowed state when the TP issued this verb.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, “Common Return Codes,” on page 273.

Possible return codes are:

primary_rc
AP_COMM_SUBSYSTEM_ABENDED
AP_CONVERSATION_TYPE_MIXED

```
AP_INVALID_VERB
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR
```

WINDOWS

```
AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT
```

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation can be in any of the following states when the TP issues this verb:

- Receive
- Confirm
- Pending_Post

State Change

The conversation state does not change for this verb.

Receiving Request-to-Send Notification

The request-to-send notification is received by the partner program through the *rts_rcvd* parameter of the following verbs:

- [MC]CONFIRM
- [MC_]RECEIVE_AND_POST
- [MC_]RECEIVE_AND_WAIT
- [MC_]RECEIVE_IMMEDIATE
- [MC_]SEND_DATA
- [MC_]SEND_ERROR

It is also indicated by a *primary_rc* of AP_OK on the [MC_]TEST_RTS verb, or by a callback on the [MC_]TEST_RTS_AND_POST verb.

Request-to-send notification is sent to the partner TP immediately; APPC does not wait until the send buffer fills up or is flushed. Consequently, the request-to-send notification may arrive out of sequence. For example, if the local TP is in Send state and issues the [MC_]PREPARE_TO_RECEIVE verb followed by the [MC_]REQUEST_TO_SEND verb, the partner TP, in Receive state, may receive the request-to-send notification before it receives the send notification. For this reason, request-to-send notification can be reported to a TP on a receive verb.

MC_SEND_CONVERSATION and SEND_CONVERSATION

The MC_SEND_CONVERSATION or SEND_CONVERSATION verb establishes a conversation with the partner TP, sends a single data record on this conversation, and deallocates the conversation. It is equivalent to issuing the three verbs [MC_]ALLOCATE, [MC_]SEND_DATA, [MC_]DEALLOCATE(FLUSH).

VCB Structure: MC_SEND_CONVERSATION

UNIX

The definition of the VCB structure for the MC_SEND_CONVERSATION verb is as follows:

```
typedef struct mc_send_conversation
{
  AP_UINT16       opcode;
  unsigned char   opext;
  unsigned char   format;                  /* Reserved             */
  AP_UINT16       primary_rc;
  AP_UINT32       secondary_rc;
  unsigned char   tp_id[8];
  unsigned char   reserv3[8];
  unsigned char   rtn_ctl;
  unsigned char   reserv4;
  AP_UINT32       conv_group_id;
  AP_UINT32       sense_data;
  unsigned char   plu_alias[8];
  unsigned char   mode_name[8];
  unsigned char   tp_name[64];
  unsigned char   security;
  unsigned char   reserv6[11];
  unsigned char   pwd[10];
  unsigned char   user_id[10];
  AP_UINT16       pip_dlen;
  unsigned char   *pip_dptr;
  unsigned char   reserv6a;
  unsigned char   fqplu_name[17];
  unsigned char   reserv7[8];
  AP_UINT16       dlen;
  unsigned char   *dptr;
} MC_SEND_CONVERSATION;
```

VCB Structure: SEND_CONVERSATION

The definition of the VCB structure for the SEND_CONVERSATION verb is as follows:

```
typedef struct send_conversation
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                  /* Reserved             */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  unsigned char    reserv3[8];
  unsigned char    rtn_ctl;
  unsigned char    reserv4;
  AP_UINT32        conv_group_id;
  AP_UINT32        sense_data;
  unsigned char    plu_alias[8];
  unsigned char    mode_name[8];
  unsigned char    tp_name[64];
  unsigned char    security;
  unsigned char    reserv5[11];
  unsigned char    pwd[10];
  unsigned char    user_id[10];
  AP_UINT16        pip_dlen;
  unsigned char    *pip_dptr;
  unsigned char    reserv5a;
  unsigned char    fqplu_name[17];
```

```
  unsigned char    reserv6[8];
  AP_UINT16        dlen;
  unsigned char    *dptr;
} SEND_CONVERSATION;
```

VCB Structure: MC_SEND_CONVERSATION (Windows)

WINDOWS

The definition of the VCB structure for the MC_SEND_CONVERSATION verb is as follows:

```
typedef struct mc_send_conversation
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned char     reserv3[8];
  unsigned char     rtn_ctl;
  unsigned char     reserv4;
  unsigned long     conv_group_id;
  unsigned long     sense_data;
  unsigned char     plu_alias[8];
  unsigned char     mode_name[8];
  unsigned char     tp_name[64];
  unsigned char     security;
  unsigned char     reserv5[11];
  unsigned char     pwd[10];
  unsigned char     user_id[10];
  unsigned short    pip_dlen;
  unsigned char far *pip_dptr;
  unsigned char     reserv6;
  unsigned char     fqplu_name[17];
  unsigned char     reserv7[8];
  unsigned short    dlen;
  unsigned char far *dptr;
} MC_SEND_CONVERSATION;
```

VCB Structure: SEND_CONVERSATION (Windows)

The definition of the VCB structure for the SEND_CONVERSATION verb is as follows:

```
typedef struct send_conversation
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned char     reserv3[8];
  unsigned char     rtn_ctl;
  unsigned char     reserv4;
  unsigned long     conv_group_id;
  unsigned long     sense_data;
  unsigned char     plu_alias[8];
  unsigned char     mode_name[8];
  unsigned char     tp_name[64];
  unsigned char     security;
  unsigned char     reserv5[11];
  unsigned char     pwd[10];
```

```
  unsigned char     user_id[10];
  unsigned short    pip_dlen;
  unsigned char far *pip_dptr;
  unsigned char     reserv6;
  unsigned char     fqplu_name[17];
  unsigned char     reserv7[8];
  unsigned short    dlen;
  unsigned char far *dptr;
} SEND_CONVERSATION;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_SEND_CONVERSATION
For the MC_SEND_CONVERSATION verb.

AP_B_SEND_CONVERSATION
For the SEND_CONVERSATION verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_SEND_CONVERSATION verb.

AP_BASIC_CONVERSATION
For the SEND_CONVERSATION verb.

If the verb is being issued as a non-blocking verb, combine the value above (using a logical OR) with the value AP_NON_BLOCKING.

tp_id Identifier for the local TP.

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

rtn_ctl Specifies when the local LU acting on a session request from the local TP is to return control to the local TP. For information about sessions, see "LU-to-LU Sessions" on page 57. Whatever the value of this parameter, the LU returns control to the TP immediately if it encounters certain errors such as a zero session limit (which mean that a session will never be allocated).

Possible values are:

AP_IMMEDIATE
If a contention-winner session is immediately available (active and not being used by another conversation), the LU allocates this conversation to it and returns control to the TP immediately. If a contention-winner session is not immediately available, control is returned to the TP immediately with a *primary_rc* of AP_UNSUCCESSFUL.

AP_WHEN_SESSION_ALLOCATED
If a session is immediately available (active and not being used by another conversation), the LU allocates this conversation to it. If a session is not immediately available but one can be activated, the LU activates it and allocates the conversation to it; if it cannot activate a session, it waits for one to become free.

AP_WHEN_SESSION_FREE
If a session is immediately available (active and not being used by another conversation), the LU allocates this conversation to it. If a session is not immediately available but one can be activated, the LU activates it and allocates the conversation to it. If no active session is free and another session cannot be activated, control is returned to the TP with the primary return code AP_ALLOCATION_ERROR and secondary return code AP_ALLOCATION_FAILURE_RETRY. This is similar to AP_WHEN_SESSION_ALLOCATED except that the LU will not wait for a session to become free.

AP_WHEN_CONWINNER_ALLOC
As for AP_WHEN_SESSION_ALLOCATED, except that the LU always allocates the conversation to a contention-winner session; it will not use a contention-loser session.

AP_WHEN_CONLOSER_ALLOC
As for AP_WHEN_SESSION_ALLOCATED, except that the LU always allocates the conversation to a contention-loser session; it will not use a contention-winner session.

AP_WHEN_CONV_GROUP_ALLOC
Use this value if you want the new conversation to use the same session as a previous conversation; set the *conv_group_id* parameter to the conversation group ID of the previous conversation, which was returned on the [MC_]ALLOCATE or RECEIVE_ALLOCATE verb.

If the session identified by the *conv_group_id* parameter is immediately available (active and not being used by another conversation), the LU allocates this conversation to it and returns control to the TP immediately. If the session is being used by another conversation, the LU waits for it to become free. If the session is no longer active, control is returned to the TP with the primary return code AP_ALLOCATION_ERROR and secondary return code AP_ALLOCATION_FAILURE_NO_RETRY

conv_group_id
Conversation group ID of the requested session for the conversation. This parameter is used only if *rtn_ctl* is set to AP_WHEN_CONV_GROUP_ALLOC; set it to binary zeros for any other value of *rtn_ctl*.

plu_alias
Alias by which the partner LU is known to the local TP. This name must match the name of a partner LU established during configuration.

This parameter is an 8-byte ASCII character string, padded on the right with ASCII blanks (0x20) if the alias is shorter than eight characters. It can consist of any of the following characters:

- Uppercase letters
- Numerals 0–9
- Blanks
- Special characters $, #, %, and @

The first character of this string cannot be a blank.

To identify the LU by its LU name instead of its LU alias, set this parameter to 8 binary zeros, and specify the LU name in the *fqplu_name* parameter.

mode_name
Name of a set of networking characteristics defined during configuration.

The value of *mode_name* must match the name of a mode associated with the partner LU during configuration.

This parameter is an 8-byte EBCDIC character string. It can consist of characters from the type-A EBCDIC character set. These characters are as follows:

- Uppercase letters
- Numerals 0–9
- Special characters $, #, and @

The first character in the string must be an uppercase letter or special character. If the mode name is fewer than eight characters long, pad it on the right with EBCDIC blanks (`0x40`).

A mode name can also be all EBCDIC blanks (`0x40`).

In a mapped conversation, the name cannot be SNASVCMG (a reserved mode name used internally by APPC). Using this name in a basic conversation is not recommended.

tp_name
Name of the invoked TP.

The value of *tp_name* specified by the [MC_]ALLOCATE verb in the invoking TP must match the value of *tp_name* specified by the RECEIVE_ALLOCATE verb in the invoked TP.

This parameter is a 64-byte EBCDIC character string; it is case-sensitive. The *tp_name* parameter normally consists of characters from the type-AE EBCDIC character set (except when naming a service TP). These characters are as follows:

- Uppercase and lowercase letters
- Numerals 0–9
- Special characters $, #, @, and period (.)

If the TP name is fewer than 64 bytes, use EBCDIC blanks (`0x40`) to pad it on the right.

The SNA convention for naming a service TP is an exception to the above; the name consists of up to four characters, of which the first character is a hexadecimal byte between `0x00` and `0x3F`. The other characters are from the EBCDIC AE character set.

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

Based on the conversation security established for the invoked TP during configuration, use one of the following values:

AP_NONE
The invoked TP does not use conversation security. (If you use this value, the invoked TP must be configured not to use conversation security.)

AP_PGM The invoked TP uses conversation security and thus requires a user ID and password. Supply this information through the *user_id* and *pwd* parameters.

AP_PGM_STRONG
The invoked TP uses conversation security and thus requires a user ID and password. In addition, setting AP_PGM_STRONG stipulates that Communications Server encrypts the password when sending it across the network. Supply the user ID and password through the *user_id* and *pwd* parameters.

AP_SAME
Use this value when your TP was invoked by another TP, using a valid user ID and password, and is now invoking a third TP which also requires conversation security. (The situation in which one TP invokes a second TP which then invokes a third TP is illustrated in "Multiple Conversations" on page 3). This value tells the third TP (the invoked TP) that conversation security has already been verified for the first invoking TP.

If you use this value, the *tp_id* supplied on this [MC_]SEND_CONVERSATION verb must be the same as the one that was returned on the RECEIVE_ALLOCATE verb when this TP was invoked.

UNIX

This value may also be used if your TP was not invoked by another TP, but has obtained and verified the appropriate security information by another means (for example from the AIX / Linux user name and password supplied during logon). In this case, APPC uses the AIX / Linux user name with which the application is running, truncated to 10 characters if necessary, as the user ID for conversation security; ensure that this name consists of valid AE-string characters (see the description of the *user_id* parameter) and is a valid user name for the TP being invoked.

If the TP has obtained the security information by another means (for example by requesting the user to type in a valid user ID and password before allocating the conversation), it should use SET_TP_PROPERTIES to specify this user ID to APPC before issuing [MC_]SEND_CONVERSATION.

pwd Password associated with *user_id*.

This parameter is required only if the *security* parameter is set to AP_PGM or AP_PGM_STRONG; otherwise it is reserved.

The *pwd* and *user_id* parameters must match a user ID/password pair configured on the computer where the invoked TP is located.

This parameter is a 10-byte EBCDIC character string; it is case-sensitive. The *pwd* parameter can consist of characters from the type-AE EBCDIC character set. These characters are as follows:

- Uppercase and lowercase letters
- Numerals 0–9

- Special characters $, #, @, and period (.)

If the password is fewer than 10 bytes, use EBCDIC blanks (0x40) to pad it on the right.

user_id User ID required to access the partner TP.

This parameter is required only if the *security* parameter is set to AP_PGM or AP_PGM_STRONG; otherwise it is reserved.

The *pwd* and *user_id* parameters must match a user ID/password pair configured on the computer where the invoked TP is located.

This parameter is a 10-byte EBCDIC character string; it is case-sensitive. The *user_id* parameter can consist of characters from the type-AE EBCDIC character set. These characters are as follows:

- Uppercase and lowercase letters
- Numerals 0–9
- Special characters $, #, @, and period (.)

If the user ID is fewer than 10 bytes, use EBCDIC blanks (0x40) to pad it on the right.

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

The range for this value is 0–32,767.

Not all APPC implementations can receive PIP data (although they may be able to send it); in addition, CPI-C does not support PIP data. Set *pip_dlen* to 0 (zero) if the partner TP is using an implementation of APPC that does not support PIP data, or if the partner is a CPI-C application.

pip_dptr
Address of buffer containing PIP data.

Use this parameter only if *pip_dlen* is greater than 0 (zero).

PIP data can consist of initialization parameters or environment setup information required by a partner TP or remote operating system. The PIP data must follow the General Data Stream format. For further information, refer to the IBM publication *Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2.*

fqplu_name
Fully qualified LU name of the partner LU. This parameter is used only if *plu_alias* is set to zeros. The name must match the name of a partner LU established during configuration.

This name is a 17-byte EBCDIC string, padded on the right with EBCDIC spaces, containing one of the following:

- A network ID of 1–8 A-string characters, an EBCDIC dot (period) character, and an LU name of 1–8 A-string characters
- An LU name of 1–8 A-string characters (without the network ID or the EBCDIC dot)

dlen Number of bytes of data to be sent. The range for this value is 0–65,535.

dptr Address of the buffer containing the data to be sent.

WINDOWS

The data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

conv_group_id
The conversation group identifier of the session used by the conversation.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_LL
This return code applies only to the SEND_CONVERSATION verb. The logical record length field of a logical record contained a value that was not valid—0x0000, 0x0001, 0x8000, or 0x8001. For more information, see "Logical Records" on page 58.

AP_BAD_RETURN_CONTROL
The value specified for *rtn_ctl* was not valid.

AP_BAD_SECURITY
The value specified for security was not valid.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

AP_PIP_LEN_INCORRECT
The value of *pip_dlen* was greater than 32,767.

AP_UNKNOWN_PARTNER_MODE
The value specified for *plu_alias* or *mode_name* was not valid.

UNIX

AP_INVALID_FORMAT
The reserved field *format* was set to a nonzero value.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

State Check: No state check errors occur for this verb.

Session Not Available: Depending on the value specified for *rtn_ctl*, APPC may return the following parameter:

primary_rc

AP_UNSUCCESSFUL
The supplied parameter *rtn_ctl* specified immediate (AP_IMMEDIATE) return of control to the TP, and the local LU did not have an available contention-winner session.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc
AP_ALLOCATION_ERROR

secondary_rc
AP_ALLOCATION_FAILURE_NO_RETRY
AP_ALLOCATION_FAILURE_RETRY
AP_CONVERSATION_TYPE_MISMATCH
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_SECURITY_NOT_VALID
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_TP_NAME_NOT_RECOGNIZED
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_SEC_REQUESTED_NOT_SUPPORTED
AP_SEC_BAD_PROTOCOL_VIOLATION
AP_SEC_BAD_PASSWORD_EXPIRED
AP_SEC_BAD_PASSWORD_INVALID
AP_SEC_BAD_USERID_REVOKED
AP_SEC_BAD_USERID_INVALID
AP_SEC_BAD_USERID_MISSING
AP_SEC_BAD_PASSWORD_MISSING
AP_SEC_BAD_UID_NOT_DEFD_TO_GRP
AP_SEC_BAD_UNAUTHRZD_AT_RLU
AP_SEC_BAD_UNAUTHRZD_FROM_LLU
AP_SEC_BAD_UNAUTHRZD_TO_TP
AP_SEC_BAD_INSTALL_EXIT_FAILED
AP_SEC_BAD_PROCESSING_FAILURE

primary_rc
AP_COMM_SUBSYSTEM_ABENDED
AP_CONV_FAILURE_NO_RETRY

```
AP_CONV_FAILURE_RETRY
AP_INVALID_VERB
AP_UNEXPECTED_SYSTEM_ERROR
```

WINDOWS

```
AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT
```

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation state is Reset when the TP issues this verb. It can be issued during an existing conversation that is in any state, because it always implies the start of a new conversation that is in Reset state.

State Change

The conversation state does not change for this verb.

MC_SEND_DATA and SEND_DATA

The MC_SEND_DATA or SEND_DATA verb puts data in the local LU's send buffer for transmission to the partner TP.

The data collected in the local LU's send buffer is transmitted to the partner LU (and partner TP) when one of the following occurs:

- The send buffer fills up.
- The local TP issues a verb that flushes the LU's send buffer. The verbs that do this are [MC_]CONFIRM, [MC_]DEALLOCATE, [MC_]FLUSH, [MC_]PREPARE_TO_RECEIVE, [MC_]RECEIVE_AND_WAIT, [MC_]RECEIVE_AND_POST, and [MC_]SEND_ERROR. ([MC_]CONFIRM, [MC_]PREPARE_TO_RECEIVE, and [MC_]RECEIVE_AND_POST apply only to half-duplex conversations.)

The MC_SEND_DATA or SEND_DATA verb also includes options that enable it to perform the function of the [MC_]CONFIRM, [MC_]DEALLOCATE, [MC_]FLUSH, or [MC_]PREPARE_TO_RECEIVE verb in addition to sending the data. This is equivalent to issuing [MC_]SEND_DATA followed by another verb. ([MC_]CONFIRM and [MC_]PREPARE_TO_RECEIVE apply only to half-duplex conversations.)

VCB Structure: MC_SEND_DATA

UNIX

The definition of the VCB structure for the MC_SEND_DATA verb is as follows:

```
typedef struct mc_send_data
{
  AP_UINT16        opcode;
  unsigned char    opext;
```

```
  unsigned char    format;
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  unsigned char    rts_rcvd;
  unsigned char    expd_rcvd;
  AP_UINT16        dlen;
  unsigned char    *dptr;
  unsigned char    type;
  unsigned char    data_type;
} MC_SEND_DATA;
```

VCB Structure: SEND_DATA

The definition of the VCB structure for the SEND_DATA verb is as follows:

```
typedef struct send_data
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                 /* Reserved             */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  unsigned char    rts_rcvd;
  unsigned char    expd_rcvd;
  AP_UINT16        dlen;
  unsigned char    *dptr;
  unsigned char    type;
  unsigned char    reserv4;
} SEND_DATA;
```

VCB Structure: MC_SEND_DATA (Windows)

WINDOWS

The definition of the VCB structure for the MC_SEND_DATA verb is as follows:

```
typedef struct mc_send_data
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     rts_rcvd;
  unsigned char     reserv3;
  unsigned short    dlen;
  unsigned char far *dptr;
  unsigned char     type;
  unsigned char     reserv4;
} MC_SEND_DATA;
```

VCB Structure: SEND_DATA (Windows)

The definition of the VCB structure for the SEND_DATA verb is as follows:

```
typedef struct send_data
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
```

```
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     rts_rcvd;
  unsigned char     reserv3;
  unsigned short    dlen;
  unsigned char far *dptr;
  unsigned char     type;
  unsigned char     reserv4;
} SEND_DATA;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_SEND_DATA
For the MC_SEND_DATA verb.

AP_B_SEND_DATA
For the SEND_DATA verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_SEND_DATA verb.

AP_BASIC_CONVERSATION
For the SEND_DATA verb.

If the verb is being issued on a full-duplex conversation or is being issued as a non-blocking verb, combine the value above (using a logical OR) with one or both of the following values:

AP_FULL_DUPLEX_CONVERSATION
The verb is being issued on a full-duplex conversation.

AP_NON_BLOCKING
The verb is being issued as a non-blocking verb.

format This parameter applies only to the mapped-conversation MC_SEND_DATA verb.

If you are building a new APPC application, or recompiling an existing APPC application with the current Communications Server APPC header file, you must set this parameter to 1. (Existing applications built with earlier versions of the header file, in which this parameter was reserved, will still operate unchanged with Communications Server and there is no need to rebuild them.)

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

dlen Number of bytes of data to be put in the local LU's send buffer. The range for this value is 0–65,535.

dptr Address of the buffer containing the data to be put in the local LU's send buffer.

WINDOWS

The data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

type Specifies whether to perform the function of another APPC verb in addition to [MC_]SEND_DATA. Possible values are:

AP_NONE
Send data only—do not perform any additional function.

AP_SEND_DATA_CONFIRM
This option is valid only in a half-duplex conversation. Do not use it if the *opext* parameter includes the option AP_FULL_DUPLEX_CONVERSATION.

Perform the function of the [MC_]CONFIRM verb. This is equivalent to issuing [MC_]SEND_DATA followed by [MC_]CONFIRM. For the basic-conversation SEND_DATA verb, the data sent on this verb must be a complete logical record or the end of a logical record; this value cannot be used if an incomplete logical record is being sent.

AP_SEND_DATA_FLUSH
Perform the function of the [MC_]FLUSH verb. This is equivalent to issuing [MC_]SEND_DATA followed by [MC_]FLUSH. For the basic-conversation SEND_DATA verb, the data sent on this verb must be a complete logical record or the end of a logical record; this value cannot be used if an incomplete logical record is being sent.

AP_SEND_DATA_P_TO_R_FLUSH
This option is valid only in a half-duplex conversation. Do not use it if the *opext* parameter includes the option AP_FULL_DUPLEX_CONVERSATION.

Perform the function of the [MC_]PREPARE_TO_RECEIVE verb with *ptr_type* set to AP_FLUSH. This is equivalent to issuing [MC_]SEND_DATA followed by [MC_]PREPARE_TO_RECEIVE. For the basic-conversation SEND_DATA verb, the data sent on this verb must be a complete logical record or the end of a logical record; this value cannot be used if an incomplete logical record is being sent.

AP_SEND_DATA_P_TO_R_CONFIRM
This option is valid only in a half-duplex conversation. Do not use it if the *opext* parameter includes the option AP_FULL_DUPLEX_CONVERSATION.

Perform the function of the [MC_]PREPARE_TO_RECEIVE verb with *ptr_type* set to AP_CONFIRM_TYPE. This is equivalent to issuing [MC_]SEND_DATA followed by [MC_]PREPARE_TO_RECEIVE.

For the basic-conversation SEND_DATA verb, the data sent on this verb must be a complete logical record or the end of a logical record; this value cannot be used if an incomplete logical record is being sent.

AP_SEND_DATA_P_TO_R_SYNC_LEVEL
This option is valid only in a half-duplex conversation. Do not use it if the *opext* parameter includes the option AP_FULL_DUPLEX_CONVERSATION.

Perform the function of the [MC_]PREPARE_TO_RECEIVE verb with *ptr_type* set to AP_SYNC_LEVEL. This is equivalent to issuing [MC_]SEND_DATA followed by [MC_]PREPARE_TO_RECEIVE. For the basic-conversation SEND_DATA verb, the data sent on this verb must be a complete logical record or the end of a logical record; this value cannot be used if an incomplete logical record is being sent.

AP_SEND_DATA_DEALLOC_FLUSH
Perform the function of the [MC_]DEALLOCATE verb with *dealloc_type* set to AP_FLUSH. This is equivalent to issuing [MC_]SEND_DATA followed by [MC_]DEALLOCATE. For the basic-conversation SEND_DATA verb, the data sent on this verb must be a complete logical record or the end of a logical record; this value cannot be used if an incomplete logical record is being sent.

AP_SEND_DATA_DEALLOC_CONFIRM
This option is valid only in a half-duplex conversation. Do not use it if the *opext* parameter includes the option AP_FULL_DUPLEX_CONVERSATION.

Perform the function of the [MC_]DEALLOCATE verb with *dealloc_type* set to AP_CONFIRM_TYPE. This is equivalent to issuing [MC_]SEND_DATA followed by [MC_]DEALLOCATE. For the basic-conversation SEND_DATA verb, the data sent on this verb must be a complete logical record or the end of a logical record; this value cannot be used if an incomplete logical record is being sent.

AP_SEND_DATA_DEALLOC_SYNC_LEVEL
Perform the function of the [MC_]DEALLOCATE verb with *dealloc_type* set to AP_SYNC_LEVEL. This is equivalent to issuing [MC_]SEND_DATA followed by [MC_]DEALLOCATE. For the basic-conversation SEND_DATA verb, the data sent on this verb must be a complete logical record or the end of a logical record; this value cannot be used if an incomplete logical record is being sent.

AP_SEND_DATA_DEALLOC_ABEND
Perform the function of the MC_DEALLOCATE verb with *dealloc_type* set to AP_ABEND, or the DEALLOCATE verb with *dealloc_type* set to AP_ABEND_PROG. This is equivalent to issuing [MC_]SEND_DATA followed by [MC_]DEALLOCATE.

You cannot use [MC_]SEND_DATA to perform the function of [MC_]DEALLOCATE in the following cases:

- The conversation type is AP_BASIC_CONVERSATION, and the required *dealloc_type* is AP_ABEND_SVC or AP_ABEND_TIMER

- The conversation's synchronization level is AP_SYNCPT, and the TP requires implied forget notification

In these cases, you need to issue [MC_]SEND_DATA and [MC_]DEALLOCATE separately. See the description of the [MC_]DEALLOCATE verb in Chapter 4, "APPC Conversation Verbs," on page 91 for more information.

UNIX

data_type
Specifies the format of the data being sent. This parameter is used only by the mapped-conversation MC_SEND_DATA verb. Possible values are:

AP_APPLICATION
Standard APPC application data. Communications Server sends the data to the partner LU in Application Data GDS variables.

AP_USER_CONTROL_DATA
User Control data. Communications Server sends the data to the partner LU in User Control Data GDS variables. Do not set this option unless the partner LU can accept data in this format.

AP_PS_HEADER
PS Header data. This data format is used only by Syncpoint TPs; do not set it unless the synchronization level of the conversation is AP_SYNCPT. The Syncpoint Manager is responsible for converting Syncpoint commands into the appropriate PS Headers.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

rts_rcvd
Request-to-send-received indicator. This parameter applies only in a half-duplex conversation; it is not used in a full-duplex conversation.

Possible values are:

AP_YES The partner TP has issued an [MC_]REQUEST_TO_SEND verb, which requests that the local TP change the conversation to Receive state. To change to Receive state, the local TP can use the [MC_]PREPARE_TO_RECEIVE, [MC_]RECEIVE_AND_WAIT, or [MC_]RECEIVE_AND_POST verb.

AP_NO The partner TP has not issued the [MC_]REQUEST_TO_SEND verb.

expd_rcvd
Expedited data indicator.

Possible values are:

AP_YES The partner TP has sent expedited data that the local TP has not yet received. To receive this data, the local TP can use the [MC_]RECEIVE_EXPEDITED_DATA verb.

This indicator can be set on a number of APPC verbs. It continues to be set on subsequent verbs until the local TP issues the [MC_]RECEIVE_EXPEDITED_DATA verb to receive the data.

AP_NO There is no expedited data waiting to be received.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_LL
This return code applies only to the SEND_DATA verb. The logical record length field of a logical record contained a value that was not valid—0x0000, 0x0001, 0x8000, or 0x8001. For more information, see "Logical Records" on page 58.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

WINDOWS

AP_INVALID_DATA_SEGMENT
The data was longer than the allocated data segment, or the address of the data buffer was incorrect.

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

UNIX

AP_INVALID_FORMAT
The *format* parameter was set to a value that was not valid.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

AP_SEND_TYPE_INVALID_FOR_FDX
The application issued this verb in a full-duplex conversation, but the *type* parameter specified a send type that was not valid in a full-duplex conversation.

State Check: If the conversation is in the wrong state when the TP issues this verb, APPC returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
Possible values are:

AP_SEND_DATA_NOT_SEND_STATE
The local TP issued the [MC_]SEND_DATA verb, but the conversation was not in Send or Send_Pending state.

AP_SEND_DATA_CONFIRM_SYNC_NONE
The local TP issued the [MC_]SEND_DATA verb with the *type* parameter set to AP_SEND_DATA_CONFIRM, but the synchronization level of the conversation was AP_NONE. The CONFIRM function is only valid if the synchronization level is AP_CONFIRM_SYNC_LEVEL.

AP_SEND_DATA_NOT_LL_BDY
(Returned for basic-conversation SEND_DATA verb only) The local TP issued the SEND_DATA verb to send an incomplete logical record, and used a *type* parameter other than AP_NONE or AP_SEND_DATA_DEALLOC_ABEND.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc
AP_ALLOCATION_ERROR

secondary_rc
AP_ALLOCATION_FAILURE_NO_RETRY
AP_ALLOCATION_FAILURE_RETRY
AP_CONVERSATION_TYPE_MISMATCH
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_SECURITY_NOT_VALID
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_TP_NAME_NOT_RECOGNIZED
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_SEC_BAD_PROTOCOL_VIOLATION
AP_SEC_BAD_PASSWORD_EXPIRED
AP_SEC_BAD_PASSWORD_INVALID
AP_SEC_BAD_USERID_REVOKED
AP_SEC_BAD_USERID_INVALID
AP_SEC_BAD_USERID_MISSING

AP_SEC_BAD_PASSWORD_MISSING
AP_SEC_BAD_UID_NOT_DEFD_TO_GRP
AP_SEC_BAD_UNAUTHRZD_AT_RLU
AP_SEC_BAD_UNAUTHRZD_FROM_LLU
AP_SEC_BAD_UNAUTHRZD_TO_TP
AP_SEC_BAD_INSTALL_EXIT_FAILED
AP_SEC_BAD_PROCESSING_FAILURE

UNIX

primary_rc
AP_BACKED_OUT

secondary_rc
AP_BO_NO_RESYNC
AP_BO_RESYNC

primary_rc
AP_COMM_SUBSYSTEM_ABENDED
AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_CONVERSATION_TYPE_MIXED
AP_DUPLEX_TYPE_MIXED
AP_PROG_ERROR_PURGING
AP_INVALID_VERB
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT

APPC does not return secondary return codes with these primary return codes.

The following primary return code is returned by the MC_SEND_DATA verb:

primary_rc
AP_DEALLOC_ABEND

APPC does not return a secondary return code with this primary return code.

The following primary return codes are returned by the SEND_DATA verb:

primary_rc
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_SVC_ERROR_PURGING

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation must be in Send_Receive (full-duplex conversation only), Send or Send_Pending state when the TP issues this verb.

State Change

State changes, summarized in the following table, are based on the *primary_rc* parameter.

primary_rc	**New state**
AP_OK	Send (half-duplex conversation) or no change (full-duplex conversation)
AP_STATE_CHECK AP_PARAMETER_CHECK AP_CONVERSATION_TYPE_MIXED AP_INVALID_VERB AP_INVALID_VERB_SEGMENT AP_STACK_TOO_SMALL AP_TP_BUSY AP_UNEXPECTED_DOS_ERROR	No change
AP_ALLOCATION_ERROR	Reset
AP_CONV_FAILURE_RETRY AP_CONV_FAILURE_NO_RETRY	Reset
AP_DEALLOC_ABEND AP_DEALLOC_ABEND_PROG AP_DEALLOC_ABEND_SVC AP_DEALLOC_ABEND_TIMER	Reset
AP_PROG_ERROR_PURGING AP_SVC_ERROR_PURGING	Receive (half-duplex conversation), Send_Receive (full-duplex conversation, verb issued in Send_Receive state), or Reset (full-duplex conversation, verb issued in Send_Only state)

Waiting for Partner TP

The [MC_]SEND_DATA verb may wait indefinitely because the partner TP has not issued a receive verb. This is because the send buffer may fill up and APPC cannot transmit its contents to the partner LU because the partner LU has no buffers to receive the data.

Logical Records in Basic Conversations

When using the basic-conversation SEND_DATA verb, the application must supply data in the form of logical records (with an LLID field at the start of each data record). For more information, see "Logical Records" on page 58.

UNIX

In a conversation with a synchronization level of AP_SYNCPT, the data to be sent may be in PS Header format; this is indicated by a length field of 0x0001. The Syncpoint Manager is responsible for setting up the appropriate PS headers based on the Syncpoint functions required by the application.

MC_SEND_ERROR and SEND_ERROR

The MC_SEND_ERROR or SEND_ERROR verb notifies the partner TP that the local TP has encountered an application-level error.

The local TP sends the error notification immediately to the partner TP; it does not hold the information in the local LU's send buffer.

For a half-duplex conversation, after successful execution of this verb, the conversation is in Send state for the local TP and in Receive state for the partner TP. For a full-duplex conversation, there is no state change after successful execution of this verb.

VCB Structure: MC_SEND_ERROR

UNIX

The definition of the VCB structure for the MC_SEND_ERROR verb is as follows:

```
typedef struct mc_send_error
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                 /* Reserved             */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  unsigned char    rts_rcvd;
  unsigned char    err_type;
  unsigned char    err_dir;
  unsigned char    expd_rcvd;
  unsigned char    reserv5[2];
  unsigned char    reserv6[4];
} MC_SEND_ERROR;
```

VCB Structure: SEND_ERROR

The definition of the VCB structure for the SEND_ERROR verb is as follows:

```
typedef struct send_error
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                 /* Reserved             */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  unsigned char    rts_rcvd;
  unsigned char    err_type;
  unsigned char    err_dir;
  unsigned char    expd_rcvd;
  AP_UINT16        log_dlen;
  unsigned char    *log_dptr;
} SEND_ERROR;
```

VCB Structure: MC_SEND_ERROR (Windows)

WINDOWS

The definition of the VCB structure for the MC_SEND_ERROR verb is as follows:

```
typedef struct mc_send_error
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     rts_rcvd;
  unsigned char     reserv3;
  unsigned char     err_dir;
  unsigned char     reserv4;
  unsigned char     reserv5[2];
  unsigned char     reserv6[4];
} MC_SEND_ERROR;
```

VCB Structure: SEND_ERROR (Windows)

The definition of the VCB structure for the SEND_ERROR verb is as follows:

```
typedef struct send_error
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     rts_rcvd;
  unsigned char     err_type;
  unsigned char     err_dir;
  unsigned char     reserv3;
  unsigned short    log_dlen;
  unsigned char far *log_dptr;
} SEND_ERROR;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_SEND_ERROR
For the MC_SEND_ERROR verb.

AP_B_SEND_ERROR
For the SEND_ERROR verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_SEND_ERROR verb.

AP_BASIC_CONVERSATION
For the SEND_ERROR verb.

If the verb is being issued on a full-duplex conversation or is being issued as a non-blocking verb, combine the value above (using a logical OR) with one or both of the following values:

AP_FULL_DUPLEX_CONVERSATION
The verb is being issued on a full-duplex conversation.

AP_NON_BLOCKING
The verb is being issued as a non-blocking verb.

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

err_type
Indicates the type of error being reported. This determines the return code that APPC sends to the partner TP to report the error; all of these return codes are described in Appendix B, "Common Return Codes," on page 273.

WINDOWS

This parameter is used only by the basic-conversation SEND_ERROR verb.

Possible values are:

AP_PROG
The error is to be reported to an application program that does not use Syncpoint. This value causes APPC to send one of the following return codes to the partner TP:

- AP_PROG_ERROR_TRUNC (if the SEND_ERROR verb is issued in Send state after sending part of a logical record)
- AP_PROG_ERROR_NO_TRUNC (if the MC_SEND_ERROR verb is issued in Send state, or if the SEND_ERROR verb is issued in Send state but an incomplete logical record has not been sent)
- AP_PROG_ERROR_PURGING (if the verb is issued in any state other than Send)

AP_SVC The error is to be reported to a service program. This value is used only by the SEND_ERROR verb. This value causes APPC to send one of the following return codes to the partner TP:

- AP_SVC_ERROR_TRUNC (if the SEND_ERROR verb is issued in Send state after sending part of a logical record)
- AP_SVC_ERROR_NO_TRUNC (if the SEND_ERROR verb is issued in Send state but an incomplete logical record has not been sent)
- AP_SVC_ERROR_PURGING (if the SEND_ERROR verb is issued in any state other than Send)

UNIX

AP_BACKOUT_NO_RESYNC
This value is allowed only if the conversation's synchronization level is AP_SYNCPT. The local TP (or another TP participating in the same logical unit of work) has issued a BACKOUT request; the local TP has completed backing out its resources. The Syncpoint Manager is responsible for issuing [MC_]SEND_ERROR with this value set when it receives the BACKOUT request. This value causes APPC to send the primary and secondary return codes AP_BACKED_OUT and AP_BO_NO_RESYNC to the partner TP.

AP_BACKOUT_RESYNC
This value is allowed only if the conversation's synchronization level is AP_SYNCPT. The local TP (or another TP participating in the same logical unit of work) has issued a backout request; resynchronization is still in progress. The Syncpoint Manager is responsible for issuing [MC_]SEND_ERROR with this value set when it receives the BACKOUT request. This value causes APPC to send the primary and secondary return codes AP_BACKED_OUT and AP_BO_RESYNC to the partner TP.

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

In a full-duplex conversation, this parameter must be set to AP_SEND_DIR_ERROR. In a half-duplex conversation, this parameter is used only when the [MC_]SEND_ERROR verb is being issued in Send_Pending state.

Possible values are:

AP_RCV_DIR_ERROR
The local TP detected an error in the data it received from the remote TP.

AP_SEND_DIR_ERROR
The local TP detected an error in its own data (for example, it could not read data from disk) or in its own processing.

log_dlen
Number of bytes of data to be sent to the error log file. This parameter is used only by the SEND_ERROR verb.

The range for this value is 0–32,767. A length of 0 (zero) indicates that there is no error log data.

log_dptr
Address of data buffer containing error information. 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 0 (zero).

The TP must format the error data as a General Data Stream (GDS) error log variable. For further information, refer to the IBM publication *Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2.*

WINDOWS

The data buffer can reside in a static data area or in a globally allocated area. The data buffer must fit entirely within this area.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

rts_rcvd
Request-to-send-received indicator. This parameter applies only in a half-duplex conversation; it is not used in a full-duplex conversation. Possible values are:

AP_YES The partner TP has issued the [MC_]REQUEST_TO_SEND verb, which requests that the local TP change the conversation to Receive state. To change to Receive state, the local TP can use the [MC_]PREPARE_TO_RECEIVE, [MC_]RECEIVE_AND_WAIT, or [MC_]RECEIVE_AND_POST verb.

AP_NO The partner TP has not issued the [MC_]REQUEST_TO_SEND verb.

expd_rcvd
Expedited data indicator.

Possible values are:

AP_YES The partner TP has sent expedited data that the local TP has not yet received. To receive this data, the local TP can use the [MC_]RECEIVE_EXPEDITED_DATA verb.

This indicator can be set on a number of APPC verbs. It continues to be set on subsequent verbs until the local TP issues the [MC_]RECEIVE_EXPEDITED_DATA verb to receive the data.

AP_NO There is no expedited data waiting to be received.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_ERROR_DIRECTION
The value of *err_dir* was not valid.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

AP_SEND_ERROR_BAD_TYPE
The value of *err_type* was not valid.

UNIX

AP_INVALID_FORMAT
The reserved field *format* was set to a nonzero value.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

WINDOWS

AP_INVALID_DATA_SEGMENT
The log data was longer than the allocated data segment, or the address of the log data buffer was incorrect.

The following *secondary_rc* value can be returned only on the SEND_ERROR verb:

AP_SEND_ERROR_LOG_LL_WRONG
The LL field of the error log GDS variable did not match the actual length of the data.

State Check: No state check errors occur for this verb.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Verb Issued in Any Allowed State: The following return codes can be generated when the [MC_]SEND_ERROR verb is issued in any allowed state:

primary_rc

```
AP_COMM_SUBSYSTEM_ABENDED
AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_CONVERSATION_TYPE_MIXED
AP_DUPLEX_TYPE_MIXED
AP_INVALID_VERB
AP_TP_BUSY
```

AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT

APPC does not return secondary return codes with these primary return codes.

Verb Issued in Send State: The following return codes can be generated only if the [MC_]SEND_ERROR verb is issued in Send state:

primary_rc

AP_ALLOCATION_ERROR

secondary_rc

AP_ALLOCATION_FAILURE_NO_RETRY
AP_ALLOCATION_FAILURE_RETRY
AP_CONVERSATION_TYPE_MISMATCH
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_SECURITY_NOT_VALID
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_TP_NAME_NOT_RECOGNIZED
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
AP_TRANS_PGM_NOT_AVAIL_RETRY
AP_SEC_BAD_PROTOCOL_VIOLATION
AP_SEC_BAD_PASSWORD_EXPIRED
AP_SEC_BAD_PASSWORD_INVALID
AP_SEC_BAD_USERID_REVOKED
AP_SEC_BAD_USERID_INVALID
AP_SEC_BAD_USERID_MISSING
AP_SEC_BAD_PASSWORD_MISSING
AP_SEC_BAD_UID_NOT_DEFD_TO_GRP
AP_SEC_BAD_UNAUTHRZD_AT_RLU
AP_SEC_BAD_UNAUTHRZD_FROM_LLU
AP_SEC_BAD_UNAUTHRZD_TO_TP
AP_SEC_BAD_INSTALL_EXIT_FAILED
AP_SEC_BAD_PROCESSING_FAILURE

UNIX

primary_rc

AP_BACKED_OUT

secondary_rc

AP_BO_NO_RESYNC
AP_BO_RESYNC

primary_rc
AP_PROG_ERROR_PURGING

APPC does not return a secondary return code with this primary return code.

The following return code can be generated only if the MC_SEND_ERROR verb is issued in Send state:

primary_rc
AP_DEALLOC_ABEND

APPC does not return a secondary return code with this primary return code.

The following return codes can be generated only if the SEND_ERROR verb is issued in Send state:

primary_rc
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_SVC_ERROR_PURGING

APPC does not return secondary return codes with these primary return codes.

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

primary_rc
AP_DEALLOC_NORMAL

APPC does not return a secondary return code with this primary return code.

State When Issued

The conversation can be in any state except Reset when the TP issues this verb.

State Change

The new state is determined by the primary return code *primary_rc*. Possible state changes are summarized in the following table.

primary_rc	**New state**
AP_OK	Send (half-duplex conversation), or no change (full-duplex conversation)
AP_PARAMETER_CHECK AP_CONVERSATION_TYPE_MIXED AP_INVALID_VERB AP_INVALID_VERB_SEGMENT AP_STACK_TOO_SMALL AP_TP_BUSY AP_UNEXPECTED_DOS_ERROR	No change
AP_ALLOCATION_ERROR	Reset
AP_CONV_FAILURE_RETRY AP_CONV_FAILURE_NO_RETRY	Reset

primary_rc	**New state**
AP_DEALLOC_ABEND AP_DEALLOC_ABEND_PROG AP_DEALLOC_ABEND_SVC AP_DEALLOC_ABEND_TIMER AP_DEALLOC_NORMAL	Reset
AP_PROG_ERROR_PURGING AP_SVC_ERROR_PURGING	Receive (half-duplex conversation), Send_Receive (full-duplex conversation, verb issued in Send_Receive state), or Reset (full-duplex conversation, verb issued in Send_Only state)

Purged Data

If the conversation is in Receive state when the TP issues the [MC_]SEND_ERROR verb, incoming data is purged by APPC. This data includes the following:

- Data sent by the [MC_]SEND_DATA verb
- Return code indicators
- Confirmation requests
- Deallocation requests

APPC does not purge an incoming REQUEST_TO_SEND indicator.

Purged Return Code Indicators

The following primary return codes indicate that the remote TP or LU has detected an error, and would normally be reported on the next APPC verb issued by the local TP. However, when the local TP issues the [MC_]SEND_ERROR verb, these return codes are purged and replaced by other return codes.

The primary return code AP_OK replaces the following purged return code indicators:

```
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
```

The primary return code AP_DEALLOC_NORMAL replaces the following purged return code indicators:

primary_rc

```
AP_ALLOCATION_ERROR
```

secondary_rc

```
AP_ALLOCATION_FAILURE_NO_RETRY
AP_ALLOCATION_FAILURE_RETRY
AP_CONVERSATION_TYPE_MISMATCH
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_SECURITY_NOT_VALID
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_TP_NAME_NOT_RECOGNIZED
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
AP_TRANS_PGM_NOT_AVAIL_RETRY
```

```
AP_SEC_BAD_PROTOCOL_VIOLATION
AP_SEC_BAD_PASSWORD_EXPIRED
AP_SEC_BAD_PASSWORD_INVALID
AP_SEC_BAD_USERID_REVOKED
AP_SEC_BAD_USERID_INVALID
AP_SEC_BAD_USERID_MISSING
AP_SEC_BAD_PASSWORD_MISSING
AP_SEC_BAD_UID_NOT_DEFD_TO_GRP
AP_SEC_BAD_UNAUTHRZD_AT_RLU
AP_SEC_BAD_UNAUTHRZD_FROM_LLU
AP_SEC_BAD_UNAUTHRZD_TO_TP
AP_SEC_BAD_INSTALL_EXIT_FAILED
AP_SEC_BAD_PROCESSING_FAILURE
```

primary_rc

```
AP_DEALLOC_ABEND
AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
```

MC_SEND_EXPEDITED_DATA and SEND_EXPEDITED_DATA

The MC_SEND_EXPEDITED_DATA or SEND_EXPEDITED_DATA verb puts data in the local LU's expedited send buffer for transmission to the partner TP.

The data collected in the local LU's send buffer is transmitted to the partner LU (and partner TP) in the same way as for the [MC_]SEND_DATA verb. However, because the data is sent over the network as expedited data, it may arrive before data that was sent earlier using [MC_]SEND_DATA.

VCB Structure: MC_SEND_EXPEDITED_DATA

The definition of the VCB structure for the MC_SEND_EXPEDITED_DATA verb is as follows:

```
typedef struct mc_send_expedited_data
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                 /* Reserved            */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  unsigned char    rts_rcvd;
  unsigned char    expd_rcvd;
  AP_UINT16        dlen;
  unsigned char    *dptr;
  unsigned char    reserv4[2];
} MC_SEND_EXPEDITED_DATA;
```

VCB Structure: SEND_EXPEDITED_DATA

The definition of the VCB structure for the SEND_EXPEDITED_DATA verb is as follows:

```
typedef struct send_expedited_data
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                 /* Reserved            */
  AP_UINT16        primary_rc;
```

```
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  unsigned char    rts_rcvd;
  unsigned char    expd_rcvd;
  AP_UINT16        dlen;
  unsigned char    *dptr;
  unsigned char    reserv4[2];
} SEND_EXPEDITED_DATA;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_SEND_EXPEDITED_DATA
For the MC_SEND_EXPEDITED_DATA verb.

AP_B_SEND_EXPEDITED_DATA
For the SEND_EXPEDITED_DATA verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_SEND_EXPEDITED_DATA verb.

AP_BASIC_CONVERSATION
For the SEND_EXPEDITED_DATA verb.

If the verb is being issued on a full-duplex conversation or is being issued as a non-blocking verb, combine the value above (using a logical OR) with one or both of the following values:

AP_FULL_DUPLEX_CONVERSATION
The verb is being issued on a full-duplex conversation.

AP_NON_BLOCKING
The verb is being issued as a non-blocking verb.

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

dlen Number of bytes of data to be put in the local LU's send buffer. The range for this value is 0–86.

dptr Address of the buffer containing the data to be put in the local LU's send buffer.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

rts_rcvd
Request-to-send-received indicator. This parameter applies only in a half-duplex conversation; it is not used in a full-duplex conversation.

Possible values are:

AP_YES The partner TP has issued an [MC_]REQUEST_TO_SEND verb, which requests that the local TP change the conversation to Receive state. To change to Receive state, the local TP can use the [MC_]PREPARE_TO_RECEIVE, [MC_]RECEIVE_AND_WAIT, or [MC_]RECEIVE_AND_POST verb.

AP_NO The partner TP has not issued the [MC_]REQUEST_TO_SEND verb.

expd_rcvd
Expedited data indicator.

Possible values are:

AP_YES The partner TP has sent expedited data that the local TP has not yet received. To receive this data, the local TP can use the [MC_]RECEIVE_EXPEDITED_DATA verb.

This indicator can be set on a number of APPC verbs. It continues to be set on subsequent verbs until the local TP issues the [MC_]RECEIVE_EXPEDITED_DATA verb to receive the data.

AP_NO There is no expedited data waiting to be received.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

Expedited Data Not Supported: If the verb does not execute because the remote LU does not support expedited data, APPC returns the following parameter:

primary_rc

AP_EXPD_NOT_SUPPORTED_BY_LU

Conversation Deallocated: If the partner TP has deallocated the conversation, APPC returns the following value:

primary_rc

AP_CONVERSATION_ENDED
This verb was issued as a non-blocking verb and was queued behind an earlier verb. The partner TP issued the [MC_]DEALLOCATE verb as for AP_DEALLOC_NORMAL above, and the first verb in the queue returned with *primary_rc* set to AP_DEALLOC_NORMAL, indicating the end of the conversation. Any subsequent verbs in the queue then return with *primary_rc* set to AP_CONVERSATION_ENDED, indicating that the conversation had already ended before the verb could be processed.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

AP_SEND_EXPD_INVALID_LENGTH
The *dlen* parameter was set to a value that was not valid.

AP_INVALID_FORMAT
The reserved field *format* was set to a nonzero value.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

State Check: If the conversation is in the wrong state when the TP issues this verb, APPC returns the following parameters:

primary_rc
AP_STATE_CHECK

secondary_rc
Possible values are:

AP_EXPD_DATA_BAD_CONV_STATE
The local TP issued the [MC_]SEND_EXPEDITED_DATA verb, but the conversation was in Reset state.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, “Common Return Codes,” on page 273.

Possible return codes are:

primary_rc
AP_ALLOCATION_ERROR

secondary_rc
AP_CONVERSATION_TYPE_MISMATCH
AP_PIP_NOT_ALLOWED
AP_PIP_NOT_SPECIFIED_CORRECTLY
AP_SECURITY_NOT_VALID
AP_SYNC_LEVEL_NOT_SUPPORTED
AP_TP_NAME_NOT_RECOGNIZED
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
AP_TRANS_PGM_NOT_AVAIL_RETRY

primary_rc
AP_BACKED_OUT

secondary_rc
AP_BO_NO_RESYNC
AP_BO_RESYNC

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
AP_CONV_FAILURE_NO_RETRY
AP_CONV_FAILURE_RETRY
AP_CONVERSATION_TYPE_MIXED
AP_DUPLEX_TYPE_MIXED
AP_PROG_ERROR_PURGING
AP_INVALID_VERB
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

APPC does not return secondary return codes with these primary return codes.

The following primary return code is returned by the MC_SEND_EXPEDITED_DATA verb:

primary_rc

AP_DEALLOC_ABEND

APPC does not return a secondary return code with this primary return code.

The following primary return codes are returned by the SEND_EXPEDITED_DATA verb:

primary_rc

AP_DEALLOC_ABEND_PROG
AP_DEALLOC_ABEND_SVC
AP_DEALLOC_ABEND_TIMER
AP_SVC_ERROR_PURGING

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation must be in any state except Reset when the TP issues this verb.

State Change

State changes, summarized in the following table, are based on the *primary_rc* parameter.

primary_rc	**New state**
AP_OK	No change
AP_STATE_CHECK	No change
AP_PARAMETER_CHECK	
AP_CONVERSATION_TYPE_MIXED	
AP_INVALID_VERB	
AP_INVALID_VERB_SEGMENT	
AP_STACK_TOO_SMALL	
AP_TP_BUSY	
AP_UNEXPECTED_DOS_ERROR	
AP_ALLOCATION_ERROR	Reset
AP_CONV_FAILURE_RETRY	Reset
AP_CONV_FAILURE_NO_RETRY	
AP_CONVERSATION_ENDED	

primary_rc	**New state**
AP_DEALLOC_ABEND AP_DEALLOC_ABEND_PROG AP_DEALLOC_ABEND_SVC AP_DEALLOC_ABEND_TIMER	Reset

Waiting for Partner TP

In the same way as for [MC_]SEND_DATA, the [MC_]SEND_EXPEDITED_DATA verb may wait indefinitely because the partner TP has not issued an [MC_]RECEIVE_EXPEDITED_DATA verb. This is because the send-expedited buffer may fill up and APPC cannot transmit its contents to the partner LU because the partner LU has no buffers to receive the data.

MC_TEST_RTS and TEST_RTS

The MC_TEST_RTS or TEST_RTS verb determines whether a REQUEST_TO_SEND notification has been received from the partner TP.

Note: This verb can be used only in a half-duplex conversation; it is not valid in a full-duplex conversation.

Normally, if the partner TP issues an [MC_]REQUEST_TO_SEND verb, the local TP will be notified of this by the *rts_rcvd* parameter on a subsequent verb (this is a received parameter on a number of verbs). This is only reported on the first subsequent verb which can return this parameter, and not on any later verb. The [MC_]TEST_RTS verb enables the local TP to check if a request-to-send notification has been received at any time since the local TP was last in Receive state.

Instead of repeatedly issuing [MC_]TEST_RTS, the application can use [MC_]TEST_RTS_AND_POST, which is described in "MC_TEST_RTS_AND_POST and TEST_RTS_AND_POST" on page 237. This verb returns asynchronously when a REQUEST_TO_SEND notification is received from the partner TP. [MC_]TEST_RTS_AND_POST operates asynchronously in the same way as [MC_]RECEIVE_AND_POST, so that the application can issue other APPC verbs while it is outstanding.

VCB Structure: MC_TEST_RTS

UNIX

The definition of the VCB structure for the MC_TEST_RTS verb is as follows:

```
typedef struct mc_test_rts
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                 /* Reserved             */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  unsigned char    reserv3;
} MC_TEST_RTS;
```

VCB Structure: TEST_RTS

The definition of the VCB structure for the TEST_RTS verb is as follows:

```
typedef struct test_rts
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                 /* Reserved            */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  unsigned char    reserv3;
} TEST_RTS;
```

VCB Structure: MC_TEST_RTS (Windows)

WINDOWS

The definition of the VCB structure for the MC_TEST_RTS verb is as follows:

```
typedef struct mc_test_rts
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     reserv3;
} MC_TEST_RTS;
```

VCB Structure: TEST_RTS (Windows)

The definition of the VCB structure for the TEST_RTS verb is as follows:

```
typedef struct test_rts
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     reserv3;
} TEST_RTS;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_TEST_RTS
For the MC_TEST_RTS verb.

AP_B_TEST_RTS
For the TEST_RTS verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_TEST_RTS verb.

AP_BASIC_CONVERSATION
For the TEST_RTS verb.

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
Indicates whether a REQUEST_TO_SEND notification has been received from the partner TP. Possible values are:

AP_OK REQUEST_TO_SEND notification has been received.

AP_UNSUCCESSFUL
REQUEST_TO_SEND notification has not been received.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

UNIX

AP_INVALID_FORMAT
The reserved field *format* was set to a nonzero value.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using

the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

AP_TEST_INVALID_FOR_FDX
The local TP attempted to use the [MC_]TEST_RTS verb in a full-duplex conversation. This verb can be used only in a half-duplex conversation.

State Check: No state check errors occur for this verb.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
AP_CONVERSATION_TYPE_MIXED
AP_INVALID_VERB
AP_TP_BUSY
AP_UNEXPECTED_SYSTEM_ERROR

WINDOWS

AP_COMM_SUBSYSTEM_NOT_LOADED
AP_STACK_TOO_SMALL
AP_INVALID_VERB_SEGMENT

APPC does not return secondary return codes with these primary return codes.

State When Issued

The conversation can be in any state except Reset when the TP issues this verb.

State Change

The conversation state does not change for this verb.

MC_TEST_RTS_AND_POST and TEST_RTS_AND_POST

The MC_TEST_RTS_AND_POST or TEST_RTS_AND_POST verb informs the application when a REQUEST_TO_SEND notification has been received from the partner TP.

Note: This verb can be used only in a half-duplex conversation; it is not valid in a full-duplex conversation.

Normally, if the partner TP issues an [MC_]REQUEST_TO_SEND verb, the local TP will be notified of this by the *rts_rcvd* parameter on a subsequent verb (this is a received parameter on a number of verbs), or by a successful return code on the

[MC_]TEST_RTS verb. The [MC_]TEST_RTS_AND_POST verb enables the local TP to receive the REQUEST_TO_SEND notification asynchronously when it arrives, instead of having to issue verbs repeatedly to obtain the notification.

VCB Structure: MC_TEST_RTS_AND_POST

UNIX

The definition of the VCB structure for the MC_TEST_RTS_AND_POST verb is as follows:

```
typedef struct mc_test_rts_and_post
{
  AP_UINT16        opcode;
  unsigned char    opext;
  unsigned char    format;                    /* Reserved            */
  AP_UINT16        primary_rc;
  AP_UINT32        secondary_rc;
  unsigned char    tp_id[8];
  AP_UINT32        conv_id;
  void             (*callback)();
  unsigned char    reserv3;
} MC_TEST_RTS_AND_POST;
```

VCB Structure: TEST_RTS_AND_POST

The definition of the VCB structure for the TEST_RTS_AND_POST verb is as follows:

```
typedef struct test_rts_and_post
{
  AP_UINT16         opcode;
  unsigned char     opext;
  unsigned char     format;                  /* Reserved            */
  AP_UINT16         primary_rc;
  AP_UINT32         secondary_rc;
  unsigned char     tp_id[8];
  AP_UINT32         conv_id;
  void              (*callback)();
  unsigned char     reserv3;
} TEST_RTS_AND_POST;
```

VCB Structure: MC_TEST_RTS_AND_POST (Windows)

WINDOWS

The definition of the VCB structure for the MC_TEST_RTS_AND_POST verb is as follows:

```
typedef struct mc_test_rts_and_post
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     reserv3;
  unsigned long     sema;
} MC_TEST_RTS_AND_POST;
```

VCB Structure: TEST_RTS_AND_POST (Windows)

The definition of the VCB structure for the TEST_RTS_AND_POST verb is as follows:

```
typedef struct test_rts_and_post
{
  unsigned short    opcode;
  unsigned char     opext;
  unsigned char     reserv2;
  unsigned short    primary_rc;
  unsigned long     secondary_rc;
  unsigned char     tp_id[8];
  unsigned long     conv_id;
  unsigned char     reserv3;
  unsigned long     sema;
} TEST_RTS_AND_POST;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode Possible values are:

AP_M_TEST_RTS_AND_POST
For the MC_TEST_RTS_AND_POST verb.

AP_B_TEST_RTS_AND_POST
For the TEST_RTS_AND_POST verb.

opext Possible values are:

AP_MAPPED_CONVERSATION
For the MC_TEST_RTS_AND_POST verb.

AP_BASIC_CONVERSATION
For the TEST_RTS_AND_POST verb.

tp_id Identifier for the local TP.

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

conv_id
Conversation identifier.

The value of this parameter was returned by the [MC_]ALLOCATE verb in the invoking TP or by RECEIVE_ALLOCATE in the invoked TP.

UNIX

callback
Address of the callback routine which APPC is to call when a REQUEST_TO_SEND notification is received. For more information, see "Usage Notes" on page 242.

WINDOWS

sema A Windows event handle, obtained by calling one of the two Windows

functions CreateEvent or OpenEvent. APPC signals this event handle to inform the TP when the REQUEST_TO_SEND notification is received.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Note: When this verb is issued, it returns immediately with a *primary_rc* which indicates whether or not the verb was issued successfully. The only returned parameters which are valid at this stage are *primary_rc* and *secondary_rc* (if the *primary_rc* is not AP_OK). The possible *primary_rc* and *secondary_rc* values are as described later in this section.

If this *primary_rc* is AP_OK, the verb has successfully begun to wait for REQUEST_TO_SEND notification. When the verb has completed (either because the notification was received, or because it was terminated by the end of the conversation or by an error), APPC calls the supplied callback routine. At this point, the returned parameters are as shown below. The *primary_rc* and *secondary_rc* parameters will now have new values indicating whether or not the REQUEST_TO_SEND notification was received, and should be examined again.

Successful Execution

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

primary_rc

AP_OK REQUEST_TO_SEND notification was received.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc

AP_PARAMETER_CHECK

secondary_rc

Possible values are:

AP_BAD_CONV_ID
The value of *conv_id* did not match a conversation identifier assigned by APPC.

AP_BAD_TP_ID
The value of *tp_id* did not match a TP identifier assigned by APPC.

AP_INVALID_FORMAT
The reserved field *format* was set to a nonzero value.

AP_SYNC_NOT_ALLOWED
The application issued this verb within a callback routine, using

the synchronous APPC entry point. Any verb issued from a callback routine must use the asynchronous entry point.

AP_INVALID_CALLBACK_HANDLE
The *callback* parameter was set to a null pointer, and the verb was issued using the synchronous entry point (or using the asynchronous entry point with a null pointer to a callback routine). For more information, see "Usage Notes" on page 242.

AP_TEST_INVALID_FOR_FDX
The local TP attempted to use the [MC_]TEST_RTS_AND_POST verb in a full-duplex conversation. This verb can be used only in a half-duplex conversation.

State Check: No state check errors occur for this verb.

Verb Canceled: This return code cannot be returned as the initial return code, but only as the subsequent return code if the initial return code is AP_OK. If the verb did not execute because it was canceled by another verb issued by the TP, APPC returns the following parameter:

primary_rc

AP_CANCELLED
The local TP issued one of the following verbs while [MC_]TEST_RTS_AND_POST was outstanding:

- DEALLOCATE with *dealloc_type* set to AP_ABEND_PROG, AP_ABEND_SVC, or AP_ABEND_TIMER
- MC_DEALLOCATE with *dealloc_type* set to AP_ABEND
- [MC_]SEND_ERROR
- TP_ENDED

Issuing one of these verbs causes the [MC_]TEST_RTS_AND_POST verb to be canceled. The callback routine is not called.

Conversation Ended: If the verb returns because the conversation has ended, APPC returns the following parameter:

primary_rc
AP_UNSUCCESSFUL

Other Conditions: If the verb does not execute because other conditions exist, APPC returns primary return codes (and, if applicable, secondary return codes). For information about these return codes, see Appendix B, "Common Return Codes," on page 273.

Possible return codes are:

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR
AP_CONVERSATION_TYPE_MIXED
AP_INVALID_VERB
AP_TP_BUSY

APPC does not return secondary return codes with these primary return codes.

State When Issued

The TP can issue [MC_]TEST_RTS_AND_POST when the conversation is in any state except Reset.

State Change

The conversation state does not change for this verb.

Usage Notes

This section provides additional usage information about the following topics:

- Callback routine
- Processing while the verb is pending
- How the TP uses the verb
- Avoiding indefinite waits

Callback Routine

UNIX

The application supplies a pointer to a callback routine as one of the parameters to the VCB. This section describes how Communications Server uses this routine, and the functions that it must perform.

The callback routine is defined as follows:

```
void (*callback) (
                    void *          vcb,
                    unsigned char   tp_id[8],
                    AP_UINT32       conv_id
                  );
```

Communications Server calls the routine with the following parameters:

vcb Pointer to the VCB supplied by the application, including the returned parameters set by Communications Server.

tp_id The 8-byte TP identifier of the TP in which the verb was issued.

conv_id
The conversation identifier of the conversation in which the verb was issued.

The callback routine need not use all of these parameters. It may perform all the necessary processing on the returned VCB, or may simply set a variable to inform the main program that the verb has completed.

The application can issue further APPC verbs from within the callback routine, if required. However, these must be asynchronous verbs. Any synchronous verbs issued from within a callback routine will be rejected with the return codes AP_PARAMETER_CHECK and AP_SYNC_NOT_ALLOWED.

Note: If the application issues the [MC_]TEST_RTS_AND_POST verb using the asynchronous APPC entry point, there are two callback routines specified: one in the VCB, the other supplied as a parameter to the entry point. In general, APPC uses the callback routine specified in the VCB and ignores

the one on the entry point; however, if the application supplies a null pointer for the callback routine in the VCB, APPC uses the callback routine on the entry point.

Continuing with Other Processing While the Verb Is Pending

Because the [MC_]TEST_RTS_AND_POST verb returns immediately without waiting for data to arrive, the TP can continue other processing while waiting for it to complete. However, the following points should be noted:

- The VCB supplied to the [MC_]TEST_RTS_AND_POST verb continues to be used until the callback routine returns. The TP must not change any fields in the VCB during this time. If it issues any other APPC verb while [MC_]TEST_RTS_AND_POST is outstanding, it must use another VCB for the new verb.
- Only one [MC_]TEST_RTS_AND_POST verb per conversation can be active at any time.

How the TP Uses the Verb

To use the [MC_]TEST_RTS_AND_POST verb, the local TP performs the following steps:

Using [MC_]TEST_RTS_AND_POST

1. Issues the [MC_]TEST_RTS_AND_POST verb.
2. Checks the value of the primary return code *primary_rc*:
   - If the primary return code is `AP_OK`, the verb is waiting for a REQUEST_TO_SEND notification from the partner TP. While receiving data asynchronously, the local TP can do the following:
     - Perform tasks not related to this conversation
     - Issue other APPC verbs on this conversation
     - Prematurely cancel the [MC_]TEST_RTS_AND_POST verb by issuing one of the following verbs:
       - DEALLOCATE with *dealloc_type* set to `AP_ABEND_PROG`, `AP_ABEND_SVC`, or `AP_ABEND_TIMER`
       - MC_DEALLOCATE with *dealloc_type* set to `AP_ABEND`
       - SEND_ERROR
       - TP_ENDED
   - If, however, the primary return code is not `AP_OK`, the [MC_]TEST_RTS_AND_POST verb has failed. In this case, the local TP does not perform Steps 3 and 4.
3. Checks that the callback routine (supplied as a parameter on this verb) has been called by APPC. When a REQUEST_TO_SEND notification is received from the partner TP, APPC calls this routine.
4. Checks the new value of the primary return code *primary_rc*.
   - If the primary return code is `AP_OK`, the partner TP has issued [MC_]REQUEST_TO_SEND.
   - If the primary return code is not `AP_OK`, the application should check the *primary_rc* and *secondary_rc* parameters to determine the action it should take.

Avoiding Indefinite Waits

If the local TP issues the [MC_]TEST_RTS_AND_POST verb and subsequently waits for the callback routine to be called, it will be suspended until REQUEST_TO_SEND notification is received from the partner TP. It could wait indefinitely if the partner TP does not issue [MC_]REQUEST_TO_SEND. If you need to have the TP operating continuously, avoid waiting on the callback routine, or use the [MC_]TEST_RTS verb.

Chapter 5. TP Server Verbs

UNIX

This chapter contains a description of each APPC TP server verb. The following information is provided for each verb:

- Definition of the verb.
- Structure defining the verb control block (VCB) used by the verb. The structure is defined in the TP Server header file **/usr/include/sna/tpsrv_c.h** (AIX) or **/opt/ibm/sna/include/tpsrv_c.h** (Linux). (Parameters beginning with *rsrvd* are reserved.)
- Parameters (VCB fields) supplied to and returned by APPC. For each parameter, the following information is provided:
  - Description
  - Possible values
  - Additional information
- Additional information describing the use of the verb.

Note:

1. TP server verbs must be issued using the asynchronous entry point `APPC_Async`, and not the synchronous entry point `APPC`. For more information about these entry points, see Chapter 2, "Writing Transaction Programs," on page 25.
2. TP server verbs do not affect APPC conversations or states.

Most parameters supplied to and returned by APPC for the TP Server verbs are hexadecimal values. To simplify coding, these values are represented by meaningful symbolic constants defined in the header file **values_c.h**, which is included by the TP Server header file **tpsrv_c.h**. For example, the *opcode* parameter of the REGISTER_TP_SERVER verb is the hexadecimal value represented by the symbolic constant `AP_REGISTER_TP_SERVER`.

It is important that you use the symbolic constant and not the hexadecimal value when setting values for supplied parameters, or when testing values of returned parameters. This is because different operating systems store these values differently in memory, so the value shown may not be in the format recognized by your system.

The TP server verbs are described in the following order:

REGISTER_TP_SERVER
UNREGISTER_TP_SERVER
REGISTER_TP
UNREGISTER_TP
QUERY_ATTACH
ACCEPT_ATTACH
REJECT_ATTACH
ABORT_ATTACH

REGISTER_TP_SERVER

The REGISTER_TP_SERVER verb is used to notify Communications Server that the application is capable of automatically starting transaction programs (TPs).

VCB Structure: REGISTER_TP_SERVER

The definition of the VCB structure for the REGISTER_TP_SERVER verb is as follows:

```
typedef struct register_tp_server
{
   AP_UINT16        opcode;
   unsigned char    rsrvd1;                 /* Reserved             */
   unsigned char    rsrvd2;                 /* Reserved             */
   AP_UINT16        primary_rc;
   AP_UINT32        secondary_rc;
   AP_UINT32        tps_id;
   unsigned char    tp_file_updates;
   AP_NOTIFY_CB     notify_cb;
} REGISTER_TP_SERVER;

  typedef void (*AP_NOTIFY_CB) (
                                unsigned char   reason,
                                unsigned char   attach_id[8],
                                AP_CORR         app_corr
                               );

 typedef union ap_corr {
                          void *          corr_p;
                          AP_UINT32       corr_l;
                          AP_INT32        corr_i;
                        } AP_CORR;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode AP_REGISTER_TP_SERVER

tp_file_updates

Requests whether the application should be notified when the **sna_tps** TP configuration file is updated. Possible values are:

AP_YES The application requests callbacks to notify it that the **sna_tps** file has been changed.

AP_NO The application does not require notification of changes to the **sna_tps** file.

notify_cb

The address of the notification callback function. APPC uses this function in conjunction with the value of the *app_corr* parameter specified on the REGISTER_TP verb to notify a TP server that one of the following has occurred:

- A suitable Attach is available
- The **sna_tps** TP configuration file has changed (if the application requested this notification by setting the *tp_file_updates* parameter to AP_YES).

For more information on how the notification callback function is used, see "Callback Routine" on page 247.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

tps_id A unique identifier for this TP server. After an application registers itself as a TP server, the value of the *tps_id* parameter is valid for that process only. The value of the *tps_id* parameter is not valid across process boundaries. If another application tries to use this value of the *tps_id* parameter on another verb, that verb is rejected with a *primary_rc* value of AP_PARAMETER_CHECK and a *secondary_rc* value of AP_BAD_TPS_ID.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_INVALID_CALLBACK
The callback function address was not valid.

Register Failure: If the application cannot be registered as a TP server, APPC returns the following parameters:

primary_rc
AP_REGISTER_FAIL

Other Conditions: If the verb does not execute because other conditions exist, APPC returns the following primary return code. For a list of return codes common to all verbs, see Appendix B, "Common Return Codes," on page 273.

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

Usage Notes

This section provides additional usage information about the callback routine.

Callback Routine

The application supplies a pointer to a callback routine as one of the parameters to the VCB. This section describes how Communications Server uses this routine and the functions that it must perform.

The callback routine is defined as follows:

```
typedef void (*AP_NOTIFY_CB) (
                              unsigned char   reason,
                              unsigned char   attach_id[8],
                              AP_CORR         app_corr
                             );
typedef union ap_corr {
                        void *          corr_p;
                        AP_UINT32       corr_l;
                        AP_INT32        corr_i;
                      } AP_CORR;
```

Communications Server calls the routine with the following parameters:

reason Type of notification. Possible values are:

AP_ATTACH
An Attach has arrived for a TP registered by this TP server. In this case, the *attach_id* parameter is passed into the notification callback because the arrival of an Attach is used to do one or more of the following:

- Optionally query for more information about automatically starting a TP
- Reject the Attach if necessary
- Identify which TP to automatically start for RECEIVE_ALLOCATE processing

AP_TP_FILE_CHANGE
The **sna_tps** TP configuration file has been modified.

attach_id
The ID of the attach, as returned by the attach notification callback.

app_corr
The correlator value supplied by the application. This value allows the application to correlate the returned information with its other processing. The meaning of the correlator passed into the notification callback depends on the notification type as indicated by the value of the *reason* flag:

- If *reason* is set to AP_ATTACH, the correlator is the correlator that was specified by the application on the REGISTER_TP verb. This allows the application to correlate the Attach with the correct registered TP.
- If *reason* is set to AP_TP_FILE_CHANGE, the correlator is the value of the *tps_id* parameter on the REGISTER_TP_SERVER verb.

The callback routine need not use all of these parameters. It may perform all the necessary processing on the returned VCB, or it may simply set a variable to inform the main program that the verb has completed.

UNREGISTER_TP_SERVER

The UNREGISTER_TP_SERVER verb is used when an application no longer wishes to receive attach notifications.

VCB Structure: UNREGISTER_TP_SERVER

The definition of the VCB structure for the UNREGISTER_TP_SERVER verb is as follows:

```
typedef struct unregister_tp_server
{
   AP_UINT16        opcode;
```

```
   unsigned char    rsrvd1;                 /* Reserved            */
   unsigned char    rsrvd2;                 /* Reserved            */
   AP_UINT16        primary_rc;
   AP_UINT32        secondary_rc;
   AP_UINT32        tps_id;
} UNREGISTER_TP_SERVER;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode AP_UNREGISTER_TP_SERVER

tps_id The ID of the TP server to be unregistered, as returned on a previous REGISTER_TP_SERVER verb.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_TPS_ID
The specified value of the *tps_id* parameter was not recognized.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns the following primary return code. For a list of return codes common to all verbs, see Appendix B, “Common Return Codes,” on page 273.

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

REGISTER_TP

The REGISTER_TP verb is used to tell Service Manager the name of a TP whose attaches are to be handled by the TP server. It can also be used to change the TP type or receive allocate timeout for a TP that has already been registered.

VCB Structure: REGISTER_TP

The definition of the VCB structure for the REGISTER_TP verb is as follows:

```
typedef struct register_tp
{
   AP_UINT16        opcode;
   unsigned char    rsrvd1;                 /* Reserved             */
   unsigned char    rsrvd2;                 /* Reserved             */
   AP_UINT16        primary_rc;
   AP_UINT32        secondary_rc;
   AP_UINT32        tps_id;
   AP_UINT32        res_id;
   unsigned char    tp_name[64];
   char             lu_alias[8];
   unsigned char    fqplu_name[17];
   unsigned char    tp_type;
   AP_INT32         rcv_alloc_timeout;
   AP_UINT16        modify_existing;
   AP_CORR          app_corr;
} REGISTER_TP;
```

```
typedef union ap_corr {
                        void *          corr_p;
                        AP_UINT32       corr_l;
                        AP_INT32        corr_i;
                      } AP_CORR;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode AP_REGISTER_TP

tps_id The ID of a TP server, as returned on a previous REGISTER_TP_SERVER verb.

res_id If REGISTER_TP is being used to change an existing TP registration (the *modify_existing* parameter is set to AP_YES), this parameter specifies the unique identifier for this resource that was returned on the original REGISTER_TP verb. Otherwise, this parameter is reserved.

tp_name

The name of the TP being registered. Specify this name in EBCDIC padded with EBCDIC spaces, if necessary, to a length of 64 characters. Specify a value of 64 EBCDIC spaces (0x40) for a TP for which all attaches will be handled.

lu_alias

The local LU alias. Specify this name in ASCII padded with ASCII spaces, if necessary, to a length of eight characters. Specify a value of eight ASCII spaces (0x20) for an LU for which all attaches will be handled.

fqplu_name

The fully qualified name of the partner LU. Specify one of the following EBCDIC strings padded with EBCDIC spaces, if necessary, to a length of 17 characters:

- A fully qualified name in EBCDIC to indicate that a match is to be made only with an attach that has the same fully qualified name.
- A value of all EBCDIC spaces (0x40) to indicate that any partner LU name is considered a match
- A partial name, followed by an EBCDIC * (0x5C) to indicate a wildcard LU name.

tp_type

Type of the TP being registered. Possible values are:

AP_TP_TYPE_QUEUED
Incoming attaches are queued to running copies of the TP before attempting to start a new TP or queue the attach to wait for a suitable TP.

AP_TP_TYPE_QUEUED_BROADCAST
Same as AP_TP_TYPE_QUEUED except that the existence of the TP is broadcast around the Communications Server domain. Broadcasting the existence of this TP obviates the need to configure attach routing data for many local LUs is they are all handled by the same TP on a single machine.

AP_TP_TYPE_NON_QUEUED
A new instance of the TP is started for each attach received, unless a running instance has a RECEIVE_ALLOCATE verb outstanding.

rcv_alloc_timeout
The amount of time in seconds that the TP's RECEIVE_ALLOCATE verb should wait for an automatically started TP. Possible values are:

0 (zero)
Do not wait. This is normally the value specified because a TP server starts TPs in response to an attach arriving, so there should always be an attach available for a TP's RECEIVE_ALLOCATE. The only exception to this is if the attach has timed out while the TP server is starting the TP.

-1 Wait indefinitely.

x where x is greater than 0
Wait the number of seconds indicated by x.

modify_existing
Specifies whether this verb is being used to change an existing registration or to register a new TP. Possible values are:

AP_YES This verb is being used to change the *rcv_alloc_timeout* parameter, the *type* parameter, or both of these parameters for an existing registration. The following restrictions apply:

- The verb must be issued by the same TP Server program that issued the original REGISTER_TP verb.
- The *res_id* parameter must be specified, and must match the value returned on the original REGISTER_TP verb.
- The *rcv_alloc_timeout* parameter, the *type* parameter, or both of these parameters can be changed from the original REGISTER_TP verb, but all other supplied parameters must match the value used on the original REGISTER_TP verb.

AP_NO This verb is being used to register a new TP.

app_corr
An application provided correlator passed into the attach notification callback. For more information, see "Usage Notes" on page 247.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

res_id The unique identifier for this resource.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_INVALID_TP_NAME
The value specified for the *tp_name* parameter was not valid.

AP_INVALID_LU_ALIAS
The value specified for the *lu_alias* parameter was not valid.

AP_INVALID_FQ_LU_NAME
The value specified for the *fqplu_name* parameter was not valid.

AP_INVALID_TIMEOUT
The value specified for the *rcv_alloc_timeout* parameter was not valid.

AP_BAD_TPS_ID
The value specified for the *tps_id* parameter was not recognized.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns the following primary return code. For a list of return codes common to all verbs, see Appendix B, "Common Return Codes," on page 273.

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

UNREGISTER_TP

The UNREGISTER_TP verb is used to notify the Service Manager that the application does not want to receive Attach notifications for the specified TP.

VCB Structure: UNREGISTER_TP

The definition of the VCB structure for the UNREGISTER_TP verb is as follows:

```
typedef struct unregister_tp
{
   AP_UINT16        opcode;
   unsigned char    rsrvd1;                 /* Reserved             */
   unsigned char    rsrvd2;                 /* Reserved             */
   AP_UINT16        primary_rc;
   AP_UINT32        secondary_rc;
   AP_UINT32        tps_id;
   AP_UINT32        res_id;
} UNREGISTER_TP;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode AP_UNREGISTER_TP

tps_id The ID of the TP server, as returned on a previous REGISTER_TP_SERVER verb.

res_id The unique identifier for this resource, as returned on a previous REGISTER_TP verb.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_TPS_ID
The value specified for the *tps_id* parameter was not recognized.

AP_BAD_RES_ID
The value specified for the *res_id* parameter was not recognized.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns the following primary return code. For a list of return codes common to all verbs, see Appendix B, "Common Return Codes," on page 273.

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

QUERY_ATTACH

The QUERY_ATTACH verb is used to retrieve information about an Attach of which Communications Server has notified the application. This verb is optional. If the data represented by the TP server correlator passed into the attach callback is sufficient for the TP server's use, the TP server does not need to issue this verb.

For security reasons, the user id and password information in the attach are made available only to a TP server whose effective user id is `root`. For applications that do not run as `root`, the returned attach has had the access security subfields stripped from it.

This verb can be issued as many times as required by the TP server. However, the PIP data can be extracted only once. To retrieve attach information without retrieving the PIP data, issued QUERY_ATTACH with *max_pip_len* set to `0` (zero).

VCB Structure: QUERY_ATTACH

The definition of the VCB structure for the QUERY_ATTACH verb is as follows:

```
typedef struct query_attach
{
   AP_UINT16         opcode;
   unsigned char     rsrvd1;                 /* Reserved            */
   unsigned char     rsrvd2;                 /* Reserved            */
   AP_UINT16         primary_rc;
   AP_UINT32         secondary_rc;
   AP_UINT32         tps_id;
   unsigned char     attach_id[8];
   unsigned char     tp_name[64];
   char              lu_alias[8];
   unsigned char     fq_plu_name[17]
   unsigned char     mode_name[8];
   AP_UINT16         max_pip_len;
   AP_UINT16         pip_dlen;
   unsigned char     *pip_dptr;
   AP_UINT16         max_fmh5_len;
   AP_UINT16         fmh5_dlen;
   unsigned char     *fmh5_dptr;
} QUERY_ATTACH;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode `AP_QUERY_ATTACH`

tps_id The ID of the TP server, as returned on a previous REGISTER_TP_SERVER verb.

attach_id
The ID of the attach, as returned by the attach notification callback.

max_pip_len
The maximum buffer space available for PIP data.

pip_dptr
Pointer to caller-allocated buffer for returned attach pip data buffer.

max_fmh5_len
The maximum buffer space available for FM header 5 (FMH5) data.

fmh5_dptr
Pointer to caller-allocated buffer for returned attach FMH5 data buffer.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
`AP_OK`

tp_name
The attach TP name.

lu_alias
The attach local LU alias.

fq_plu_name
The attach fully qualified partner LU name.

mode_name
The attach mode name.

pip_dlen
The actual number of bytes of PIP data returned.

fmh5_dlen
The actual number of bytes of FMH5 data returned.

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_ATTACH_ID
The value specified for the *attach_id* parameter was not recognized.

AP_BAD_TPS_ID
The value specified for the *tps_id* parameter was not recognized.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns the following primary return code. For a list of return codes common to all verbs, see Appendix B, "Common Return Codes," on page 273.

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

ACCEPT_ATTACH

The ACCEPT_ATTACH verb is used to continue the processing of the attach by this TP server.

VCB Structure: ACCEPT_ATTACH

The definition of the VCB structure for the ACCEPT_ATTACH verb is as follows:

```
typedef struct accept_attach
{
   AP_UINT16        opcode;
   unsigned char    rsrvd1;                 /* Reserved            */
   unsigned char    rsrvd2;                 /* Reserved            */
   AP_UINT16        primary_rc;
   AP_UINT32        secondary_rc;
   AP_UINT32        tps_id;
   unsigned char    attach_id[8];
} ACCEPT_ATTACH;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode AP_ACCEPT_ATTACH

tps_id The ID of the TP server, as returned on a previous REGISTER_TP_SERVER verb.

attach_id
The ID of the attach, as returned by the attach notification callback.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_ATTACH_ID
The value specified for the *attach_id* parameter was not recognized.

AP_BAD_TPS_ID
The value specified for the *tps_id* parameter was not recognized.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns the following primary return code. For a list of return codes common to all verbs, see Appendix B, "Common Return Codes," on page 273.

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

REJECT_ATTACH

The REJECT_ATTACH verb is used to end the processing of the attach by this TP server.

VCB Structure: REJECT_ATTACH

The definition of the VCB structure for the REJECT_ATTACH verb is as follows:

```
typedef struct reject_attach
{
   AP_UINT16       opcode;
   unsigned char   rsrvd1;                /* Reserved            */
```

```
   unsigned char    rsrvd2;                 /* Reserved            */
   AP_UINT16        primary_rc;
   AP_UINT32        secondary_rc;
   AP_UINT32        tps_id;
   unsigned char    attach_id[8];
   AP_UINT32        reason;
} REJECT_ATTACH;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode AP_REJECT_ATTACH

tps_id The ID of the TP server, as returned on a previous REGISTER_TP_SERVER verb.

attach_id
The ID of the attach, as returned by the attach notification callback.

reason The reason the automatic start is being rejected. The value is an SNA sense code as shown in "SNA Sense Codes" on page 258.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_ATTACH_ID
The value specified for the *attach_id* parameter was not recognized.

AP_BAD_TPS_ID
The value specified for the *tps_id* parameter was not recognized.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns the following primary return code. For a list of return codes common to all verbs, see Appendix B, "Common Return Codes," on page 273.

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

SNA Sense Codes: Table 10 shows common SNA sense codes used to reject an Attach as follows:

Table 10. Common SNA Sense Codes

Symbol	Value	Meaning
AP_SECURITY_INVALID	080F6051	security not valid
AP_SEC_BAD_PASSWORD_EXPIRED	080FFF00	password has expired
AP_SEC_BAD_PASSWORD_INVALID	080FFF01	password is not valid
AP_SEC_BAD_USERID_REVOKED	080FFF02	user ID has been revoked
AP_SEC_BAD_USERID_INVALID	080FFF03	user ID is not valid
AP_SEC_BAD_USERID_MISSING	080FFF04	user ID is missing
AP_SEC_BAD_PASSWORD_MISSING	080FFF05	password is missing
AP_SEC_BAD_GROUP_INVALID	080FFF06	group is not valid
AP_SEC_BAD_UID_REVOKED_IN_GRP	080FFF07	user ID is revoked in the specified group
AP_SEC_BAD_UID_NOT_DEFD_TO_GRP	080FFF08	user ID is not defined in the specified group
AP_SEC_BAD_UNAUTHRZD_AT_RLU	080FFF09	user ID is not defined to use the remote LU
AP_SEC_BAD_UNAUTHRZD_FROM_LLU	080FFF0A	user ID is not defined to use the remote LU from the local LU
AP_SEC_BAD_UNAUTHRZD_TO_TP	080FFF0B	user ID is not defined to use the TP at the remote LU
AP_SEC_BAD_INSTALL_EXIT_FAILED	080FFF0C	installation exit processing at the remote LU failed
AP_SEC_BAD_PROCESSING_FAILURE	080FFF0D	processing failed between the local and remote LUs, but the condition is temporary
AP_SEC_BAD_PROTOCOL_VIOLATION	080F6058	a protocol violation resulted in a security validation failure
AP_TRANS_PGM_NOT_AVAIL_RETRY	084B6031	TP not available—retry
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY	084C0000	TP not available—no retry
AP_PIP_INVALID	1008201D	PIP data not valid
AP_ATTACH_LEN_INVALID	10086000	attach length not valid
AP_SECURITY_LEN_INVALID	10086005	security length not valid

Table 10. Common SNA Sense Codes (continued)

Symbol	Value	Meaning
AP_PARM_LEN_INVALID	10086009	parameter length not valid
AP_LUWID_LEN_INVALID	10086011	LUWID length not valid
AP_TP_NAME_NOT_RECOGNIZED	10086021	TP name not recognized
AP_PIP_NOT_ALLOWED	10086031	PIP data not allowed
AP_PIP_FIELDS_REQUIRED	10086032	PIP data required
AP_CONVERSATION_TYPE_MISMATCH	10086034	Conversation type mismatch
AP_LU_CAPABILITY_CONFLICT	10086040	attach LU capabilities conflict with bind
AP_SYNC_LEVEL_NOT_SUPPORTED	10086041	sync level not supported by TP

Note: The generic AP_SECURITY_INVALID sense code (080F651) may be substituted by Communications Server for sense codes in the range 080FFF00–080FFFFF if the remote LU does not want the extended security information.

ABORT_ATTACH

The ABORT_ATTACH verb is used to end the processing of the attach by this TP server after the attach has been accepted using an ACCEPT_ATTACH verb because the TP server or TP has encountered an error during further processing. For example, the TP server was unable to fork to the TP. The ABORT_ATTACH verb can be issued by both the TP server and the TP processes.

VCB Structure: ABORT_ATTACH

The definition of the VCB structure for the ABORT_ATTACH verb is as follows:

```
typedef struct abort_attach
{
   AP_UINT16        opcode;
   unsigned char    rsrvd1;                 /* Reserved            */
   unsigned char    rsrvd2;                 /* Reserved            */
   AP_UINT16        primary_rc;
   AP_UINT32        secondary_rc;
   AP_UINT32        tps_id;
   unsigned char    attach_id[8];
   AP_UINT32        reason;
} ABORT_ATTACH;
```

Supplied Parameters

The TP supplies the following parameters to APPC:

opcode AP_ABORT_ATTACH

tps_id The ID of the TP server, as returned on a previous REGISTER_TP_SERVER verb.

attach_id
The ID of the attach to be aborted, as returned by the attach notification callback.

reason The reason the automatic start is being aborted. The value is an SNA sense code as shown in "SNA Sense Codes" on page 258.

Returned Parameters

After the verb executes, APPC returns parameters to indicate whether the execution was successful and, if not, to indicate the reason the execution was not successful.

Successful Execution

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

primary_rc
AP_OK

Unsuccessful Execution

If the verb does not execute successfully, APPC returns a primary return code parameter to indicate the type of error and a secondary return code parameter to provide specific details about the reason for unsuccessful execution.

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

primary_rc
AP_PARAMETER_CHECK

secondary_rc
Possible values are:

AP_BAD_ATTACH_ID
The value specified for the *attach_id* parameter was not recognized.

AP_BAD_TPS_ID
The value specified for the *tps_id* parameter was not recognized.

Other Conditions: If the verb does not execute because other conditions exist, APPC returns the following primary return code. For a list of return codes common to all verbs, see Appendix B, "Common Return Codes," on page 273.

primary_rc
AP_UNEXPECTED_SYSTEM_ERROR

Chapter 6. Sample Transaction Programs

The Communications Server APPC sample transaction programs (TPs) illustrate the use of APPC verbs in a mapped conversation.

The programs are provided with Communications Server as **asample1.c** and **asample2.c**, in the directory **/usr/lib/sna/samples** (AIX) or **/opt/ibm/sna/samples** (Linux).

The following information is provided in this chapter:

- Processing overview of the sample TPs
- Pseudocode for each TP
- Instructions for compiling, linking, and running the TPs

Processing Overview

The TPs presented in this chapter enable a user to browse through a file on another system. The user is presented with a single data block at a time, in hexadecimal and character format. After each block, a user can request the next block, request the previous block, or quit.

asample1 (the invoking TP) sends a file name to **asample2** (the invoked TP). If **asample2** locates the file, it returns the first data block to **asample1**; otherwise, it deallocates the conversation and ends.

If **asample1** receives a block, it displays the block on the screen and waits for the user to enter **F** for forward, **B** for backward, or **Q** for quit. If the user selects forward or backward, **asample1** sends the request to **asample2**, which in turn sends the appropriate block. This process continues until the user selects the quit option, at which time **asample1** deallocates the conversation and both programs end.

If the user asks for the next block and **asample2** has sent the last one, **asample2** wraps to the beginning of file. Similarly, **asample2** wraps to send the last block if the user requests the previous one and the first block is displayed.

Neither program attempts to recover from errors. A bad return code from APPC causes the program to terminate with an explanatory message.

Pseudocode

This section contains the pseudocode for the TPs **asample1** and **asample2**.

asample1 (Invoking TP)

The pseudocode for **asample1** (the invoking TP) is as follows:

```
TP_started
mc_allocate (sync_level none)
mc_send_data (data = filename), send type prepare_to_receive_flush
do while no error and prompt not Q
    mc_receive_and_wait
    if data block received
       display data block
```

```
    else if permission to send received
        get user prompt (F, B, or Q)
        if prompt = F or B     /* Not Q */
           mc_send_data (data = prompt), send type p_to_r_flush
        endif
    endif
  end do
  mc_deallocate
  TP_ended
```

asample2 (Invoked TP)

The pseudocode for **asample2** (the invoked TP) is as follows:

```
receive_allocate
do while conversing
   mc_receive_and_wait (return status with data)
   if data received and send indication received
      if first time (data = filename)
         open file
         if file not found
             mc_deallocate
             set conversing false
         endif
       else (data = prompt)
           read and store prompt
       endif
       if (conversing)
           read file block
           mc_send_data (file block)
       endif
    else if deallocate received
       set conversing false
    endif
    end while conversing
   close file
   tp_ended
```

Testing the TPs

After examining the source code for the two programs, you may want to test the programs.

Although APPC is normally used for communications between a local and a remote computer, you may find it convenient to run both TPs on the same Communications Server computer for testing purposes.

To compile and link the TPs, take the following steps.

1. Copy the two files **asample1.c** and **asample2.c** from **/usr/lib/sna/samples** (AIX) or **/opt/ibm/sna/samples** (Linux) to a private directory.
2. To compile and link the programs for AIX, use the following commands:

```
cc -o asample1 -I /usr/include/sna -bimport:/usr/lib/sna/appc_r.exp -bimport:/usr/lib/sna/csv_r.exp asample1.c
cc -o asample2 -I /usr/include/sna -bimport:/usr/lib/sna/appc_r.exp -bimport:/usr/lib/sna/csv_r.exp asample2.c
```

To compile and link the programs for Linux, use the following commands:

```
gcc -o asample1 -I /opt/ibm/sna/include -L /opt/ibm/sna/lib -lappc -lsna_r -lcsv -lpLiS -lpthread asample1.c
gcc -o asample2 -I /opt/ibm/sna/include -L /opt/ibm/sna/lib -lappc -lsna_r -lcsv -lpLiS -lpthread asample2.c
```

To run the TPs, perform the following steps. Note that some of these steps involve updating the Communications Server configuration, which is usually performed by the System Administrator.

The TPs can run on the same computer, or on separate computers. In the following steps, the "source computer" is the computer where the invoking TP **asample1** runs, and the "target computer" is the computer where the invoked TP **asample2** runs.

1. If you are running the TPs on separate computers, configure the communications link to support CP-CP sessions between the source and target computers. See *IBM Communications Server for Data Center Deployment on AIX or Linux Administration Guide* for more information.
2. Configure a mode. Specify LOCMODE as the mode name. Leave the default values for the other parameters.
3. Configure a logical unit (LU) on the source computer for **asample1** (the invoking TP). Set both the LU name and LU alias to **TPLU1** (the LU alias specified in the **asample1** program). Leave the default values for the other parameters.
4. If you are running the TPs on separate computers, configure a partner LU alias on the source computer to identify the target LU. Set the partner LU name to *netname***.TPLU2**, where *netname* is the SNA network name of the target computer.
5. Configure an LU on the target computer for the invoked TP. Set both the LU name and LU alias to **TPLU2** (the alias by which the **asample1** program refers to the LU serving **asample2**). Leave the default values for the other parameters.
6. Configure the invoked TP in the Communications Server invokable TP data file on the target computer. Refer to the *IBM Communications Server for Data Center Deployment on AIX or Linux Administration Guide* for more information.
   - For the *TP name* parameter, specify **TPNAME2** (the name specified by the invoking TP).
   - For *Full path to TP executable*, enter the full path name of the executable file **asample2**.
   - For the *User ID* parameter, specify your AIX / Linux user ID on the target computer.
   - Leave the default values for other parameters.
7. If the invoked TP is to run with a *user_id* of `root`, change the permissions on the executable file to allow it to do so. Use the following command:

   chmod +s asample2
8. Start the invoking program, **asample1**. This program requires one parameter, the full path name (on the target computer) of the file to be displayed. For example:

   asample1 /usr/john/myfile.text
9. Enter **F** or **B** to display blocks of the requested file.
10. Use **Q** to end program 1; program 2 will end as well.

Appendix A. Return Code Values

This appendix lists all the possible return codes in the APPC interface in numerical order. The values are defined in the header file **values_c.h**(for AIX / Linux) or **winappc.h** (for Windows).

You can use this appendix as a reference to check the meaning of a return code received by your application.

Primary Return Codes

The following primary return codes are used in APPC applications.

AP_OK	0x0000
AP_PARAMETER_CHECK	0x0100
AP_STATE_CHECK	0x0200
AP_INDICATION	0x0210
AP_TP_BUSY	0x02F0
AP_ALLOCATION_ERROR	0x0300
AP_ACTIVATION_FAIL_RETRY	0x0310
AP_COMM_SUBSYSTEM_ABENDED	0x03F0
AP_ACTIVATION_FAIL_NO_RETRY	0x0410
AP_COMM_SUBSYSTEM_NOT_LOADED	0x04F0
AP_DEALLOC_ABEND	0x0500
AP_LU_SESS_LIMIT_EXCEEDED	0x0510
AP_DEALLOC_ABEND_PROG	0x0600
AP_FUNCTION_NOT_SUPPORTED	0x0610
AP_THREAD_BLOCKING	0x06F0
AP_DEALLOC_ABEND_SVC	0x0700
AP_DEALLOC_ABEND_TIMER	0x0800
AP_DATA_POSTING_BLOCKED	0x0810
AP_INVALID_VERB_SEGMENT	0x08F0
AP_DEALLOC_NORMAL	0x0900
AP_PATH_SWITCH_NOT_ALLOWED	0x0910
AP_CP_CP_SESS_ACT_FAILURE	0x0A10
AP_PROG_ERROR_NO_TRUNC	0x0C00
AP_PROG_ERROR_TRUNC	0x0D00
AP_PROG_ERROR_PURGING	0x0E00
AP_CONV_FAILURE_RETRY	0x0F00
AP_CONV_FAILURE_NO_RETRY	0x1000
AP_SVC_ERROR_NO_TRUNC	0x1100
AP_UNEXPECTED_DOS_ERROR	0x11F0
AP_SVC_ERROR_TRUNC	0x1200
AP_SVC_ERROR_PURGING	0x1300
AP_UNSUCCESSFUL	0x1400
AP_STACK_TOO_SMALL	0x15F0
AP_MIXED_API_USED	0x16F0
AP_IN_PROGRESS	0x17F0
AP_CNOS_PARTNER_LU_REJECT	0x1800
AP_COMPLETED	0x18F0
AP_CONVERSATION_TYPE_MIXED	0x1900
AP_NODE_STOPPING	0x1A00
AP_NODE_NOT_STARTED	0x1B00
AP_CANCELLED	0x2100
AP_BACKED_OUT	0x2200
AP_DUPLEX_TYPE_MIXED	0x2300
AP_LS_FAILURE	0x2300
AP_OPERATION_INCOMPLETE	0x4000
AP_OPERATION_NOT_ACCEPTED	0x4100
AP_CONVERSATION_ENDED	0x4200
AP_ERROR_INDICATION	0x4300
AP_EXPD_NOT_SUPPORTED_BY_LU	0x4400

AP_BUFFER_TOO_SMALL	0x4500
AP_MEMORY_ALLOCATION_FAILURE	0x4600
AP_INVALID_VERB	0xFFFF

Secondary Return Codes

The following secondary return codes are used in APPC applications.

AP_AS_SPECIFIED	0x00000000
AP_ALLOCATION_ERROR_PENDING	0x00000300
AP_DEALLOC_ABEND_PROG_PENDING	0x00000600
AP_DEALLOC_ABEND_SVC_PENDING	0x00000700
AP_DEALLOC_ABEND_TIMER_PENDING	0x00000800
AP_UNKNOWN_ERROR_TYPE_PENDING	0x00001100
AP_BO_NO_RESYNC	0x00002408
AP_TRANS_PGM_NOT_AVAIL_NO_RETRY	0x00004C08
AP_INVALID_SET_PROT	0x00070000
AP_INVALID_DLUS_NAME	0x00900000
AP_SEC_BAD_PASSWORD_EXPIRED	0x00FF0F08
AP_BAD_TP_ID	0x01000000
AP_BO_RESYNC	0x01002408
AP_INVALID_NEW_PROT	0x01070000
AP_DLC_ACTIVE	0x01100000
AP_NO_DEFAULT_DLUS_DEFINED	0x01900000
AP_BAD_TPSID	0x01FF0000
AP_SEC_BAD_PASSWORD_INVALID	0x01FF0F08
AP_BAD_CONV_ID	0x02000000
AP_SEND_ERROR_LOG_LL_WRONG	0x02010000
AP_INVALID_SET_UNPROT	0x02070000
AP_INVALID_NUMBER_OF_NODE_ROWS	0x02080000
AP_DUPLICATE_CP_NAME	0x02100000
AP_INVALID_PU_ID	0x02900000
AP_NOT_OWNER	0x02FF0000
AP_SEC_BAD_USERID_REVOKED	0x02FF0F08
AP_BAD_LU_ALIAS	0x03000000
AP_BAD_DLOAD_ID	0x03000001
AP_BAD_REMOTE_LU_ALIAS	0x03000002
AP_SEND_ERROR_BAD_TYPE	0x03010000
AP_INVALID_NEW_UNPROT	0x03070000
AP_DUPLICATE_DEST_ADDR	0x03100000
AP_PU_ALREADY_ACTIVATING	0x03900000
AP_INSUFFICIENT_PRIVILEGES	0x03FF0000
AP_SEC_BAD_USERID_INVALID	0x03FF0F08
AP_ALLOCATION_FAILURE_NO_RETRY	0x04000000
AP_SEND_ERROR_BAD_STATE	0x04010000
AP_INVALID_SET_USER	0x04070000
AP_NODE_ROW_WGT_LESS_THAN_LAST	0x04080000
AP_CANT_MODIFY_PORT_NAME	0x04100000
AP_PU_ALREADY_DEACTIVATING	0x04900000
AP_INVALID_CALLBACK	0x04FF0000
AP_SEC_BAD_USERID_MISSING	0x04FF0F08
AP_ALLOCATION_FAILURE_RETRY	0x05000000
AP_BAD_ERROR_DIRECTION	0x05010000
AP_INVALID_DATA_TYPE	0x05070000
AP_TG_ROW_WGT_LESS_THAN_LAST	0x05080000
AP_DUPLICATE_PORT_NUMBER	0x05100000
AP_PU_ALREADY_ACTIVE	0x05900000
AP_BAD_TP_TYPE	0x05FF0000
AP_SEC_BAD_PASSWORD_MISSING	0x05FF0F08
AP_INVALID_STATS_TYPE	0x06070000
AP_DUPLICATE_PORT_NAME	0x06100000
AP_PU_NOT_ACTIVE	0x06900000
AP_ALREADY_REGISTERED	0x06FF0000
AP_SEC_BAD_GROUP_INVALID	0x06FF0F08
AP_AS_NEGOTIATED	0x07000000
AP_INVALID_TABLE_TYPE	0x07070000
AP_INVALID_DLC_NAME	0x07100000

AP_DLUS_REJECTED	0x07900000
AP_SEC_BAD_UID_REVOKED_IN_GRP	0x07FF0F08
AP_PORT_DEACTIVATED	0x08070000
AP_INVALID_DLC_TYPE	0x08100000
AP_DLUS_CAPS_MISMATCH	0x08900000
AP_SEC_BAD_UID_NOT_DEFD_TO_GRP	0x08FF0F08
AP_ALLOCATE_NOT_PENDING	0x09050000
AP_INVALID_SET_PASSWORD	0x09070000
AP_INVALID_NUMBER_OF_TG_ROWS	0x09080000
AP_INVALID_LINK_ACTIVE_LIMIT	0x09100000
AP_PU_FAILED_ACTPU	0x09900000
AP_SEC_BAD_UNAUTHRZD_AT_RLU	0x09FF0F08
AP_SNA_DEFD_COS_CANT_BE_CHANGE	0x0A080000
AP_SNA_DEFD_COS_CANT_BE_CHANGED	0x0A080000
AP_PU_NOT_RESET	0x0A900000
AP_SEC_BAD_UNAUTHRZD_FROM_LLU	0x0AFF0F08
AP_INVALID_NUM_PORTS_SPECIFIED	0x0B100000
AP_PU_OWNS_LUS	0x0B900000
AP_SEC_BAD_UNAUTHRZD_TO_TP	0x0BFF0F08
AP_INVALID_PORT_NAME	0x0C100000
AP_INVALID_FILTER_OPTION	0x0C900000
AP_SEC_BAD_INSTALL_EXIT_FAILED	0x0CFF0F08
AP_INVALID_PORT_TYPE	0x0D100000
AP_INVALID_STOP_TYPE	0x0D900000
AP_SEC_BAD_PROCESSING_FAILURE	0x0DFF0F08
AP_UNRECOGNIZED_DEACT_TYPE	0x0E050000
AP_PORT_ACTIVE	0x0E100000
AP_PU_ALREADY_DEFINED	0x0E900000
AP_NO_PORTS_DEFINED_ON_DLC	0x0F100000
AP_DEPENDENT_LU_NOT_SUPPORTED	0x0F900000
AP_INVALID_DLC	0x10050000
AP_COS_NAME_NOT_DEFD	0x10080000
AP_DUPLICATE_PORT	0x10100000
AP_INVALID_DSPU_SERVICES	0x10900000
AP_BAD_CONV_TYPE	0x11000000
AP_SNA_DEFD_COS_CANT_BE_DELETE	0x11080000
AP_SNA_DEFD_COS_CANT_BE_DELETED	0x11080000
AP_STOP_PORT_PENDING	0x11100000
AP_DSPU_SERVICES_NOT_SUPPORTED	0x11900000
AP_BAD_SYNC_LEVEL	0x12000000
AP_LU_NAU_ADDR_ALREADY_DEFD	0x12020000
AP_INVALID_SESSION_ID	0x12050000
AP_LINK_DEACT_IN_PROGRESS	0x12100000
AP_INVALID_DSPU_NAME	0x12900000
AP_BAD_SECURITY	0x13000000
AP_INVALID_NN_SESSION_TYPE	0x13050000
AP_LINK_DEACTIVATED	0x13100000
AP_PARTNER_NOT_FOUND	0x13200000
AP_PARTNER_NOT_RESPONDING	0x13300000
AP_ERROR	0x13400000
AP_DSPU_ALREADY_DEFINED	0x13900000
AP_BAD_RETURN_CONTROL	0x14000000
AP_INVALID_MAX_NEGOT_SESS_LIM	0x14020000
AP_INVALID_SET_COLLECT_STATS	0x14050000
AP_LINK_ACT_BY_REMOTE	0x14100000
AP_INVALID_SOLICIT_SSCP_SESS	0x14900000
AP_INVALID_BACK_LEVEL_SUPPORT	0x15000000
AP_INVALID_MODE_NAME	0x15020000
AP_INVALID_SET_COLLECT_NAMES	0x15050000
AP_LINK_ACT_BY_LOCAL	0x15100000
AP_INVALID_TG_NUMBER	0x15500000
AP_MISSING_CP_NAME	0x15510000
AP_MISSING_CP_TYPE	0x15520000
AP_INVALID_CP_TYPE	0x15520000
AP_DUPLICATE_TG_NUMBER	0x15530000
AP_TG_NUMBER_IN_USE	0x15540000
AP_MISSING_TG_NUMBER	0x15550000

AP_PARALLEL_TGS_NOT_ALLOWED	0x15570000
AP_INVALID_BKUP_DLUS_NAME	0x15900000
AP_PIP_LEN_INCORRECT	0x16000000
AP_INVALID_RECV_PACING_WINDOW	0x16020000
AP_INVALID_SET_COLLECT_RSCVS	0x16050000
AP_SEC_REQUESTED_NOT_SUPPORTED	0x16900000
AP_NO_USE_OF_SNASVCMG	0x17000000
AP_INVALID_CNOS_SLIM	0x17020000
AP_LINK_NOT_DEFD	0x17100000
AP_INVALID_DUPLEX_SUPPORT	0x17900000
AP_UNKNOWN_PARTNER_MODE	0x18000000
AP_INVALID_TARGET_PACING_CNT	0x18020000
AP_PS_CREATION_FAILURE	0x18100000
AP_QUEUE_PROHIBITED	0x18900000
AP_INVALID_MAX_RU_SIZE_UPPER	0x19020000
AP_TP_ACTIVE	0x19100000
AP_INVALID_TEMPLATE_NAME	0x19900000
AP_INVALID_SNASVCMG_MODE_LIMIT	0x1A020000
AP_MODE_ACTIVE	0x1A100000
AP_CLASHING_NAU_RANGE	0x1A900000
AP_PLU_ACTIVE	0x1B100000
AP_INVALID_NAU_RANGE	0x1B900000
AP_INVALID_COS_SNASVCMG_MODE	0x1C020000
AP_INVALID_PLU_NAME	0x1C100000
AP_INVALID_NUM_DSLU_TEMPLATES	0x1C900000
AP_INVALID_DEFAULT_RU_SIZE	0x1D020000
AP_INVALID_SET_NEGOTIABLE	0x1D100000
AP_GLOBAL_TIMEOUT_NOT_DEFINED	0x1D900000
AP_INVALID_MIN_CONWINNERS	0x1E020000
AP_INVALID_MODE_NAME_SELECT	0x1E100000
AP_INVALID_RESOURCE_NAME	0x1E900000
AP_INVALID_RESPONSIBLE	0x1F100000
AP_INVALID_DLUS_RETRY_TIMEOUT	0x1F900000
AP_MODE_SESS_LIM_EXCEEDS_NEG	0x20020000
AP_INVALID_DRAIN_SOURCE	0x20100000
AP_INVALID_DLUS_RETRY_LIMIT	0x20900000
AP_CPSVCMG_ALREADY_DEFD	0x21020000
AP_INVALID_CN_NAME	0x21080000
AP_INVALID_DRAIN_TARGET	0x21100000
AP_TP_NAME_NOT_RECOGNIZED	0x21600810
AP_INVALID_MIN_CONLOSERS	0x21900000
AP_BAD_DUPLEX_TYPE	0x22000000
AP_INVALID_BYPASS_SECURITY	0x22020000
AP_DEF_LINK_INVALID_SECURITY	0x22080000
AP_INVALID_FORCE	0x22100000
AP_SYSTEM_TP_CANT_BE_CHANGED	0x22600810
AP_INVALID_MAX_RU_SIZE_LOW	0x22900000
AP_FDX_NOT_SUPPORTED_BY_LU	0x23000000
AP_TEST_INVALID_FOR_FDX	0x23010000
AP_INVALID_IMPLICIT_PLU_FORBID	0x23020000
AP_INVALID_PROPAGATION_DELAY	0x23080000
AP_SYSTEM_TP_CANT_BE_DELETED	0x23600810
AP_INVALID_MAX_RECV_PACING_WIN	0x23900000
AP_SEND_EXPD_INVALID_LENGTH	0x24010000
AP_INVALID_SPECIFIC_SECURITY	0x24020000
AP_INVALID_EFFECTIVE_CAPACITY	0x24080000
AP_INVALID_CLEANUP_TYPE	0x24100000
AP_INVALID_DYNAMIC_LOAD	0x24600810
AP_RU_SIZE_LOW_UPPER_MISMATCH	0x24900000
AP_RCV_EXPD_INVALID_LENGTH	0x25010000
AP_INVALID_DELAYED_LOGON	0x25020000
AP_INVALID_COS_NAME	0x25100000
AP_INVALID_ENABLED	0x25600810
AP_LU_ALREADY_ACTIVATING	0x25900000
AP_EXPD_BAD_RETURN_CONTROL	0x26010000
AP_INVALID_CNOS_PERMITTED	0x26020000
AP_PW_SUB_NOT_SUPP_ON_SESS	0x26050000

AP_INVALID_SESSION_LIMIT	0x26100000
AP_INVALID_PIP_ALLOWED	0x26600810
AP_LU_DEACTIVATING	0x26900000
AP_EXPD_DATA_BAD_CONV_STATE	0x27010000
AP_INVALID_DRAIN	0x27100000
AP_LU_ALREADY_ACTIVE	0x27900000
AP_INVALID_PRLL_SESS_SUPP	0x28100000
AP_INVALID_MIN_CONTENTION_SUM	0x28900000
AP_INVALID_LU_NAME	0x29100000
AP_COMPRESSION_NOT_SUPPORTED	0x29900000
AP_MODE_NOT_RESET	0x2A100000
AP_INVALID_MAX_COMPRESS_LVL	0x2A900000
AP_MODE_RESET	0x2B100000
AP_INVALID_COMPRESSION	0x2B900000
AP_CNOS_REJECT	0x2C100000
AP_INVALID_EXCEPTION_INDEX	0x2C900000
AP_INVALID_OP_CODE	0x2D100000
AP_INVALID_MAX_LS_EXCEPTION	0x2D900000
AP_INVALID_DISABLE	0x2E900000
AP_INVALID_MODIFY_TEMPLATE	0x2F900000
AP_INVALID_ALLOW_TIMEOUT	0x30900000
AP_CONFIRM_ON_SYNC_LEVEL_NONE	0x31000000
AP_PIP_NOT_ALLOWED	0x31600810
AP_TRANS_PGM_NOT_AVAIL_RETRY	0x31604B08
AP_POST_ON_RECEIPT_BAD_FILL	0x31900000
AP_CONFIRM_BAD_STATE	0x32000000
AP_UNKNOWN_USER	0x32100000
AP_POST_ON_RECEIPT_BAD_STATE	0x32900000
AP_CONFIRM_NOT_LL_BDY	0x33000000
AP_NO_PROFILES	0x33100000
AP_INVALID_HPR_SUPPORT	0x33900000
AP_CONFIRM_INVALID_FOR_FDX	0x34000000
AP_CONVERSATION_TYPE_MISMATCH	0x34600810
AP_INVALID_LU_MODEL	0x34900000
AP_INVALID_MODEL_NAME	0x35900000
AP_TOO_MANY_PROFILES	0x36100000
AP_INVALID_CRYPTOGRAPHY	0x36900000
AP_INVALID_UPDATE_TYPE	0x37100000
AP_INVALID_CLU_CRYPTOGRAPHY	0x37900000
AP_DIR_ENTRY_PARENT	0x38100000
AP_INVALID_RESOURCE_TYPES	0x38900000
AP_NODE_ALREADY_STARTED	0x39100000
AP_CHECKSUM_FAILED	0x39900000
AP_NODE_FAILED_TO_START	0x3A100000
AP_DATA_CORRUPT	0x3A900000
AP_LU_ALREADY_DEFINED	0x3B100000
AP_INVALID_RETRY_FLAGS	0x3B900000
AP_IMPLICIT_LU_DEFINED	0x3C100000
AP_DELAYED_VERB_PENDING	0x3C900000
AP_PORT_INACTIVE	0x3D100000
AP_DSLU_ACTIVE	0x3D900000
AP_ACTIVATION_LIMITS_REACHED	0x3E100000
AP_ACTIVATION_LIMITS_REACHED	0x3E100000
AP_INVALID_BRANCH_LINK_TYPE	0x3E900000
AP_PARALLEL_TGS_NOT_SUPPORTED	0x3F100000
AP_INVALID_BRNN_SUPPORT	0x3F900000
AP_DLC_INACTIVE	0x40100000
AP_BRNN_SUPPORT_MISSING	0x40900000
AP_CONFIRMED_BAD_STATE	0x41000000
AP_NO_LINKS_DEFINED	0x41100000
AP_SYNC_LEVEL_NOT_SUPPORTED	0x41600810
AP_INVALID_UPLINK	0x41900000
AP_CONFIRMED_INVALID_FOR_FDX	0x42000000
AP_STOP_DLC_PENDING	0x42100000
AP_INVALID_DOWNLINK	0x42900000
AP_INVALID_LS_ROLE	0x43100000
AP_INVALID_IMPLICIT_UPLINK	0x43900000

AP_INVALID_BTU_SIZE	0x44100000
AP_INVALID_ROCP_NAME	0x44900000
AP_LAST_LINK_ON_ACTIVE_PORT	0x45100000
AP_INVALID_REG_WITH_NN	0x45900000
AP_DYNAMIC_LOAD_ALREADY_REGD	0x46100000
AP_LS_PENDING_RETRY	0x46900000
AP_INVALID_LIST_OPTION	0x47100000
AP_INVALID_COS_TABLE_VERSION	0x47900000
AP_INVALID_RES_NAME	0x48100000
AP_CFRTP_REQUIRED_FOR_MLTG	0x48900000
AP_INVALID_RES_TYPE	0x49100000
AP_INVALID_MLTG_PAC_ALGORITHM	0x49900000
AP_INVALID_ADJ_NNCP_NAME	0x4A100000
AP_LIM_RESRCE_INVALID_FOR_MLTG	0x4A900000
AP_INVALID_NODE	0x4B100000
AP_AUTO_ACT_INVALID_FOR_MLTG	0x4B900000
AP_INVALID_ORIGIN_NODE	0x4C100000
AP_MLTG_LS_VISIBILITY_MISMATCH	0x4C900000
AP_INVALID_TG	0x4D100000
AP_SLTG_LINK_ACTIVE	0x4D900000
AP_INVALID_FQPCID	0x4E100000
AP_MLTG_LINK_PROPERTIES_DIFFER	0x4E900000
AP_INVALID_POOL_NAME	0x4F100000
AP_INVALID_ADJ_CP_NAME	0x4F900000
AP_BAD_TYPE	0x50020000
AP_INVALID_NAU_ADDRESS	0x50100000
AP_INVALID_ENABLE_POOL	0x50300000
AP_INVALID_SEND_TERM_SELF	0x50900000
AP_DEALLOC_BAD_TYPE	0x51000000
AP_LU_NAME_POOL_NAME_CLASH	0x51100000
AP_SECURITY_NOT_VALID	0x51600F08
AP_INVALID_TERM_METHOD	0x51900000
AP_DEALLOC_FLUSH_BAD_STATE	0x52000000
AP_INVALID_PRIORITY	0x52100000
AP_INVALID_DISABLE_BRANCH_AWRN	0x52900000
AP_DEALLOC_CONFIRM_BAD_STATE	0x53000000
AP_INVALID_DNST_LU_NAME	0x53100000
AP_INVALID_SHARING_PROHIBITED	0x53900000
AP_INVALID_HOST_LU_NAME	0x54100000
AP_INVALID_LINK_SPEC_FORMAT	0x54900000
AP_DEALLOC_NOT_LL_BDY	0x55000000
AP_PU_NOT_DEFINED	0x55100000
AP_INVALID_CN_TYPE	0x55900000
AP_INVALID_PU_NAME	0x56100000
AP_INVALID_PU_TYPE	0x56600000
AP_INCONSISTENT_BEST_EFFORT	0x56900000
AP_DEALLOC_LOG_LL_WRONG	0x57000000
AP_CNOS_MODE_NAME_REJECT	0x57010000
AP_INVALID_MAX_IFRM_RCVD	0x57100000
AP_INVALID_CN_TG	0x57900000
AP_INVALID_SYM_DEST_NAME	0x58100000
AP_SEC_BAD_PROTOCOL_VIOLATION	0x58600F08
AP_INVALID_LINK_SPEC_DATA	0x58900000
AP_INVALID_LENGTH	0x59100000
AP_DLC_UI_ONLY	0x59900000
AP_INVALID_ISR_THRESHOLDS	0x5A100000
AP_ADJ_CP_WRONG_TYPE	0x5A900000
AP_BAD_PARTNER_LU_ALIAS	0x5B010000
AP_INVALID_NUM_LUS	0x5B100000
AP_CP_CP_SESS_ALREADY_ACTIVE	0x5B900000
AP_EXCEEDS_MAX_ALLOWED	0x5C010000
AP_CANT_DELETE_ADJ_ENDNODE	0x5C100000
AP_NO_ACTIVE_CP_CP_LINK	0x5C900000
AP_LU_MODE_SESSION_LIMIT_ZERO	0x5D010000
AP_INVALID_RESOURCE_TYPE	0x5D100000
AP_PU_CONC_NOT_SUPPORTED	0x5E100000
AP_INVALID_IMPL_APPN_LINKS_LEN	0x5E900000

AP_CNOS_COMMAND_RACE_REJECT	0x5F010000
AP_DLUR_NOT_SUPPORTED	0x5F100000
AP_INVALID_LIMIT_ENABLE	0x5F900000
AP_INVALID_SVCMG_LIMITS	0x60010000
AP_INVALID_RTP_CONNECTION	0x60100000
AP_INVALID_LS_ATTRIBUTE	0x60900000
AP_FLUSH_NOT_SEND_STATE	0x61000000
AP_PATH_SWITCH_IN_PROGRESS	0x61100000
AP_HPR_NOT_SUPPORTED	0x62100000
AP_SOME_ENABLED	0x62900000
AP_RTP_NOT_SUPPORTED	0x63100000
AP_NONE_ENABLED	0x63900000
AP_COS_TABLE_FULL	0x64100000
AP_INCONSISTENT_IMPLICIT	0x64900000
AP_INVALID_DAYS_LEFT	0x65100000
AP_INVALID_PREFER_ACTIVE_DLUS	0x65900000
AP_ANYNET_NOT_SUPPORTED	0x66100000
AP_INVALID_PERSIST_PIPE_SUPP	0x66900000
AP_INVALID_DISCOVERY_SUPPORT	0x67100000
AP_ACTIVATION_PROHIBITED	0x67900000
AP_SESSION_FAIL_ALREADY_REGD	0x68100000
AP_INVALID_NULL_ADDR_MEANING	0x68900000
AP_CANT_MODIFY_VISIBILITY	0x69100000
AP_INVALID_CPLU_SYNCPT_SUPPORT	0x69900000
AP_CANT_MODIFY_WHEN_ACTIVE	0x6A100000
AP_INVALID_CPLU_ATTRIBUTES	0x6A900000
AP_INVALID_BASE_NUMBER	0x6B100000
AP_INVALID_REG_LEN_SUPPORT	0x6B900000
AP_DEACT_CG_INVALID_CGID	0x6C020000
AP_INVALID_NAME_ATTRIBUTES	0x6C100000
AP_LUNAME_CGID_MISMATCH	0x6C900000
AP_NAU_ADDRESS_MISMATCH	0x6D100000
AP_INVALID_DDDLU_OFFLINE	0x6D900000
AP_POSTED_DATA	0x6E100000
AP_POSTED_NO_DATA	0x6F100000
AP_DEF_PLU_INVALID_FQ_NAME	0x74020000
AP_DLC_DEACTIVATING	0x86020000
AP_INVALID_WILDCARD_NAME	0x8C020000
AP_DUPLICATE	0x8D020000
AP_LU_NAME_WILDCARD_NAME_CLASH	0x8E020000
AP_INVALID_USERID	0x90020000
AP_INVALID_PASSWORD	0x91020000
AP_INVALID_PROFILE	0x93020000
AP_INVALID_TP_NAME	0xA0020000
AP_P_TO_R_INVALID_TYPE	0xA1000000
AP_INVALID_CONV_TYPE	0xA1020000
AP_P_TO_R_NOT_LL_BDY	0xA2000000
AP_P_TO_R_NOT_SEND_STATE	0xA3000000
AP_INVALID_SYNC_LEVEL	0xA3020000
AP_P_TO_R_INVALID_FOR_FDX	0xA5000000
AP_INVALID_LINK_NAME_SPECIFIED	0xB0020000
AP_RCV_AND_WAIT_BAD_STATE	0xB1000000
AP_INVALID_LU_ALIAS	0xB1020000
AP_RCV_AND_WAIT_NOT_LL_BDY	0xB2000000
AP_INVALID_NUM_LS_SPECIFIED	0xB2020000
AP_PLU_ALIAS_CANT_BE_CHANGED	0xB3020000
AP_PLU_ALIAS_ALREADY_USED	0xB4020000
AP_RCV_AND_WAIT_BAD_FILL	0xB5000000
AP_INVALID_AUTO_ACT_SUPP	0xB5020000
AP_CANT_DELETE_IMPLICIT_LU	0xB6020000
AP_FORCED	0xB7020000
AP_INVALID_LS_NAME	0xB7030000
AP_INVALID_LFSID_SPECIFIED	0xB7040000
AP_INVALID_FILTER_TYPE	0xB7050000
AP_INVALID_MESSAGE_TYPE	0xB7060000
AP_CANT_DELETE_CP_LU	0xB7070000
AP_ALL_RESOURCES_NOT_DEFINED	0xB7090000

AP_INVALID_LIST_TYPE	0xB70A0000
AP_RESOURCE_NAME_NOT_ALLOWED	0xB70B0000
AP_LU_ALIAS_CANT_BE_CHANGED	0xB8020000
AP_LU_ALIAS_ALREADY_USED	0xB9020000
AP_INVALID_LINK_ENABLE	0xBA020000
AP_INVALID_CLU_COMPRESSION	0xBB020000
AP_INVALID_DLUR_SUPPORT	0xBC020000
AP_ALREADY_STARTING	0xC0010000
AP_RCV_IMMD_BAD_STATE	0xC1000000
AP_INVALID_LINK_NAME	0xC1010000
AP_INVALID_USER_DEF_1	0xC3010000
AP_RCV_IMMD_BAD_FILL	0xC4000000
AP_INVALID_USER_DEF_2	0xC4010000
AP_INVALID_NODE_TYPE	0xC4020000
AP_INVALID_USER_DEF_3	0xC5010000
AP_INVALID_NAME_LEN	0xC5020000
AP_INVALID_NETID_LEN	0xC6020000
AP_INVALID_NODE_TYPE_FOR_HPR	0xC8020000
AP_INVALID_MAX_DECOMPRESS_LVL	0xC9020000
AP_INVALID_CP_NAME	0xCA010000
AP_INVALID_COMP_IN_SERIES	0xCA020000
AP_INVALID_LIMITED_RESOURCE	0xCE010000
AP_RCV_AND_POST_BAD_STATE	0xD1000000
AP_INVALID_BYTE_COST	0xD1010000
AP_RCV_AND_POST_NOT_LL_BDY	0xD2000000
AP_RCV_AND_POST_BAD_FILL	0xD5000000
AP_INVALID_TIME_COST	0xD6010000
AP_BAD_RETURN_STATUS_WITH_DATA	0xD7000000
AP_LOCAL_CP_NAME	0xD7010000
AP_LS_ACTIVE	0xDA010000
AP_INVALID_FQ_OWNING_CP_NAME	0xDB020000
AP_R_T_S_BAD_STATE	0xE1000000
AP_R_T_S_INVALID_FOR_FDX	0xE2000000
AP_BAD_LL	0xF1000000
AP_SEND_DATA_NOT_SEND_STATE	0xF2000000
AP_CP_OR_SNA_SVCMG_UNDELETABLE	0xF3010000
AP_SEND_DATA_INVALID_TYPE	0xF4000000
AP_DEL_MODE_DEFAULT_SPCD	0xF4010000
AP_SEND_DATA_CONFIRM_SYNC_NONE	0xF5000000
AP_MODE_NAME_NOT_DEFD	0xF5010000
AP_SEND_DATA_NOT_LL_BDY	0xF6000000
AP_MODE_UNDELETABLE	0xF6010000
AP_SEND_TYPE_INVALID_FOR_FDX	0xF7000000
AP_INVALID_FQ_LU_NAME	0xFD010000
AP_INVALID_PARTNER_LU	0xFE010000
AP_INVALID_LOCAL_LU	0xFF010000

Appendix B. Common Return Codes

This appendix describes the primary return codes (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.

Common return codes are described in the following sections.

AP_ALLOCATION_ERROR

The primary and secondary return codes are:

primary_rc

AP_ALLOCATION_ERROR
APPC has failed to allocate a conversation. The conversation state is set to RESET. This code may be returned through a verb issued after [MC_]ALLOCATE.

secondary_rc
Possible values are:

AP_ALLOCATION_FAILURE_NO_RETRY
The conversation cannot be allocated because of a permanent condition, such as a configuration error or session protocol error. 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 allow the condition to clear.

AP_CONVERSATION_TYPE_MISMATCH
The partner LU or TP does not support the conversation type (basic or mapped) specified in the allocation request.

AP_PIP_NOT_ALLOWED
The allocation request specified PIP data, but the partner TP did not accept it. This may be because the partner TP does not require this data, because it is using an APPC implementation which does not support receiving PIP data, or because the partner is a CPI-C application (CPI-C does not support PIP data).

AP_PIP_NOT_SPECIFIED_CORRECTLY
The partner TP requires PIP data; but the allocation request specified either no PIP data or an incorrect number of parameters.

AP_SECURITY_NOT_VALID
The user ID or password specified in the allocation request was not accepted by the partner LU.

AP_SYNC_LEVEL_NOT_SUPPORTED
The partner TP does not support the *sync_level* (AP_NONE,

AP_CONFIRM_SYNC_LEVEL, or AP_SYNCPT) specified in the allocation request, or the *sync_level* was not recognized.

AP_TP_NAME_NOT_RECOGNIZED
The partner LU does not recognize the TP name specified in the allocation request.

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

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

AP_SEC_BAD_PROTOCOL_VIOLATION
The remote LU rejected the allocation request due to a protocol violation.

AP_SEC_BAD_PASSWORD_EXPIRED
The remote LU rejected the allocation request because the password provided is no longer valid.

AP_SEC_BAD_PASSWORD_INVALID
The remote LU rejected the allocation request because the password is not valid.

AP_SEC_BAD_USERID_REVOKED
The remote LU rejected the allocation request because the user ID is no longer valid.

AP_SEC_BAD_USERID_INVALID
The remote LU rejected the allocation request because the user ID is not valid.

AP_SEC_BAD_USERID_MISSING
The remote LU rejected the allocation request because a user ID was not specified but is required.

AP_SEC_BAD_PASSWORD_MISSING
The remote LU rejected the allocation request because a password was not specified but is required.

AP_SEC_BAD_GROUP_INVALID
The remote LU rejected the allocation request because the group is not valid.

AP_SEC_BAD_UID_REVOKED_IN_GRP
The remote LU rejected the allocation request because the user ID is no longer in the group.

AP_SEC_BAD_UID_NOT_DEFD_TO_GRP
The remote LU rejected the allocation request because the user ID is not in the group.

AP_SEC_BAD_UNAUTHRZD_AT_RLU
The remote LU rejected the allocation request because the user ID is not authorized to start this TP at the remote LU.

AP_SEC_BAD_UNAUTHRZD_FROM_LLU
The remote LU rejected the allocation request because the user ID is not authorized to start this TP from the local LU.

AP_SEC_BAD_UNAUTHRZD_TO_TP
The remote LU rejected the allocation request because the user ID is not authorized to start this TP.

AP_SEC_BAD_INSTALL_EXIT_FAILED
The remote LU rejected the allocation request because it failed to install a required exit.

AP_SEC_BAD_PROCESSING_FAILURE
The remote LU rejected the allocation request because of a processing failure at the remote LU.

Because providing detailed information about security failures is a potential security flaw, it is possible to turn off support for these AP_SEC_BAD_* return codes. If this is done, all of these errors are reported to the application as AP_SECURITY_NOT_VALID. See the **define_defaults** command in the *IBM Communications Server for Data Center Deployment on AIX or Linux Administration Command Reference* and DEFINE_DEFAULTS NOF verb in the *IBM Communications Server for Data Center Deployment on AIX or Linux NOF Programmer's Guide* for details.

AP_BACKED_OUT

UNIX

The primary and secondary return codes are:

primary_rc

AP_BACKED_OUT
The partner TP (or another TP participating in the same logical unit of work) has issued a backout request. The Syncpoint Manager is responsible for performing the appropriate Syncpoint processing based on the secondary return code, which is one of the following:

secondary_rc
Possible values are:

AP_BO_NO_RESYNC
The partner TP has completed backing out its resources.

AP_BO_RESYNC
A failure occurred while the partner TP was attempting to back out its resources; resynchronization is still in progress.

AP_CANCELLED

WINDOWS

The primary return code is:

primary_rc

AP_CANCELLED
The verb was issued using the `WinAsyncAPPC` entry point, and was then canceled using the `WinAPPCCancel` entry point. For more information about these entry points, see "APPC Entry Points: Windows Systems" on page 36.

A secondary return code is not returned.

AP_COMM_SUBSYSTEM_ABENDED

The primary return code is:

primary_rc

AP_COMM_SUBSYSTEM_ABENDED
The return code indicates that the Communications Server software has ended abnormally, or that there is a problem with the LAN. The System Administrator should examine the error log to determine the reason for the abend.

A secondary return code is not returned.

AP_COMM_SUBSYSTEM_NOT_LOADED

The primary return code is:

primary_rc

AP_COMM_SUBSYSTEM_NOT_LOADED
This return code indicates that an attempt to start a TP using the TP_STARTED or RECEIVE_ALLOCATE verb cannot be accepted because of one of the following conditions.

UNIX

- The Communications Server software has not been loaded, or the local node that owns the LU used by this TP is not started. Contact the System Administrator for corrective action.
- The maximum number of users permitted by the Communications Server license are already using Communications Server. You cannot start this TP at present because it would exceed the user limit; you may be able to start it later when there are fewer users on the system.

WINDOWS

- The Communications Server software has not been loaded, or a communications component used by the APPC LU you have specified is inactive. Contact the System Administrator for corrective action.
- The LU alias specified on a TP_STARTED verb was not recognized. Check that the LU alias specified matches an APPC local LU alias in the configuration file.
- The maximum number of Communications Server users permitted by the Communications Server licence are already using the local node that owns the APPC local LU you are using. You cannot start this TP at present because it would exceed the user limit; you may be able to start it later when there are fewer users on the system.

A secondary return code is not returned.

AP_CONV_FAILURE_NO_RETRY

The primary return code is:

primary_rc

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.

A secondary return code is not returned.

AP_CONV_FAILURE_RETRY

The primary return code is:

primary_rc

AP_CONV_FAILURE_RETRY
The conversation was terminated because of a temporary error. Restart the TP 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.

A secondary return code is not returned.

AP_CONVERSATION_TYPE_MIXED

The primary return code is:

primary_rc

AP_CONVERSATION_TYPE_MIXED
The TP has issued both basic and mapped verbs. Only one type can be issued in a single conversation.

A secondary return code is not returned.

AP_DEALLOC_ABEND

The primary return code is:

primary_rc

AP_DEALLOC_ABEND
The conversation has been deallocated for one of the following reasons:

- The partner TP has issued the MC_DEALLOCATE verb with *dealloc_type* set to AP_ABEND.
- The partner TP has ended abnormally, causing the partner LU to send an MC_DEALLOCATE request.

A secondary return code is not returned.

AP_DEALLOC_ABEND_PROG

The primary return code is:

primary_rc

AP_DEALLOC_ABEND_PROG
The conversation has been deallocated for one of the following reasons:

- The partner TP has issued the DEALLOCATE verb with *dealloc_type* set to AP_ABEND_PROG.
- The partner TP has ended abnormally, causing the partner LU to send a DEALLOCATE request.

A secondary return code is not returned.

AP_DEALLOC_ABEND_SVC

The primary return code is:

primary_rc

AP_DEALLOC_ABEND_SVC
The conversation has been deallocated because the partner TP issued the DEALLOCATE verb with *dealloc_type* set to AP_ABEND_SVC.

A secondary return code is not returned.

AP_DEALLOC_ABEND_TIMER

The primary return code is:

primary_rc

AP_DEALLOC_ABEND_TIMER
The conversation has been deallocated because the partner TP has issued the DEALLOCATE verb with *dealloc_type* set to AP_ABEND_TIMER.

A secondary return code is not returned.

AP_DEALLOC_NORMAL

The primary return code is:

primary_rc

AP_DEALLOC_NORMAL
This return code does not indicate an error.

The partner TP issued the [MC_]DEALLOCATE verb with *dealloc_type* set to one of the following:

- AP_FLUSH
- AP_SYNC_LEVEL with the synchronization level of the conversation specified as AP_NONE

A secondary return code is not returned.

AP_DUPLEX_TYPE_MIXED

The primary return code is:

primary_rc

AP_DUPLEX_TYPE_MIXED
The TP has issued a conversation verb with a duplex type that does not match the conversation. If the conversation is full-duplex (as specified by the *duplex_type* parameter on [MC_]ALLOCATE or RECEIVE_ALLOCATE), the TP must set the option AP_FULL_DUPLEX_CONVERSATION in the *opext* parameter of all other verbs in this conversation. If the conversation is half-duplex, it must not set this option.

A secondary return code is not returned.

AP_INVALID_VERB

The primary return code is:

primary_rc

AP_INVALID_VERB
The opcode supplied for the verb is not valid. The verb did not execute.

This return code is also returned if you attempt to issue the [MC_]RECEIVE_AND_POST verb in a full-duplex conversation. [MC_]RECEIVE_AND_POST can be used only in a half-duplex conversation.

A secondary return code is not returned.

AP_INVALID_VERB_SEGMENT

WINDOWS

The primary return code is:

primary_rc

AP_INVALID_VERB_SEGMENT
The verb control block extended beyond the end of a data segment. The verb did not execute.

A secondary return code is not returned.

AP_PROG_ERROR_NO_TRUNC

The primary return code is:

primary_rc

AP_PROG_ERROR_NO_TRUNC
The partner TP 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.

A secondary return code is not returned.

AP_PROG_ERROR_PURGING

The primary return code is:

primary_rc

AP_PROG_ERROR_PURGING
The partner TP issued one of the following verbs:

- SEND_ERROR with *err_type* set to AP_PROG
- MC_SEND_ERROR

  while in Receive, Pending_Post, Confirm, Confirm_Send, or Confirm_Deallocate state. Data sent but not yet received is purged.

A secondary return code is not returned.

AP_PROG_ERROR_TRUNC

The primary return code is:

primary_rc

AP_PROG_ERROR_TRUNC
In SEND state, after sending an incomplete logical record, the partner TP issued a SEND_ERROR verb with *err_type* set to AP_PROG. The local TP may have received the first part of the logical record through a receive verb.

A secondary return code is not returned.

AP_SVC_ERROR_NO_TRUNC

The primary return code is:

primary_rc

AP_SVC_ERROR_NO_TRUNC
While in SEND state, the partner TP (or partner LU) issued a SEND_ERROR verb with *err_type* set to AP_SVC. Data was not truncated.

A secondary return code is not returned.

AP_SVC_ERROR_PURGING

The primary return code is:

primary_rc

AP_SVC_ERROR_PURGING
The partner TP (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 TP may have been purged.

A secondary return code is not returned.

AP_SVC_ERROR_TRUNC

The primary return code is:

primary_rc

AP_SVC_ERROR_TRUNC
In Send state, after sending an incomplete logical record, the partner TP (or partner LU) issued a SEND_ERROR verb. The local TP may have received the first part of the logical record.

A secondary return code is not returned.

AP_THREAD_BLOCKING

WINDOWS

The primary return code is:

primary_rc

AP_THREAD_BLOCKING
The verb was issued using the APPC (blocking) entry point, but another blocking APPC verb was already outstanding. For more information about these entry points, see “APPC Entry Points: Windows Systems” on page 36.

A secondary return code is not returned.

AP_TP_BUSY

The primary return code is:

primary_rc

AP_TP_BUSY
The local TP has issued a call to APPC while APPC was processing another call for the same TP. This may occur if the local TP has started multiple processes, and more than one process is issuing APPC calls using the same *tp_id*. However, ensure that each process issues its own TP_STARTED or RECEIVE_ALLOCATE verb to obtain its own *tp_id*; the results of multiple processes using the same *tp_id* are unpredictable.

WINDOWS

This return code may also indicate that the application issuing the verb was invoked using the Windows function `SendMessage` instead of `PostMessage`; the application cannot issue any verbs in this state. For more information, see "Windows Considerations" on page 51.

A secondary return code is not returned.

AP_UNEXPECTED_SYSTEM_ERROR

The primary return code is:

primary_rc

AP_UNEXPECTED_SYSTEM_ERROR
The operating system has encountered an error while processing an APPC call from the local TP. The operating system return code is returned through the *secondary_rc*. If the problem persists, consult your System Administrator.

UNIX

For the meaning of the operating system return code, see the file **/usr/include/errno.h** on the computer where the error occurred.

WINDOWS

For the meaning of the operating system return code, refer to your operating system documentation.

A secondary return code is not returned.

Appendix C. APPC State Changes

The following tables show the conversation states in which each APPC verb may be issued, and the state change which 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 shown in the same column as the verb. Where no *primary_rc* values are shown, the state changes are the same for all return codes, except as indicated in the notes following each table.

The possible conversation states are shown as column headings. For each combination of verb and *primary_rc* value, the following abbreviations and symbols are given under each state to indicate the results of issuing the verb in that state:

X Verb cannot be issued in this state.

T, S, R, ...
State of the conversation after the verb has completed.

For half-duplex conversations:

T	Reset
S	Send
SP	Send_Pending
R	Receive
C	Confirm
CS	Confirm_Send
CD	Confirm_Deallocate
P	Pending_Post

For full-duplex conversations:

T	Reset
SR	Send_Receive
S	Send_Only
R	Receive_Only

- There is no conversation state after the verb is issued.

/ It is not applicable to consider the previous state, because the verb starts a new conversation as though from Reset state; there is no effect on any existing conversation.

(blank)
The return code shown cannot occur in this state.

Half-duplex conversations

Verb and *primary_rc* Values	State in Which Issued							
	Reset (T)	**Send (S)**	**Send Pend (SP)**	**Recv (R)**	**Confm (C)**	**Confm Send (CS)**	**Confm Deall (CD)**	**Pend Post (PP)**
TP_STARTED								
AP_OK	T	/	/	/	/	/	/	/
other *primary_rc* values	-							
TP_ENDED								
AP_OK	-	-	-	-	-	-	-	-
other *primary_rc* values	T	S	SP	R	C	CS	CD	P
RECEIVE_ALLOCATE								
AP_OK	R	/	/	/	/	/	/	/
other *primary_rc* values	-							
GET_LU_STATUS	X	S	SP	R	C	CS	CD	P
GET_TP_PROPERTIES	T	S	SP	R	C	CS	CD	P
SET_TP_PROPERTIES	T	S	SP	R	C	CS	CD	P
GET_TYPE	X	S	SP	R	C	CS	CD	P
CANCEL_CONVERSATION	X	T	T	T	T	T	T	T
[MC_]ALLOCATE								
AP_OK	S	/	/	/	/	/	/	/
other *primary_rc* values	T							
[MC_]CONFIRM								
AP_OK	X	S	S	X	X	X	X	X
AP_ERROR		R	R					
[MC_]CONFIRMED	X	X	X	X	R	S	T	X
[MC_]DEALLOCATE								
AP_ABEND_* *dealloc_type* values	X	T	T	T	T	T	T	T
other *dealloc_type* values								
AP_ERROR	X	R	R	X	X	X	X	X
other *primary_rc* values	T	T						
[MC_]FLUSH	X	S	S	X	X	X	X	X
[MC_]GET_ ATTRIBUTES	X	S	SP	R	C	CS	CD	P
[MC_]PREPARE_TO_RECEIVE	X	R	R	X	X	X	X	X
[MC_]RECEIVE_AND_POST (See Note 4)	X	P	P	P	X	X	X	X
[MC_]RECEIVE_AND_WAIT	X	See Note 5	See Note 5	See Note 5	X	X	X	X
[MC_]RECEIVE_IMMEDIATE	X	X	X	See Note 5	X	X	X	X
[MC_]RECEIVE_EXPEDITED_DATA	X	X	X	R	C	X	X	P

Verb and *primary_rc* Values	State in Which Issued							
	Reset (T)	Send (S)	Send Pend (SP)	Recv (R)	Confm (C)	Confm Send (CS)	Confm Deall (CD)	Pend Post (PP)
[MC_]REQUEST_TO_SEND	X	X	X	R	C	X	X	P
[MC_]SEND_CONVERSATION	T	/	/	/	/	/	/	/
[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_]SEND_EXPEDITED_DATA	X	X	X	R	C	X	X	P
[MC_]TEST_RTS	X	S	S	R	C	CS	CD	P
[MC_]TEST_RTS_AND_POST	X	S	S	R	C	CS	CD	P

Note:

1. In the Return codes column of the table, the abbreviation AP_ERROR is used for:
   - AP_BACKED_OUT
   - 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 always enters Reset state if one of the following return codes is 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_INVALID_VERB
   - 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 has 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:

   AP_DATA, AP_DATA_COMPLETE, AP_DATA_INCOMPLETE
   Receive state

   AP_SEND
   Send state

   AP_DATA_SEND, AP_DATA_COMPLETE_SEND
   Send Pending state

   AP_CONFIRM_WHAT_RCVD, AP_DATA_CONFIRM, AP_DATA_COMPLETE_CONFIRM
   Confirm state

   AP_CONFIRM_SEND, AP_DATA_CONFIRM_SEND, AP_DATA_COMPLETE_CONFIRM_SEND
   Confirm Send state

   AP_CONFIRM_DEALLOCATE, AP_DATA_CONFIRM_DEALLOCATE, AP_DATA_COMPLETE_CONFIRM_DEALL
   Confirm Deallocate state

Full-duplex conversations

Verb and *primary_rc* Values	**State in Which Issued**			
	Reset (T)	**Send Receive (SR)**	**Send Only (S)**	**Receive Only (R)**
TP_STARTED				
AP_OK	T	/	/	/
other *primary_rc* values	-			
TP_ENDED				
AP_OK	-	-	-	-
other *primary_rc* values	T	SR	S	R
RECEIVE_ALLOCATE				
AP_OK	SR	/	/	/
other *primary_rc* values	-			
GET_LU_STATUS	X	SR	S	R
GET_TP_PROPERTIES	T	SR	S	R
SET_TP_PROPERTIES	T	SR	S	R

Verb and *primary_rc* Values	State in Which Issued			
	Reset (T)	Send Receive (SR)	Send Only (S)	Receive Only (R)
GET_TYPE	X	SR	S	R
CANCEL_CONVERSATION	X	T	T	T
[MC_]ALLOCATE				
AP_OK	SR	/	/	/
other *primary_rc* values	T			
[MC_]DEALLOCATE				
AP_ABEND_* *dealloc_type* values	X	T	T	T
other *dealloc_type* values	X	R	T	X
[MC_]FLUSH	X	SR	S	X
[MC_]GET_ ATTRIBUTES	X	SR	S	R
[MC_]RECEIVE_AND_WAIT				
AP_OK	X	SR	X	R
AP_ERROR	X	SR	X	R
AP_DEALLOC_NORMAL	X	S	X	T
[MC_]RECEIVE_IMMEDIATE				
AP_OK	X	SR	X	R
AP_ERROR	X	SR	X	R
AP_DEALLOC_NORMAL	X	S	X	T
[MC_]RECEIVE_ EXPEDITED_DATA	X	SR	S	R
[MC_]SEND_DATA				
AP_OK	X	SR	S	X
AP_ERROR	X	SR	T	X
[MC_]SEND_ERROR				
AP_OK	X	SR	S	X
AP_ERROR	X	SR	T	X
[MC_]SEND_ EXPEDITED_DATA	X	SR	S	R

Note:

1. In the Return codes column of the table, the abbreviation AP_ERROR is used for:
   AP_BACKED_OUT
   AP_PROG_ERROR_TRUNC
   AP_PROG_ERROR_NO_TRUNC
   AP_SVC_ERROR_TRUNC
   AP_SVC_ERROR_NO_TRUNC
2. The conversation always enters Reset state if one of the following return codes is 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

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_INVALID_VERB
   AP_PARAMETER_CHECK
   AP_STATE_CHECK
   AP_TP_BUSY
   AP_UNEXPECTED_SYSTEM_ERROR
   AP_UNSUCCESSFUL

Appendix D. SNA LU 6.2 Support

This appendix details how the Communications Server implementation of APPC relates to the LU 6.2 architecture. It includes the following information:

- A summary of the LU 6.2 option sets supported by Communications Server
- A list of the control operator verbs which are included in the Communications Server APPC implementation
- A list of the control operator verbs whose functions are performed in Communications Server by the administration tools or by the NOF API.

LU 6.2 Option Set Support

Communications Server APPC supports the base set of LU 6.2 functions, and a selection of the option sets. Some of these option sets are supported by APPC verbs; others are supported by the administration tools or by the NOF API.

The following tables list the option sets supported by Communications Server, with the option set reference number specified in IBM's *Transaction Programmer's Reference Manual for LU Type 6.2.* (Earlier versions of this IBM manual used different reference numbers.)

LU 6.2 Option Sets Supported by APPC Verbs

Reference Number	Option set
101	Flushing the LU's send buffer
102	GET_ATTRIBUTES
103	POST_ON_RECEIPT with test for posting *
104	POST_ON_RECEIPT with wait *
105	PREPARE_TO_RECEIVE
106	RECEIVE_IMMEDIATE
109	Get TP name and instance identifier
110	GET_CONVERSATION_TYPE
112	Full-duplex conversations and expedited data
113	Non-blocking 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 for when session free
211	Session-level LU-LU verification
212	User ID verification
213	Program-supplied user ID and password
214	User ID authorization
241	Sending PIP data
242	Receiving PIP data
243	Accounting
244	Long locks
245	Test for REQUEST_TO_SEND received
247	User Control data
290	Logging of data in a system log
291	Mapped conversation LU Services component
401	Reliable One-Way Brackets

Reference Number	Option set
616	CPSVCMG mode name support

*Options 103 and 104 are supported by the [MC_]RECEIVE_AND_POST verb.

LU 6.2 Option Sets Supported by the Administration Tools and by the NOF API

Reference Number	Option set
501	CHANGE_SESSION_LIMIT
502	ACTIVATE_SESSION
504	DEACTIVATE_SESSION
505	LU definition verbs
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
611	Session-level mandatory cryptography
612	Contention winner automatic activation limit
613	Local maximum (LU, mode) session limit

Control Operator Verb Support

The functions of the following control operator verbs are provided as part of the Communications Server APPC implementation:

RECEIVE_ALLOCATE
TP_STARTED
TP_ENDED

The functions of the following control operator verbs are provided by the Communications Server administration programs and by the NOF API.

INITIALIZE_SESSION_LIMITS
CHANGE_SESSION_LIMITS
RESET_SESSION_LIMITS
DISPLAY_LU
DISPLAY_REMOTE_LU
DISPLAY_MODE
DISPLAY_TP
ACTIVATE_SESSION
DEACTIVATE_SESSION
DEFINE_LOCAL_LU
DEFINE_REMOTE_LU
DEFINE_MODE
DELETE

Appendix E. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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, North Carolina 27709-2195
U.S.A.

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

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

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE: This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

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

® (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. ® Copyright IBM Corp. _enter the year or years_.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Adobe and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other product and service names might be trademarks of IBM or other companies.

Bibliography

The following IBM publications provide information about the topics discussed in this library. The publications are divided into the following broad topic areas:

- IBM Communications Server for AIX
- IBM Communications Server for Linux
- Systems Network Architecture (SNA)
- Advanced Program-to-Program Communication (APPC)
- Programming

For IBM Communications Server for AIX and IBM Communications Server for Linux books, brief descriptions are provided. For other books, only the titles and order numbers are shown here.

IBM Communications Server for AIX Publications

The IBM Communications Server for AIX library comprises the following books. In addition, softcopy versions of these documents are provided on the CD-ROM. See *IBM Communications Server for AIX Quick Beginnings* for information about accessing the softcopy files on the CD-ROM. To install these softcopy books on your system, you require 9–15 MB of hard disk space (depending on which national language versions you install).

- *IBM Communications Server for AIX Migration Guide* (SC31-8585)

  This book explains how to migrate from Communications Server for AIX Version 4 Release 2 or earlier to IBM Communications Server for AIX Version 6.
- *IBM Communications Server for AIX Quick Beginnings* (GC31-8583)

  This book is a general introduction to IBM Communications Server for AIX, including information about supported network characteristics, installation, configuration, and operation.
- *IBM Communications Server for AIX Administration Guide* (SC31-8586)

  This book provides an overview of SNA and IBM Communications Server for AIX, and information about IBM Communications Server for AIX configuration and operation.
- *IBM Communications Server for AIX Administration Command Reference* (SC31-8587)

  This book provides information about SNA and IBM Communications Server for AIX commands.
- *IBM Communications Server for AIX or Linux CPI-C Programmer's Guide* (SC23-8591)

  This book provides information for experienced "C" or Java programmers about writing SNA transaction programs using the IBM Communications Server CPI Communications API.
- *IBM Communications Server for AIX or Linux APPC Programmer's Guide* (SC23-8592)

  This book contains the information you need to write application programs using Advanced Program-to-Program Communication (APPC).
- *IBM Communications Server for AIX or Linux LUA Programmer's Guide* (SC23-8590)

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

- *IBM Communications Server for AIX or Linux CSV Programmer's Guide* (SC23-8589)

  This book contains the information you need to write application programs using the Common Service Verbs (CSV) application program interface (API).
- *IBM Communications Server for AIX or Linux MS Programmer's Guide* (SC23-8596)

  This book contains the information you need to write applications using the Management Services (MS) API.
- *IBM Communications Server for AIX NOF Programmer's Guide* (SC31-8595)

  This book contains the information you need to write applications using the Node Operator Facility (NOF) API.
- *IBM Communications Server for AIX Diagnostics Guide* (SC31-8588)

  This book provides information about SNA network problem resolution.
- *IBM Communications Server for AIX or Linux APPC Application Suite User's Guide* (SC23-8595)

  This book provides information about APPC applications used with IBM Communications Server for AIX.
- *IBM Communications Server for AIX Glossary* (GC31-8589)

  This book provides a comprehensive list of terms and definitions used throughout the IBM Communications Server for AIX library.

IBM Communications Server for Linux Publications

The IBM Communications Server for Linux library comprises the following books. In addition, softcopy versions of these documents are provided on the CD-ROM. See *IBM Communications Server for Linux Quick Beginnings* for information about accessing the softcopy files on the CD-ROM. To install these softcopy books on your system, you require 9–15 MB of hard disk space (depending on which national language versions you install).

- *IBM Communications Server for Linux Quick Beginnings* (GC31-6768 and GC31-6769)

  This book is a general introduction to IBM Communications Server for Linux, including information about supported network characteristics, installation, configuration, and operation. There are two versions of this book:
  GC31-6768 is for IBM Communications Server for Linux on the i686, x86_64, and ppc64 platforms
  GC31-6769 is for IBM Communications Server for Linux for System z.
- *IBM Communications Server for Linux Administration Guide* (SC31-6771)

  This book provides an overview of SNA and IBM Communications Server for Linux, and information about IBM Communications Server for Linux configuration and operation.
- *IBM Communications Server for Linux Administration Command Reference* (SC31-6770)

  This book provides information about SNA and IBM Communications Server for Linux commands.
- *IBM Communications Server for AIX or Linux CPI-C Programmer's Guide* (SC23-8591)

  This book provides information for experienced "C" or Java programmers about writing SNA transaction programs using the IBM Communications Server CPI Communications API.
- *IBM Communications Server for AIX or Linux APPC Programmer's Guide* (SC23-8592)

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

- *IBM Communications Server for AIX or Linux LUA Programmer's Guide* (SC23-8590)

  This book contains the information you need to write applications using the Conventional LU Application Programming Interface (LUA).
- *IBM Communications Server for AIX or Linux CSV Programmer's Guide* (SC23-8589)

  This book contains the information you need to write application programs using the Common Service Verbs (CSV) application program interface (API).
- *IBM Communications Server for AIX or Linux MS Programmer's Guide* (SC23-8596)

  This book contains the information you need to write applications using the Management Services (MS) API.
- *IBM Communications Server for Linux NOF Programmer's Guide* (SC31-6778)

  This book contains the information you need to write applications using the Node Operator Facility (NOF) API.
- *IBM Communications Server for Linux Diagnostics Guide* (SC31-6779)

  This book provides information about SNA network problem resolution.
- *IBM Communications Server for AIX or Linux APPC Application Suite User's Guide* (SC23-8595)

  This book provides information about APPC applications used with IBM Communications Server for Linux.
- *IBM Communications Server for Linux Glossary* (GC31-6780)

  This book provides a comprehensive list of terms and definitions used throughout the IBM Communications Server for Linux library.

Systems Network Architecture (SNA) Publications

The following books contain information about SNA networks:

- *Systems Network Architecture: Format and Protocol Reference Manual—Architecture Logic for LU Type 6.2* (SC30-3269)
- *Systems Network Architecture: Formats* (GA27-3136)
- *Systems Network Architecture: Guide to SNA Publications* (GC30-3438)
- *Systems Network Architecture: Network Product Formats* (LY43-0081)
- *Systems Network Architecture: Technical Overview* (GC30-3073)
- *Systems Network Architecture: APPN Architecture Reference* (SC30-3422)
- *Systems Network Architecture: Sessions between Logical Units* (GC20-1868)
- *Systems Network Architecture: LU 6.2 Reference—Peer Protocols* (SC31-6808)
- *Systems Network Architecture: Transaction Programmer's Reference Manual for LU Type 6.2* (GC30-3084)
- *Systems Network Architecture: 3270 Datastream Programmer's Reference* (GA23-0059)
- *Networking Blueprint Executive Overview* (GC31-7057)
- *Systems Network Architecture: Management Services Reference* (SC30-3346)

APPC Publications

The following books contain information about Advanced Program-to-Program Communication (APPC):

- *APPC Application Suite V1 User's Guide* (SC31-6532)
- *APPC Application Suite V1 Administration* (SC31-6533)
- *APPC Application Suite V1 Programming* (SC31-6534)

- *APPC Application Suite V1 Online Product Library* (SK2T-2680)
- *APPC Application Suite Licensed Program Specifications* (GC31-6535)
- *z/OS V1R2.0 Communications Server: APPC Application Suite User's Guide* (SC31-8809)

Programming Publications

The following books contain information about programming:

- *Common Programming Interface Communications CPI-C Reference* (SC26-4399)
- *Communications Server for OS/2 Version 4 Application Programming Guide* (SC31-8152)

Index

Special characters

A

B

C

D

E

F

G

H

I

L

M

N

P

Q

R

S

T

U

V

W

Communicating your comments to IBM

If you especially like or dislike anything about this document, 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 document. 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.

Send your comments to us in any of the following ways:

- To send comments by FAX, use this number: 1+919-254-1258
- To send comments electronically, use this address: comsvrcf@us.ibm.com
- To send comments by post, use this address:

  International Business Machines Corporation
  Attn: z/OS® Communications Server Information Development
  P.O. Box 12195, 3039 Cornwallis Road
  Department AKCA, Building 501
  Research Triangle Park, North Carolina 27709-2195

Make sure to include the following in your note:

- Title and publication number of this document
- Page number or topic to which your comment applies.

Product Number: 5725-H32

Printed in USA

SC23-8592-01