Advanced Program-to-Program Communication for the IBM Personal Computer

Programming Guide

Communications Family

Personal Computer Software

61X3842

Advanced Program-to-Program Communication for the IBM Personal Computer

Programming Guide

Communications Family

Personal Computer Software

First Edition (February 1986)

The program product described in this manual, and all licensed material available for it, is provided by IBM under terms of the Agreement for IBM Licensed Programs. Changes are made periodically to the information herein; these changes will be incorporated into new editions of this publication.

Any reference to an IBM program product in this document is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Publications are not stocked at the address given below. If you want more IBM publications, ask your IBM representative or write to the IBM branch office serving your locality.

A form for your comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Department E02, P.O. Box 12195, Research Triangle Park, North Carolina 27709. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Preface

This manual describes how to develop a program that uses the Advanced Program-to-Program Communication for the IBM Personal Computer (APPC/PC) program product. APPC/PC is a data communication system that enables IBM PC transaction programs to communicate with other transaction programs on other IBM PCs that have APPC/PC. APPC/PC also enables the IBM PC to communicate with APPC on other systems, such as the Series/1, System/36, System/38, and System/370 CICS.

APPC/PC provides programs with a distributed transaction processing capability. APPC/PC enables a program on an IBM PC to communicate with another program on another system without operator intervention. This manual shows you how to write IBM PC transaction programs that use APPC/PC to communicate with other programs.

Who Should Use This Manual

Application programmers can use this manual to develop programs that use the APPC/PC application program interface (API). The API consists of the commands (verbs) and return codes described in this manual. The application programmer needs to know how to program the IBM PC, and needs to understand SNA concepts.

How This Manual is Organized

Chapter 1, "Introduction to APPC/PC," describes supported features of APPC/PC and how APPC/PC fits in the IBM PC relative to other IBM PC programs. It describes transaction programs and application subsystem programs and introduces other terms used throughout this manual.

Chapter 2, "Developing an Application Subsystem," describes functions that should be considered when designing an application subsystem.

Chapter 3, "Developing a Transaction Program," describes functions that should be considered when designing a transaction program.

Chapter 4, "Introduction to APPC/PC Verbs," briefly describes verb types and the common verb format used in chapters 5 through 9.

Chapter 5, "Using Control Verbs," describes the application program interface (API) to control verbs. First, the control verbs that the application subsystem sends to APPC/PC are described, followed by the control verbs APPC/PC sends to the application subsystem. Near the end of this chapter there are examples of activating and deactivating a node.

Chapter 6, "Using Transaction Mapped Conversation Verbs," describes the application program interface for mapped conversation verbs. Preceding the individual verb descriptions is a discussion of the conversation states that determine which verbs can be issued, and a discussion on understanding mapped conversation return codes.

Chapter 7, "Using Transaction Basic Conversation Verbs," describes the application program interface for basic conversation verbs. Preceding the individual verb descriptions is a discussion of the conversation states that determine which verbs can be issued, and a discussion on understanding basic conversation return codes.

Chapter 8, "Using the Network Management Verb," describes the verb used to provide management services information to a network management services function.

Chapter 9, "Other APPC/PC Services," describes other verbs provided by APPC/PC for the convenience of the programmer. One verb assists with communication between the application subsystem and transaction programs. Other verbs assist with data conversion (ASCII/EBCDIC), tracing facilities, and disabling and

reenabling APPC/PC to avoid recursion problems in exit routines.

Chapter 10, "Resolving Error Conditions," describes three types of error conditions with possible solutions and ways to avoid them. The types of errors discussed are return codes, SYSLOG reported errors, and system deadlocks.

Appendix A, "Verb Operation Codes and Formats," lists the operation codes for APPC/PC verbs and the internal formats for the parameter lists passed between the application subsystem or the transaction program, and APPC/PC.

Appendix B, "Conversation State Matrices," shows the conversation state transitions that can occur when a program issues a conversation verb.

Appendix C, "Verb Return Codes," shows the return codes that APPC/PC can report to a program through the RETURN_CODE parameter of each verb.

Appendix D, "SYSLOG Type Codes," lists the SYSLOG type codes that represent error conditions and includes data errors reported by the transaction program, link errors, configuration errors, and system protocol errors.

Appendix E, "Sample Programs," describes the sample programs supplied on the *APPC/PC Structures and Sample Programs* diskette. There is also a listing for a sample CICS program.

Appendix F, "Sample CICS Host Configuration for APPC/PC," describes sample CICS and VTAM definitions necessary to use APPC/PC.

Appendix G, "APPC/PC Implementation of the LU6.2 Architecture," describes the optional functions of APPC architecture supported by APPC/PC. This appendix also includes a mapping of APPC/PC verbs and parameters with the verbs and parameters used in the APPC architecture documents.

Appendix H, "ASCII/EBCDIC Translation Tables," describes translation tables used by the conversion verb provided by APPC/PC.

Appendix I, "Statement of Service," contains a discussion of IBM service as related to the APPC/PC program product.

Following the appendices are a glossary and an index.

Related Publications

The following publications provide additional information on the topics discussed in this manual.

- *Advanced Program-to-Program Communication for the IBM Personal Computer Installation and Configuration Guide* contains planning and IBM PC set-up information for APPC/PC.

- *Introduction to Advanced Program-to-Program Communication*, GG24-1584, contains general information about APPC.

- *Systems Network Architecture Concepts and Products*, GC30-3072, contains basic information on SNA for those readers wanting either an overview or a foundation for further study.

- *Systems Network Architecture Technical Overview*, GC30-3073, contains additional details on SNA, especially on functions and control sequences; bridges the gap between the most elementary overview of SNA and the detailed descriptions of the formats and protocols.

- *Systems Network Architecture Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084, contains reference information on LU type

6.2 (APPC) verbs for programmers writing transaction programs to run on SNA.

- *Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*, SC30-3269, contains information for system programmers and others who need detailed information about SNA LU Type 6.2 (APPC) to adapt a program to function within an SNA network.

- *Systems Network Architecture Sessions Between Logical Units*, GC20-1868, contains references on SNA formats and protocols for LU types other than Type 6.2.

- *Systems Network Architecture Reference Summary*, GA27-3136, contains summary information on SNA formats and sequences.

- *IBM SDLC General Information*, GA27-3093, contains supplementary details of Synchronous Data Link Control.

- *IBM Token-Ring Network Introduction and Planning Guide*, GA27-3677, contains planning information for IBM Token-Ring Network.

- *IBM Token-Ring Network PC Adapter Guide to Operations*, SA27-3710, is the IBM Token-Ring Network operations guide.

- *IBM Token-Ring Network PC Adapter Technical Reference Manual*, SC30-3383, contains additional reference information for IBM Token-Ring Network.

- *IBM Token-Ring Network Problem Determination Guide*, SY27-1280, contains information on problem diagnosis for the IBM Token-Ring Network.

- *IBM Option Instructions for the SDLC Communication Adapter*, supplied with the IBM SLDC communication adapter, contains adapter installation and connector information.

Contents

Chapter 1. Introduction to APPC/PC

The APPC/PC Program

APPC/PC is a data communications subsystem for the IBM Personal Computer. APPC/PC provides SNA Advanced Program-to-Program Communication (APPC) for application programs that perform distributed transaction processing.

An application program using APPC/PC can communicate with application programs on other systems that support APPC. In this book a *transaction program* is an application program that uses APPC/PC communication functions.

Transaction programs use APPC/PC verb sequences to communicate with other programs at other SNA nodes. You can regard this set of verbs as a programming language in which you can write conversations. APPC/PC verbs are coded as records, each having a precisely defined syntax. Your transaction programs gain access to APPC/PC facilities by providing verb records to the APPC/PC application program interface (API) that appears as an operating system extension of PC DOS.

Whether you write your own application or use an existing application, you must first set up your IBM Personal Computer to operate in a computer network. APPC/PC simplifies setup by providing configuration menus that help you configure your IBM Personal Computer for connection to an IBM Token-Ring Network, to an SNA data communication network using Synchronous Data Link Control (SDLC) leased or switched facilities, or to both. A simple direct connection

between IBM Personal Computers using SDLC is also possible.

What You Need

To use APPC/PC as described in this manual, you need:

- The *APPC/PC program product* consisting of the *APPC/PC Installation and Configuration Guide* and two diskettes: the *APPC/PC Program Diskette*, and the *APPC/PC Structures and Sample Programs* diskette.

  The *Guide* describes hardware and software requirements and provides information on planning, configuring, and loading APPC/PC.

  The *APPC/PC Program Diskette* contains:

  - APPC/PC load and unload commands
  - The APPC/PC configuration program
  - Data Link Control files
  - A sample type G conversation table.

  The *APPC/PC Structures and Sample Programs* diskette contains:

  - APPC/PC assembler verb structures, which are used to write application subsystems and transaction programs
  - A set of sample programs to illustrate the design of an application subsystem and the coding of APPC/PC verbs using the macro assembler verb structures.

- In addition to the APPC/PC program product, you need a language to compile your application subsystem and transaction programs. You can use any language that supports:
  - Issuing software interrupts
  - Setting general registers
  - Building parameter lists (as described in Appendix A)
  - Providing addressability to sections of code.

For information on specifying the APPC/PC configuration and running APPC/PC, see the *APPC/PC Installation and Configuration Guide.*

Features Supported

APPC/PC provides the SNA APPC node support as defined in the *SNA Format and Protocol Reference Manual.*

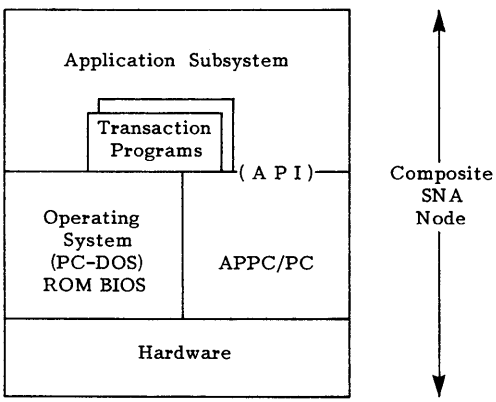
APPC/PC supports an open API including the following:

- Base APPC (LU Type 6.2/PU Type 2.1)
- Parallel sessions (when attached to peer PU Type 2.1 nodes)
- Synchronization level of None or Confirm
- Mapped conversation support (but no data mapping)
- Peer and boundary function attachment
- Support for conversation- and session-level security

- Network Management support for ALERT, PDSTATS, and general NMVTs
- Normal Response Mode SDLC
  - Primary point-to-point
  - Secondary point-to-point or multipoint
  - Switched support for manual dial, manual answer, and auto-answer
- IBM Token-Ring Network support
- Compatibility with CICS/VS Version 1.7
- Operation under the latest level of IBM PC DOS at availability of the IBM Token-Ring Network.

SNA and APPC Terminology

The diagram below illustrates the relationship between the IBM PC hardware and the software components involved in the operation of APPC/PC.



The *application subsystem* refers to one or more programs whose primary function is to provide services for APPC/PC and transaction programs. The application subsystem logs errors, manages incoming conversations, loads transaction programs, and provides other services. The *transaction programs* use APPC/PC communication services to communicate with a partner transaction program to perform transactions. The API is the set of commands that the application subsystem and transaction programs use to communicate with APPC/PC. The application subsystem and transaction programs are provided by the user.

A remote transaction program can request a local application subsystem to start a local transaction program so that the programs can exchange data. The corresponding transaction programs are called *partner transaction programs.*

You can choose the type of *conversation* that your transaction program uses: a *basic conversation* or a *mapped conversation.* The type of conversation you use depends on whether you need full access to the SNA *general data stream* (GDS) as provided by basic conversations. A header field (LLID) precedes all data that a program sends in the GDS format. The logical length (LL) portion of the header field specifies the overall length of the data and the identification (ID) portion specifies the type of data.

In basic conversations, data passed to and received from the APPC/PC API must contain at least the LL fields of the GDS headers. (See *SNA Reference Summary* for information on GDS variables.) The transaction program must build and interpret the LL fields but the ID fields are optional. The ID information is necessary only if the partner program expects to receive GDS variables.

In mapped conversations, the data that programs pass to and receive from the APPC/PC API is simply user data. A transaction program using mapped conversations does not require GDS headers to describe the data; therefore, the program does not have to build or interpret these headers. When the transaction program uses mapped

conversations, APPC/PC builds and interprets GDS variables.

APPC/PC verbs fall into three general categories:

- Conversation verbs used by a transaction program to communicate data
- Control verbs that an application subsystem uses to request services from or provide services to APPC/PC
- System services verbs for tasks such as network management and ASCII/EBCDIC conversion.

APPC/PC for the Transaction Programmer

APPC/PC provides a transaction program API and an interface to the control functions within the application subsystem that the system programmer provides.

The communication services of APPC/PC extend the services that the operating system normally provides. These services include communication primitives that enable a transaction to use a conversation to communicate with a partner transaction. Each conversation is half-duplex—that is, the transaction program with the right to send data must give up that right before its partner transaction program can send data. For descriptions of the capabilities of the transaction API, see Chapter 6, "Using Transaction Mapped Conversation Verbs," and Chapter 7, "Using Transaction Basic Conversation Verbs."

Understanding Locally Initiated and Remotely Initiated Transactions

A transaction can start in one of two ways: by an action initiated at your IBM PC or by an action initiated by a remote transaction program. Initially, the creating transaction program has the right to send data and the created transaction program does not have the right to send data. After initialization, the verb sequences that the programs issue determine the right to send data.

Supporting Multiple Conversations

A transaction program can have conversations with several partners simultaneously. However, a transaction that a remote program initiates is always a new transaction. Therefore, to have more than one conversation with remote programs, a local program must initiate all conversations except the first one.

Each conversation uses a logical resource called a *session*, and the conversation can use this session for as long as it requires. However, while a conversation is using a session, no other conversation can use it. When a transaction requests APPC/PC to allocate a conversation, APPC/PC responds by establishing a conversation and assigning it to a session.

APPC/PC for the System Programmer

The system programmer provides an application subsystem that uses the defined APPC/PC interface. The primary function of the application subsystem is to provide services for APPC/PC, but the system programmer can also use the application subsystem to provide services for the transaction programs.

The application subsystem manages the services of the communication node not managed by APPC/PC. These services include:

- Defining the logical characteristics of the node, including the Physical Unit (PU), the Logical Units (LUs), the partner LUs, and the desired number of sessions with each partner
- Activating the adapters
- Handling logged error messages
- Providing LU-LU passwords (if security features are required)
- Validating and loading a remotely initiated transaction program (if the capability for remote initiation is required)
- Managing cancellation of a transaction program (optional).

In addition to the application subsystem services just listed, the system programmer can provide other services to the transaction program, as appropriate.

The interface between the application subsystem and APPC/PC consists of two parts:

- A verb interface to establish the PU and the LUs, and to define partner LUs, session limits, and other communication parameters. Control verbs are described in Chapter 5, "Using Control Verbs."
- A set of exit routines to manage incoming transaction requests, log errors, and provide LU-LU passwords, also described in Chapter 5, "Using Control Verbs."

For information on the design of an application subsystem see Chapter 2, "Developing an Application Subsystem."

Understanding Initial Application Subsystem to APPC/PC Interactions

The application subsystem must issue verbs to define the capabilities of the communication node. An ATTACH_PU verb establishes the PU and an ATTACH_LU verb establishes each LU. These verbs provide information such as the LU name, processing capabilities, and a method of handling incoming requests for conversations. The application subsystem issues verbs so that the transaction programmer does not need to be concerned with the system definition.

For example, the system programmer may want to provide, as part of the application subsystem, predefined utility routines to execute attachment sequences for particular system configurations.

Managing Incoming Requests for Conversations

The application subsystem must manage requests for conversations (incoming ALLOCATEs) from other transaction programs. The system programmer must decide, for each LU, the best way to manage these incoming ALLOCATEs. Three options are available:

- The application subsystem can reject incoming ALLOCATEs entirely
- The application subsystem can provide an exit procedure that APPC/PC can call when an incoming ALLOCATE arrives (asynchronous option).
- You can direct the LU to queue the incoming ALLOCATEs until the application subsystem requests them (synchronous option)

To process an incoming ALLOCATE, the application subsystem must validate the request, load and/or initiate the requested transaction program, and provide the

transaction program with the parameters it must have to issue APPC/PC conversation verbs.

Chapter 2. Developing an Application Subsystem

APPC/PC runs under an application subsystem that manages features such as node definition, incoming requests for conversations (incoming ALLOCATEs), and error logging. This chapter describes the APPC/PC support options available to the application subsystem designer.

Efficient designs separate the application subsystem from the transaction programs, and define and manage nodes separately from the transaction programs so that these transaction programs can concentrate on conversations with particular partners. In this way you need to define the characteristics of the node and its partners only once, independently of the individual conversations.

Certain portions of the application subsystem should be loaded and remain resident during transaction processing, so that they can communicate with the transaction programs as necessary. These portions become a DOS extension, just as APPC/PC becomes a DOS extension. An example of how to create an application subsystem is described in Appendix E, "Sample Programs."

Coding the Initial Attach Sequence

After loading APPC/PC through a DOS command, the normal sequence for the application subsystem to issue APPC/PC verbs is as follows:

The **ATTACH_PU** verb defines and establishes APPC/PC conversations for the PU. It must be the first verb issued after APPC/PC loading, or, if the node is to be redefined, the first verb issued after issuing DETACH_PU.

The **ATTACH_LU** verb defines each local LU and its characteristics, the potential partner LUs, and the transmission service modes (called modes). It specifies session limits for the LU, the partner LU, and the mode. The ATTACH_LU verb returns an identifier of the LU (LU_ID). The application subsystem must save the LU_ID so that it can be used to specify the LU to which the transaction belongs.

The **ACTIVATE_DLC** verb opens the adapters designated in the APPC/PC menus. The application subsystem should issue this verb after issuing the ATTACH_LU verb. This sequence avoids receiving messages from a remote node before the application subsystem defines the local LU.

If you are using a switched SDLC link, you should make the connection at this point.

The **CNOS** verb sets the active number of sessions with a partner LU, and negotiates session limits with a partner LU if parallel sessions are defined. The application subsystem should issue the CNOS verb after issuing the ACTIVATE_DLC verb. This sequence assumes that the communication adapters will be open in case CNOS negotiation is required.

If the session limit is greater than one, the application subsystem should issue the first CNOS for the special

mode named 'SNASVCMG' (EBCDIC characters). This special mode enables the establishment of a session on which to perform CNOS negotiation with the partner LU.

CNOS must set the number of sessions before APPC/PC can start any conversations.

Initiate transactions resulting from incoming ALLOCATEs (through the CREATE_TP exit or the GET_ALLOCATE verb) or initiate a transaction locally by issuing a TP_STARTED verb. The transaction programs can now communicate with each other using conversation verbs.

The **TP_ENDED** verb signals that the transaction program is ending its operation.

The application subsystem must issue TP_ENDED before issuing DETACH_LU.

Use the **CNOS** verb to lower the session limits (partner LU and mode name) to 0, which also causes deactivation of the sessions. The application subsystem must set the session limits for all partner LUs to 0 through CNOS before issuing DETACH_LU.

The **DETACH_LU** or **DETACH_PU** verbs enable the application subsystem to undefine local LUs, their partner LUs, and/or the PU.

Starting a Transaction Program

A transaction program may begin operating because of an action initiated at your local IBM PC or an action initiated by a remote partner.

Locally Initiated Transactions

A locally initiated transaction is initiated by an action at the local node, rather than by an incoming ALLOCATE. The application subsystem must issue a TP_STARTED verb for each locally initiated transaction program to establish APPC/PC resources to support that program.

The TP_STARTED verb returns an identifer for the transaction program (TP_ID). All further verbs issued by the transaction program must include this TP_ID. The application subsystem must communicate the TP_ID to the transaction program after it receives the TP_ID from APPC/PC. For more information on the TP_ID identifier, see "Communicating Identifiers" on page 2-6.

Remotely Initiated Transactions

A transaction program operating at a remote node can initiate a transaction at the local node by issuing an ALLOCATE. This type of transaction is called a remotely initiated transaction. The remote LU sends a request for a transaction, called an incoming ALLOCATE, to the local node. The local node responds by accepting or rejecting the incoming ALLOCATE.

If accepted, the incoming ALLOCATE establishes an initial conversation, and the local transaction program starts in RECEIVE state.

Three options for managing transaction initiation from incoming ALLOCATEs are available:

Option 1: Rejection

The application subsystem rejects incoming ALLOCATEs. No exit routine is needed.

Option 2: Synchronous Management

The local LU queues incoming ALLOCATEs until you request one by issuing an ALLOCATE through a GET_ALLOCATE verb. APPC/PC assigns a TP_ID and

CONV_ID (conversation ID) at this time. The application subsystem should not issue a TP_STARTED verb for a transaction program started by an incoming ALLOCATE. The application subsystem must communicate the TP_ID and CONV_ID parameters to the remotely initiated transaction program. For more information on the TP_ID and CONV_ID parameters, see "Communicating Identifiers" on page 2-6.

A TP_VALID verb must follow the GET_ALLOCATE to tell APPC/PC whether the transaction program is valid and correctly loaded.

After APPC/PC accepts a request for a transaction program, you may choose to issue the CHANGE_LU verb to purge the incoming ALLOCATE queues for all LUs and then begin rejecting incoming ALLOCATEs. This procedure keeps the partner who issued the second ALLOCATE from waiting until the accepted transaction program is finished. In some cases this procedure avoids potential deadlocks. (For more information on resolving deadlocks, see "System Deadlocks" on page 10-2.)

When the accepted transaction program ends, you should issue TP_ENDED to free APPC/PC resources for the transaction program and CHANGE_LU to resume queueing incoming ALLOCATEs.

The GET_ALLOCATE verb does not suspend while waiting for an incoming ALLOCATE, but simply checks whether one is present (and optionally dequeues it). Therefore, you can use GET_ALLOCATE in a loop along with checking for input from the communication line, the keyboard, and other asynchronous devices.

Issuing CNOS(PARTNER_LU_MODE_SESSION_LIMIT=0) does not deactivate all sessions immediately because each queued incoming ALLOCATE is using a session. The CNOS verb deactivates each session after you reject or service the incoming ALLOCATE using that session. Issue the CHANGE_LU verb to reject queued incoming ALLOCATEs. To service these incoming ALLOCATEs, continue issuing GET_ALLOCATE and

initiate the requested transactions until the DISPLAY verb indicates that no sessions remain active.

Option 3: Asynchronous Management

The application subsystem processes each incoming ALLOCATE as it arrives. You must provide an exit address for a routine that will validate and, if necessary, load the requested transaction program. APPC/PC assigns a TP_ID and CONV_ID at this time, so the application subsystem should not issue TP_STARTED for a transaction program started by an incoming ALLOCATE. The application subsystem must communicate the TP_ID and CONV_ID to the remotely initiated transaction program.

When the transaction program ends, you should issue TP_ENDED to free APPC/PC resources for the transaction program.

For more information on asynchronous management of incoming ALLOCATEs, see "Managing Asynchronous Incoming ALLOCATEs (CREATE_TP Exit)" on page 2-12.

Communicating Identifiers

A transaction program must know the TP_ID and CONV_ID identifiers before it can issue valid verbs. Because the transaction program did not issue ATTACH_LU, it may not have direct access to the LU_ID it requires before it can issue the TP_STARTED verb.

For a locally initiated transaction, the application subsystem can use the SET_PASSTHROUGH and PASSTHROUGH verbs to communicate the TP_ID to the transaction program. (APPC/PC returns the CONV_ID on the ALLOCATE verb that starts each conversation.) Using the PASSTHROUGH and SET_PASSTHROUGH verbs, the application subsystem can define its own utility verb which, in turn, issues the TP_STARTED verb, and returns the TP_ID to the transaction program.

For a remotely initiated transaction, the application subsystem can use the same utility verb to return both the TP_ID and CONV_ID without issuing the TP_STARTED verb.

The utility verb will use the same interrupt mechanism that APPC/PC uses (INT X'68'). The application subsystem has to keep information obtained from files, or from its own menus, relating the transaction program names to the LU names and LU_IDs returned from the ATTACH_LU verb. See Appendix E, "Sample Programs" for an example of this technique. TP_INITIATE, the sample utility verb described in Appendix E, is user-defined and not a part of APPC/PC.

Communicating within the Application Subsystem

It may also be necessary to communicate the LU_ID between portions of the application subsystem. For instance, one section of the application subsystem may issue the ATTACH_LU verb, and another manage incoming ALLOCATEs. The application subsystem may use the passthrough feature to perform this communication by having the incoming ALLOCATE routine request the LU_ID for a given LU name. The LU_ID would have been saved by the program that issued the ATTACH_LU. For an example of communicating within the application subsystem, see Appendix E, "Sample Programs."

Supporting Multiple Transaction Programs

APPC/PC assumes a single-tasking operating system in the IBM PC. If the transaction program issues a verb that suspends APPC/PC operation (such as RECEIVE_AND_WAIT if not enough data is available),

APPC/PC normally loops internally, returning to the transaction program only after the verb completes. Even if an interrupt occurs that permits the application subsystem or the transaction program to regain control, issuing a second APPC/PC verb is not valid until the actions resulting from the first have completed. With few exceptions APPC/PC requires serialized issuance of verbs.

This single-threadedness does not prevent the existence of multiple transaction programs.

Two approaches to providing a multiple transaction capability are:

1. Code the transaction programs independently.

   A sophisticated application subsystem can help serialize the verbs from several transaction programs. For example, at appropriate points a transaction program might relinquish control so that other transaction programs can issue APPC/PC verbs. The application subsystem might provide runtime constructs (such as macros) for suspension of a transaction program.

   For designs where control might pass from one transaction program to another while a verb is being executed (for example, time slicing) the transaction programs can determine if APPC/PC is busy executing another verb. The transaction program can then loop on each verb until the verb that the transaction program is ready to issue can be executed. (The return code field contains a value which indicates that a verb has been rejected because APPC/PC was busy. For more information, see Appendix C, "Verb Return Codes.")

2. Code one program that emulates several transaction programs. For example, code a program that issues APPC/PC verbs using several TP_IDs.

With either approach, problems can occur when communicating with a partner that does not support multitasking (such as another instance of APPC/PC). In

this case, using simultaneous multiple transaction programs can lead to a deadlock situation in which each side is blocked from sending because another transaction program is waiting to receive data.

If you choose to develop an application subsystem that can support more than one transaction program simultaneously, you must analyze the communication topology and incoming ALLOCATE management to ensure that no deadlocks will occur.

Deadlock situations between IBM PCs can be caused by an unfinished verb in a transaction program preventing the issuance of a verb in another TP in the same node.

One way of preventing such deadlocks is to specify the RETURN_CONTROL(INCOMPLETE) option on the ATTACH_PU verb. This will cause control to return to the verb issuer with an INCOMPLETE return code if a verb cannot finish. Verbs from other transaction programs can then be issued before the incomplete verb is re-issued.

If you select the RETURN_CONTROL(INCOMPLETE) option of the ATTACH_PU verb, you should consider designing the application subsystem to perform all verb issuances. It can then perform a round-robin scheduling on the various TPs, including periodic re-issuance of incomplete verbs. See "System Deadlocks" on page 10-2 for a more complete explanation of deadlocks.

Using Application Subsystem Exits

Three application subsystem exits are available for specification in the ATTACH_LU verb:

1. An exit for LU-LU security that provides LU-LU passwords for an LU and a named partner LU

2. An exit that accepts errors logged during program execution (There is also an error log exit, primarily for hardware errors, specified on the ATTACH_PU verb.)

3. An exit that manages incoming ALLOCATEs and validates (and loads, if necessary) the requested transaction programs.

Different LUs can use the same or different exit addresses. APPC/PC provides pointers to these asynchronous exits. These pointers give the application subsystem access to the appropriate verb (ACCESS_LU_LU_PW, SYSLOG, or CREATE_TP). Unlike other APPC/PC verbs, these are passed from APPC/PC to the application subsystem.

Supporting LU-LU Passwords (ACCESS_LU_LU_PW Exit)

If LU-LU security is supported, the application subsystem must supply an exit routine for the verb ACCESS_LU_LU_PW. This exit routine provides the password that allows a specified partner LU to establish a session with a local LU. The application subsystem can maintain the passwords in a password table (preferably in encoded form), or the program can request them from the operator as needed. All security measures for passwords are the operator's responsibility. For example, if you keep passwords on hard disk or diskette, consider what physical security is required to maintain confidentiality.

Managing Logged Errors (SYSLOG Exit)

If you wish to log the errors occurring during a conversation, you must provide an exit routine to accept such errors. You cannot issue another APPC/PC verb from this exit routine.

If you specify this error log exit, you may want to print or display a console message, create a file of logged errors, or send an error message to another node. However, if the log message occurs during the processing of an

interrupt for incoming data, and the interrupted code is within DOS, then trying to use DOS to write to the console, a file, or a printer results in a recursive use of DOS, which leads to unpredictable results.

Certain uses of BIOS within this exit may also lead to unpredictable or undesirable results, depending on the use of DOS and BIOS within the application.

APPC/PC provides an interface to suspend operation. While APPC/PC operation is suspended, it will not call the SYSLOG exit (or any other APPC/PC exit). You may choose to have your application subsystem assume control of the DOS X'21', BIOS X'10', and BIOS X'17' (and possibly other) interrupt vectors, and then disable APPC/PC (using the DISABLE/ENABLE_APPC verb) until the DOS or BIOS call is finished. The application subsystem should then regain control and re-enable APPC/PC (using the DISABLE/ENABLE_APPC verb). You can return control to the application subsystem from DOS or BIOS by re-establishing the interrupt vector and issuing your own INT X'21', INT X'10', or INT X'17' (or by simulating the effect of these interrupts). The application subsystem then reassumes control of the interrupts and performs an IRET (Interrupt Return) to the transaction program.

When enabled again, APPC/PC will process any delayed messages it received while it was disabled.

The DOS Load and Execute call (X'4B'), which loads a transaction program and starts it executing, does not return until the transaction program finishes. Therefore, if APPC/PC is disabled at the beginning of this DOS call, you must find an opportunity to re-enable. Either do not disable APPC/PC for this call, or have each transaction program enable APPC/PC when the transaction program starts executing. The section on the sample program (see Appendix E, "Sample Programs") discusses a user-defined TP_INITIATE verb that APPC/PC passes through the application subsystem. Although the sample program does not perform such re-enabling, this may be a convenient place to do it.

If your transaction programs use DOS or BIOS calls then you must either correct the recursion problem, or you should not use DOS or BIOS at the asynchronous exit.

Managing Asynchronous Incoming ALLOCATEs (CREATE_TP Exit)

If you accept incoming ALLOCATEs as they arrive, you must provide an exit to process them.

One way of managing this is to start an application subsystem program that waits (by looping until a switch or counter is set) for an incoming ALLOCATE. When an incoming ALLOCATE arrives the looping program is interrupted by APPC/PC and the CREATE_TP exit is called. The CREATE_TP exit code sets the switch or counter and returns to APPC/PC. When the looping program regains control it ends the loop, then loads the requested transaction program and saves the TP_ID and CONV_ID.

If your program does not support more than one simultaneous transaction program and a transaction program is running when the ALLOCATE arrives, you should either queue the incoming ALLOCATE at the CREATE_TP exit, or reject it.

To support more than one simultaneous transaction, you can choose to pre-load all transaction programs and activate the appropriate one for each incoming request.

Alternatively, you can use DOS to load the transaction programs at the CREATE_TP exit. You must suspend APPC/PC while using DOS to prevent DOS recursion. See "Managing Logged Errors (SYSLOG Exit)" on page 2-10 for a description of DOS recursion.

If the asynchronous exit saves any data for access by another section of the application subsystem, then you should inhibit interrupts around the access code. Otherwise, a hardware interrupt might occur at the point of access, causing unpredictable results.

Canceling a Transaction

You may wish to permit the operator to cancel a transaction by pressing Ctrl-Break on the keyboard. The DOS Control Break option in the APPC/PC menus specifies whether or not APPC/PC should issue a DOS call ("check keyboard status") while it is suspended with no work. For example, APPC/PC has no work while the transaction program is waiting for data after issuing a RECEIVE_AND_WAIT verb. This DOS call does not cause DOS, used in this manner, recursion problems.

If you select this option, the DOS Control Break routine gains control and cancels the transaction program when Ctrl-Break is pressed. However, APPC/PC is still in the middle of processing the RECEIVE_AND_WAIT verb, and cannot be used to process another verb. You can restore APPC/PC to a usable state by issuing the DETACH_PU(TYPE = HARD) verb, which closes all adapters and cleans up APPC/PC conversations. In the environment just described, the operator has to start a utility program that issues DETACH_PU(TYPE = HARD).

You can simplify the procedures for the operator by having the application subsystem substitute its own DOS Control Break routine (DOS interrupt X'23') and then by issuing the DETACH_PU verb from this routine. DETACH_PU overrides any existing verbs, regains outstanding storage, and closes the adapters. The routine should then execute the original DOS Control Break code to cancel the application.

After canceling a transaction, the application subsystem must repeat the ATTACH_PU, ATTACH_LU, ACTIVATE_DLC, and CNOS sequence to redefine APPC/PC and restart the adapters.

If the IBM PC is communicating with a host, the effect of an ungraceful shutdown of lines (and sessions) may result in Network Management messages flowing to a host application, alerting it to a potential problem. To prevent the user from causing these messages to flow to the host,

you should prohibit the Ctrl-Break key from canceling an application. In this case, the application subsystem should substitute its own DOS Control Break routine, which ignores the Ctrl-Break key. If you choose this method, do not select the APPC/PC menu option "DOS Control Break."

Another possibility which may be desirable in some circumstances, is to use the BIOS keyboard Control Break routine (interrupt X'1B') which takes effect immediately upon pressing the Ctrl-Break key. If you choose this method, do not select the APPC/PC menu option "DOS Control Break."

You must consider the total operating environment when choosing your method of cancelation. In particular, you must make sure that other resident programs that use the same interrupt are not affected.

DOS Interrupts

APPC/PC uses software interrupt X'68' to issue verbs. The APPC/PC loader establishes this interrupt vector and the APPCUNLD program resets it. The application subsystem or the transaction program can determine if APPC/PC is loaded by comparing the character string starting at 9 bytes before the address referred to in the X'68' vector (location 416 in storage) to the ASCII character string 'APPC/PC'. If the value matches, APPC/PC is loaded.

Different adapters use different hardware interrupts. The SDLC adapter uses interrupts 3 and 4. The IBM Token-Ring Network adapter uses interrupt 2, 3, 6, or 7 depending on the setting of switch 1 on the adapter card. For more information on this switch setting, see the *IBM Token-Ring Network PC Adapter Guide to Operations.* For more information on configuring APPC/PC for different adapters see the *APPC/PC Installation and Configuration Guide.*

Chapter 3. Developing a Transaction Program

When developing a transaction program, you must choose between certain design alternatives. The following list describes the design issues to consider:

- Choosing either basic or mapped conversations
- Deciding whether to start conversations with or without confirmation
- Choosing a name for the transaction program
- Using the network management verb
- Using the security features
- Providing for conversion of ASCII names and data (if necessary).

The first part of this chapter provides background information on the application protocols, conversation states, the APPC/PC support tasks, and data formats. The remaining parts of this chapter describe specific requirements for operating a transaction program.

Understanding Application Protocols

APPC/PC support enables your program to accomplish program-to-program communication. The design of your program depends on the protocols that you define and the communication that your program must accomplish.

In addition to any rules that you define for your program, APPC/PC defines rules that your program must follow when using a conversation. To enforce these rules, APPC/PC manages the state of your conversation and allows your program to perform certain operations only when the conversation is in the correct state. For example:

- Your program cannot send data unless it has permission to send.
- Your program cannot receive data unless the partner program has permission to send.
- Your program cannot use a conversation after it has been deallocated.

Understanding the Conversation States

APPC/PC manages and enforces the following conversation states:

State	Definition
Reset	The conversation does not exist.
Send	The program can send data, request confirmation, or deallocate the conversation.
Receive	The program can receive information from the partner program.
Confirm	The program can reply to a confirmation request.

Descriptions of state changes, and of the valid verbs that a transaction program can issue in each state, are contained in Chapter 6, "Using Transaction Mapped

Conversation Verbs," and Chapter 7, "Using Transaction Basic Conversation Verbs." Appendix B, "Conversation State Matrices," summarizes the ways the state of a conversation can change.

Understanding the Available APPC/PC Services

APPC/PC support includes a series of services that your transaction program can use to communicate with another program. The following list includes the name of each service followed by a short description of the service and the names of the conversation verbs corresponding to the service:

Allocate a Conversation

Requests that the local LU start a conversation with a transaction program in a partner LU (the partner program).

Corresponding verbs: ALLOCATE and MC_ALLOCATE.

Send Data

Sends data to the partner program.

Corresponding verbs: SEND_DATA and MC_SEND_DATA.

Force Data in the Internal Buffers to be Sent

Forces the LU to send to the partner program all data it is holding in an internal buffer.

Note: You do not normally have to use this service to cause the LU to send the data. The LU automatically sends the data it stores in an internal buffer when the buffer is full or when it determines that your program is done sending.

Corresponding verbs: FLUSH and MC_FLUSH.

Receive Data

Receives data from the partner program.

Corresponding verbs: RECEIVE_AND_WAIT, RECEIVE_IMMEDIATE, MC_RECEIVE_AND_WAIT, and MC_RECEIVE_IMMEDIATE.

Request Permission to Send

Requests from the partner program permission to send data.

Corresponding verbs: REQUEST_TO_SEND and MC_REQUEST_TO_SEND.

Grant Permission to Send

Gives the partner program permission to send data.

Corresponding verbs: PREPARE_TO_RECEIVE and MC_PREPARE_TO_RECEIVE.

Request Confirmation

Requests that the partner program confirm that all the data has been received and processed successfully.

Corresponding verbs: CONFIRM and MC_CONFIRM.

Accept or Reject Confirmation

Sends a reply to a confirmation request.

Corresponding verbs: CONFIRMED, MC_CONFIRMED, SEND_ERROR, and MC_SEND_ERROR.

Request to be Posted when Information is Available

Requests that the LU post an event when the conversation has information available to be received.

Corresponding verb: POST_ON_RECEIPT.

Report an Error

Reports that an error has occurred.

Corresponding verbs: SEND_ERROR and MC_SEND_ERROR.

Obtain Conversation Attributes

Obtains the attributes of a conversation. These attributes include:

- The name of the local LU
- The name of the partner LU

- The mode name of the session's transmission service mode
- The type of confirmation protocols supported by the conversation
- The type of conversation

  Corresponding verbs: GET_ATTRIBUTES, MC_GET_ATTRIBUTES, and GET_TYPE.

Deallocate a Conversation

Ends a conversation with the partner program.

Corresponding verbs: DEALLOCATE and MC_DEALLOCATE.

Choosing Between Basic and Mapped Conversations

APPC/PC supports two types of conversations: basic and mapped. Mapped conversations are for transaction programs that are the final users of the data exchanged. A mapped conversation provides the features required for advanced program-to-program communication in an easy-to-use record-level manner.

Basic conversations are for LU service transaction programs that provide services and exchange data for other transaction programs. A basic conversation has all the features of a mapped conversation but also has additional features that transaction programs may require to provide services for other applications.

APPC/PC support provides separate sets of verbs for basic and mapped conversations. In general, mapped conversation verbs have fewer options and are easier to

use. Basic conversation verbs are more powerful and provide the additional flexibility required for use by an LU service program.

You should consider the following topics when choosing between basic and mapped conversations.

Understanding the Conversation Type

The ALLOCATE verb designates a conversation as either a mapped conversation or a basic conversation. A program can issue only basic conversation verbs for a basic conversation. A program using a mapped conversation normally issues only mapped conversation verbs.

You can provide your own mapped conversation support using only basic conversation verbs for a conversation designated as mapped. If you choose to provide your own mapped conversation support, your program must conform to the mapped conversation formats and protocols.

See the *SNA Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2* for more information on mapped conversation formats and protocols.

The verbs you use must be consistent for the entire conversation. You cannot use basic conversation verbs for some requests and mapped conversation verbs for other requests. APPC/PC rejects the verbs if you change from one type of verb to another within a conversation. A remotely intitated transaction program can issue the GET_TYPE verb to determine the conversation type.

Sending Data

Use a basic conversation when you need to optimize your program's performance by sending the data from a buffer that contains more than one logical record or a partial logical record. You can use this option to improve your program's execution efficiency by enabling it to send several logical records with one request.

To use this basic conversation feature, your program must provide a 2-byte length field (LL field) at the beginning of every logical record where:

- The last 15 bits of the LL field contain a binary value equal to the length of the logical record, including the 2-byte length field. The 15-bit limit restricts the value to a maximum of 32767 (32765 bytes of user data plus the 2-byte length field). If you use a value larger than 32767, APPC/PC does not detect the error. APPC/PC uses the last 15 bits of the LL field even if the value is larger than 32767.

  The smallest value possible is 2 (the LL field followed by no data). If you use a value that is less than 2, APPC/PC indicates an error.

- APPC/PC ignores the first bit of the LL field. This bit is a concatenation indicator. If the concatenation indicator is set, the transaction program must append the data from the following logical record to the data received up to that point. This concatenation process should continue until the transaction program receives a record in which the concatenation indicator is not set. This definition provides for logical records that are longer than 32767 bytes.

- You must manage the reversal of byte values in your IBM PC.

  The IBM PC stores all numeric 16- or 32-bit values with the low-order (least significant) byte stored in the lower numbered address. Therefore, if a transaction program computes the length of a logical message and

stores that value as the LL field, the 2 bytes appear in memory with the low-order byte first, and your IBM PC will send the bytes in this order (incorrectly!) over the communication line.

The transaction program is responsible for putting all transaction-level data, including LL fields, in the correct order (high-order byte first).

Use a mapped conversation if you do not need to send partial logical records, or more than one logical record. When you send data with the mapped conversation verbs, APPC/PC assumes that the buffer contains exactly one complete record. The mapped conversation support automatically provides length fields in the correct byte-reversed order and uses concatenated logical records as needed.

Receiving Data

Use a basic conversation when you need to receive more than one logical record in one buffer. This option can improve your program's execution efficiency by enabling it to receive several logical records with one request (the BUFFER option).

When you use this basic conversation feature, APPC/PC places the logical records in your buffer with the 2-byte LL fields intact. The bytes are reversed from normal IBM PC order. (See "Sending Data" on page 3-8 to find the format of this field.)

Your program must examine the returned fields of the verb to determine if it has received a complete logical record and if so, where the next logical record begins. APPC/PC provides the rest of an incomplete logical record after a subsequent request to receive data.

If you do not need to receive more than one record with a single request, use a mapped conversation. When you receive data with the mapped conversation verbs, APPC/PC ends the receive operation when your program receives a complete record (which may consist of several

concatenated logical records) or when your buffer is full. APPC/PC supplies a return code when it fills your buffer before your program has received a complete logical record.

Your program can receive the rest of the logical record by issuing a subsequent request to receive data. The APPC/PC mapped conversation support removes any length fields and automatically concatenates records as necessary.

Reporting Errors and Abnormal Termination

Use a basic conversation for the following reasons:

- To distinguish between errors detected by your program and errors detected by an application that is using your program
- To distinguish between an abnormal termination caused by your program and one caused by an application using your program.

When reporting an error or when abnormally terminating a conversation with an LU service program, the basic conversation verbs enable you to indicate which program detected the error. When the partner LU reports the error to the partner program with a return code, the value of the return code indicates where APPC/PC detected the error.

If you do not need to distinguish between errors detected by your program and errors detected by other applications, use a mapped conversation. The mapped conversation verbs assume that your program detected the error.

Sending an Error Log Data Record

Use a basic conversation to send a log record when you detect an error or abnormally terminate a conversation. The basic conversation verbs enable you to specify an error log general data stream (GDS) variable when you report an error or abnormally terminate a conversation. APPC/PC sends this log record to the local log and to the partner LU to be recorded in that log. This feature is useful when your program detects a critical or unrecoverable error and you want it to record the event to help determine the problem.

If you send an error log GDS variable, the format of the record must conform to the formats defined by SNA. See *SNA Reference Summary* for more information on the error log GDS variable format.

Use a mapped conversation if you do not need to send a log record when you detect an error or abnormally terminate a conversation. The mapped conversation verbs assume that your program does not need to record error data in the log to help determine the problem.

Abnormally Terminating Because of a Time-out

To indicate that your program has abnormally terminated the conversation because of a time-out condition, use a basic conversation. When abnormally terminating your conversation, the basic conversation verbs enable you to indicate that your program is abnormally terminating the conversation because the partner program has not done the necessary processing in the time allowed. When APPC/PC reports the error to the partner transaction program, the return code value indicates that a time-out condition caused the abnormal termination.

If you do not need to report the cause of an abnormal termination, use a mapped conversation. The mapped conversation verbs assume that your program requested

the abnormal termination because of a critical or unrecoverable error.

Requesting Confirmation

Requesting confirmation is an efficient way to determine that the partner program has received all the data sent so far. If you plan to request confirmation during the conversation, the allocating transaction must indicate this fact when you request the allocation of the conversation.

If you use conversation protocols that do not request confirmation, you should not request the allocation of a conversation supporting confirmation services.

You can write a transaction program that participates in conversations that use confirmation requests and conversations that do not use confirmation requests.

Choosing a Transaction Program Name

When you name a transaction program, choose a name that has a first character with an EBCDIC code greater than an EBCDIC blank (X'40'). Transaction program names containing first characters with EBCDIC codes less than X'40' are reserved for LU service programs that are defined by SNA. Transaction program names can include up to 64 characters.

Using the Network Management Verb

You can send network management information to a host application. In addition, you can log such information locally. The network management verb enables you to report local I/O errors such as disk errors. APPC/PC sends IBM Token-Ring Network errors to the host automatically, provided the IBM PC successfully establishes a connection with the host.

Using the Security Features

APPC/PC provides two types of security function: partner-LU verification and end-user verification. Partner-LU verification is a session-level security protocol that takes place at the time the session is activated. End-user verification is a conversation-level security protocol that takes place at the time a conversation is started. The ATTACH_LU verb specifies the level of security.

Partner-LU Verification (Session-Level Security)

Partner-LU verification is performed by an exchange of security information between the two LUs. This exchange is called LU-LU verification. This level of security is generally required when the communications network is not physically secure. The local LU and the remote LU each provide their password and APPC/PC performs encryption for password verification.

Passwords are established by the application subsystem and are passed to APPC/PC when requested, using the ACCESS_LU_LU_PW verb. LU-LU passwords are

established for each LU pair. It is recommended, but not required, that each LU pair have a unique password.

End-User Verification (Conversation-Level Security)

End-user verification is used to enable the requested LU to verify the identity of the requester before providing access to the requested transaction program and its resources. The security information exchanged can include a user ID and a password. The user IDs provided by conversation-level security can also be used for auditing and accounting purposes.

In conversation-level security the requesting transaction program provides the security information on the ALLOCATE verb and the remote application subsystem performs the verification. (See ALLOCATE and MC_ALLOCATE verb descriptions). If the requesting transaction program has not supplied the correct user ID and password, the remote application system rejects the request.

An intermediate transaction program (one started by another transaction program) that requires conversation-level security may be used to access an additional transaction program that requires conversation-level security. In this case, an already-verified indicator is set in the allocation request for the additional transaction program. The user ID saved from the first request which initiated the intermediate transaction program is automatically supplied in the second request.

Converting between EBCDIC and ASCII

APPC/PC assumes that the interface between it and the transaction program (or the application subsystem) uses EBCDIC characters. Your program may send some data or parameters to a system services control point (SSCP) or partner LU where they will be compared to table values expressed in EBCDIC. These values include the transaction program name, the partner LU name supplied on ALLOCATE, the mode name, the network identifier supplied on the ATTACH_PU verb, the user ID, and the user password.

Incoming names are in EBCDIC, and your program must translate them to ASCII (if the application subsystem stores them in ASCII). Therefore, both the application subsystem and the transaction program must be prepared to perform ASCII-to-EBCDIC conversion.

Do not yield to the temptation to define private protocols between two IBM PCs which avoid this conversion. Private protocols can cause unexpected difficulties and hamper communication with other types of computers. For example, protocol errors or problems with unrecognized names can result from the use of an ASCII "K" (X'4B') on an LU or net name. The EBCDIC period used in a fully qualified name also has the value X'4B'.

The following names must be eight EBCDIC characters long (padded on the right with blanks if the name is less than 8 characters). These names must be Type A symbol strings (uppercase letters; numerics; or the special characters $, #, and @, with the first character not a numeric.)

LU_NAME
MODE_NAME
NET_NAME
PARTNER_LU_NAME

Whether a transaction program needs to translate data depends on private agreement between the partner transaction programs. If your program is communicating with a node that normally uses EBCDIC, you should convert data to EBCDIC as appropriate.

As a convenience, APPC/PC provides a verb (CONVERT) that converts ASCII codes to EBCDIC or EBCDIC codes to ASCII.

Chapter 4. Introduction to APPC/PC Verbs

A transaction program communicates with other transaction programs by issuing *conversation verbs*. Just as conversation verbs provide a mechanism for communication between transaction programs, other verbs communicate requests to the internal components of APPC/PC. For example, these requests control resources, such as conversations and sessions, available to and provided by APPC/PC.

Verb Types

The different types of verbs that APPC/PC recognizes and issues can be classified into the following categories:

- Control verbs
- Mapped conversation verbs
- Basic conversation verbs
- Network management verb
- Other verbs.

Control Verbs

A few verbs, called control verbs, provide a system definition capability. The application subsystem uses these verbs to attach and detach the PU and LUs, to start and end locally initiated transaction programs, and to determine local parameter settings.

In addition, APPC/PC issues three other control verbs (CREATE_TP, SYSLOG, and ACCESS_LU_LU_PW) to

the application subsystem. These verbs request an action from the application subsystem. See Chapter 5, "Using Control Verbs," for detailed descriptions of the control verbs.

Mapped Conversation Verbs

A transaction program uses mapped conversation verbs to exchange arbitrary data records with other transaction programs, in any format on which the programmers agree. See Chapter 6, "Using Transaction Mapped Conversation Verbs," for detailed descriptions of the mapped conversation verbs.

Basic Conversation Verbs

A transaction program uses basic conversation verbs to exchange records containing a 2-byte *length prefix* with other transaction programs.

End-user transaction programs typically use mapped conversations. Service transaction programs typically use only basic conversations. See Chapter 7, "Using Transaction Basic Conversation Verbs," for detailed descriptions of the basic conversation verbs.

Network Management Verb

A transaction program uses the Network Management Verb (TRANSFER_MS_DATA) to provide management information to a network management services function connected to an SNA network. The network management services function uses this information to help manage the network to which this SNA node belongs. See Chapter 8, "Using the Network Management Verb," for a detailed description of the network management verb.

Other Verbs

APPC/PC also supports five miscellaneous verbs:

- CONVERT
- DISABLE/ENABLE_APPC
- SET_PASSTHROUGH
- PASSTHROUGH
- TRACE.

See Chapter 9, "Other APPC/PC Services," for detailed descriptions of these verbs.

Verb Descriptions

The following verb syntax shows the function and information content of the interactions between transaction programs and APPC/PC. The transaction verbs, (the mapped conversation and basic conversation verbs) are described in Chapters 6 and 7 respectively. The user must perform the actual encoding. The following syntax is for descriptive purposes only.

General Description

Each transaction program verb is associated with information relative to its function. This information consists of:

- Supplied parameters
- Returned parameters
- Return codes
- Error conditions associated with each verb.

The rest of this chapter refers to the following sample verb description. The name of the verb occupies the left side of the figure in an area of its own. In the sample, the name of the transaction program verb is "VERB_NAME."

VERB_NAME	Supplied Parameters: VARIABLE (variable) CONSTANT (VALUE_1) (VALUE_2)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters

With each verb are several parameters that the transaction program uses to supply information to APPC/PC. The top area at the right side of the sample verb description identifies each parameter the transaction must supply with that verb. Following each parameter name, the verb description identifies the kind of parameter value expected by APPC/PC. In general, there are two kinds of supplied parameters: parameters that have a variable parameter value, and parameters that have a limited choice of constant values.

The sample verb description includes two supplied parameters named VARIABLE and CONSTANT. VARIABLE is a supplied parameter with a value that the transaction program needs to define.

The other supplied parameter, CONSTANT, has a more limited range of possible values. A constant parameter can take only the values listed in the verb description, in this case VALUE_1 or VALUE_2.

Returned Parameters

In addition to supplied parameters, there is a list of parameters that APPC/PC returns to the transaction program after it finishes executing the verb. These returned parameters provide information about the result of the verb execution, with details to inform the transaction program about possible conversation state transitions, or problems that APPC/PC encountered while executing the verb.

Return Codes

Most transaction program verbs include a *return code* as one of the returned parameters. APPC/PC places a value in the return code that indicates the success or failure of the request.

Detailed parameter descriptions immediately follow each verb description.

Error Conditions

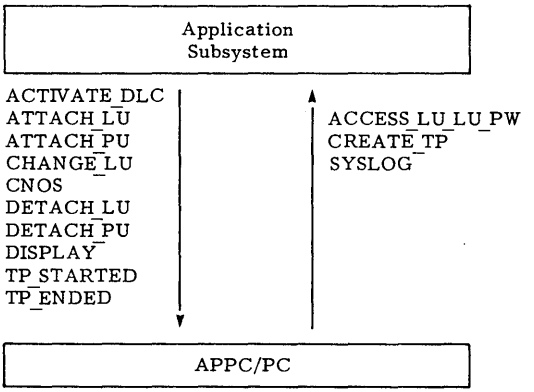
Each of the transaction program verbs recognized by APPC/PC has unique error conditions that can occur during verb execution. The possible error conditions that APPC/PC can encounter during verb execution are identified and discussed in detail in the list of returned parameter descriptions.

Chapter 5. Using Control Verbs

The application interface to APPC/PC includes control verbs the application subsystem used to manage the node. These control verbs do not transfer program data over the communication line, but perform the following operations:

- Define the PU
- Define local LUs (the LU name, potential LU partners, and the parameters defining the sessions between LUs)
- Start a local transaction program
- Process asynchronous events detected by APPC/PC (for example, incoming ALLOCATEs from partner LUs or information for the system log caused by hardware alerts or received log messages).

The diagram below shows the relationship of the control verbs to the application subsystem and APPC/PC. The arrows in this figure show which control verbs the application subsystem sends to APPC/PC and which control verbs APPC/PC sends to the application subsystem.



The rest of this chapter describes the operation of each of the control verbs. See Chapter 4, "Introduction to APPC/PC Verbs," for a description of the different verb types and an explanation of the format used in the verb descriptions.

Understanding Control Verb Return Codes

All control verbs that a program can issue have a parameter called RETURN_CODE that APPC/PC uses to pass a return code back to the program after it executes (or attempts to execute) a verb. The return code indicates the results of verb execution. APPC/PC returns only one return code at a time.

Appendix C, "Verb Return Codes," provides detailed information on all return codes, including the actions you should take. The following return codes can be returned on almost any verb.

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- INCOMPLETE
- OK.

Detailed descriptions of these return codes follow. Brief references to these return codes appear in the individual verb descriptions later in this chapter.

- APPC_ABENDED indicates that APPC/PC has been abnormally terminated.

- APPC_BUSY indicates that APPC/PC is currently executing another verb and cannot execute this verb. This error can occur if a verb is issued after APPC/PC execution is interrupted (for example, by a Ctrl-Break or timer interrupt).

- APPC_DISABLED indicates that APPC/PC is disabled as a result of the DISABLE/ENABLE_APPC verb.

- INCOMPLETE indicates that APPC/PC has not finished executing the verb but is returning control so that you can issue verbs for other transaction programs. The only control verbs that return this code are TP_ENDED and TP_VALID. APPC/PC returns

this indication only if you specify RETURN_CONTROL(INCOMPLETE) on the ATTACH_PU verb.

- OK indicates that APPC/PC executed the verb successfully. That is, APPC/PC performed the function defined for the verb, up to the point at which it returns control to the program.

Application Subsystem to APPC/PC

The application subsystem uses the following verbs to request actions from APPC/PC:

ACTIVATE_DLC

Activates a DLC adapter. The application subsystem must issue an ACTIVATE_DLC verb for each DLC adapter installed in the IBM PC. Before issuing ACTIVATE_DLC verbs, the application subsystem should issue the ATTACH_LU verbs to define the LUs before traffic arrives on the adapter.

ACTIVATE_DLC	Supplied Parameters: DLC_NAME (ITRN) (SDLC) ADAPTER_NUMBER (variable)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

DLC_NAME specifies the name of the adapter that the application subsystem is activating.

- ITRN: specifies the IBM Token-Ring Network adapter.
- SDLC: specifies the IBM Synchronous Data Link Control adapter.

ADAPTER_NUMBER specifies whether the IBM Token-Ring Network adapter is configured as a primary or secondary adapter. Set this value to 0 if the specified adapter is the primary IBM Token-Ring Network adapter or 1 if it is the secondary adapter. The adapter is configured as a primary/secondary adapter if switch 2 on

the adapter card is set to the primary/secondary position. For use with an SDLC adapter, set the ADAPTER_NUMBER value to 0.

Returned Parameters:

RETURN_CODE indicates whether the ACTIVATE_DLC verb is acceptable (0) or not acceptable (non-0). If the verb is not acceptable, APPC/PC does not activate the adapter. The ACTIVATE_DLC return codes are as follows:

- NO_PU_ATTACHED: APPC/PC has not yet received a valid ATTACH_PU verb
- UNRECOGNIZED_DLC: APPC/PC could not find the specified DLC name and number.
- DUPLICATE_DLC: The specified DLC is already activated
- DLC_FAILURE: APPC/PC could not activate the specified DLC.

For detailed information on the following return codes, see "Understanding Control Verb Return Codes" on page 5-3, and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- OK.

ATTACH_LU

Requests APPC/PC to define a local LU with the specified parameters. These parameters include the set of partner LUs (and mode names) for which APPC/PC can build sessions. The application subsystem must use this verb to define all connections to partner LUs.

The LU name, partner LU names, and mode names must each be 8 characters long (padded on the right with blanks if the names are less than 8 characters) using uppercase letters; numerics; or the special characters $, #, and @. (The first character cannot be numeric.)

The LU can have several partners and several modes defined for each partner. You must repeat the PARTNER_LU and PARTNER_SESSION_LIMIT parameters for each partner LU you define relative to the local LU_NAME. In addition, you must repeat the parameters relating to MODE_NAME (MODE_NAME through MODE_MAX_NEGOTIABLE_SESSION_LIMIT) for each mode name defined for the particular PARTNER_LU. The indentation within the following table indicates that you can specify multiple partner LUs for each local LU and multiple mode names (and other parameters) for each partner LU.

ATTACH_LU	Supplied Parameters: LU_NAME (variable) LU_LOCAL_ADDRESS (variable) LU_SESSION_LIMIT (variable) CREATE_TP_EXIT (variable) SYSTEM_LOG_EXIT (variable) LU_LU_PASSWORD_EXIT (variable) MAX_TPS (variable) QUEUE_ALLOCATES (YES) (NO) QUEUE_DEPTH (variable) ..PARTNER_LU_NAME (variable) ..PARTNER_LU_SESSION_LIMIT (variable) ..PARTNER_LU_MAX_MC_SEND_LL (variable) ..PARTNER_LU_DLC_NAME (variable) ..PARTNER_LU_ADAPTER_NUMBER (variable) ..PARTNER_LU_ADAPTER_ADDRESS (variable) ..PARTNER_LU_SECURITY_CAPABILITIES (variable) ....MODE_NAME (variable) ....MAX_RU_SIZE (variable : variable) ....PACING_SIZE (variable) ....MODE_MAX_NEGOTIABLE_SESSION_LIMIT (variable)
	Returned Parameters: RETURN_CODE (variable) LU_ID (variable)
	;

Supplied Parameters:

LU_NAME specifies the local LU's network name (in EBCDIC), that is, the name of the LU as known to the network.

LU_LOCAL_ADDRESS specifies the local SNA (NAU) address of this LU (as defined in VTAM and NCP when connected to a host). This address should be set to zero for LUs that establish only peer-to-peer sessions.

LU_SESSION_LIMIT specifies the maximum number of sessions you want to permit between the local LU and other partner LUs.

CREATE_TP_EXIT specifies the address of the application subsystem code to process incoming ALLOCATEs (in SNA terms, ATTACHes or FMH-5s) from partner LUs. If you do not wish to accept any incoming ALLOCATEs, the application subsystem must set this parameter to all 1s (X'FFFFFFFF'). In this case, APPC/PC rejects any incoming ALLOCATEs.

SYSTEM_LOG_EXIT specifies the address of the application subsystem code to process log messages sent from the partner LU or transaction program, or generated locally from detected protocol errors. If you do not wish to process any of these types of errors, you should set this parameter to all 1s (X'FFFFFFFF').

Warning: You must provide a log exit if you want APPC/PC to provide notification or error information.

LU_LU_PASSWORD_EXIT specifies the address of the application subsystem code to supply the LU-LU password for the given partner LU. If you do not plan to support LU-LU verification security, you should set this parameter to all 1s (X'FFFFFFFF').

MAX_TPS specifies the maximum number of transaction programs that APPC/PC can run under this LU. APPC/PC rejects an incoming ALLOCATE or a TP_STARTED verb if it has already initiated this maximum number.

QUEUE_ALLOCATES specifies whether or not you want the LU to queue incoming ALLOCATEs, and return them one at a time each time the application subsystem issues a GET_ALLOCATE verb.

- YES: Set the CREATE_TP_EXIT parameter to all 0s (X'00000000') to direct the LU to queue incoming ALLOCATEs.

- NO: Set the CREATE_TP_EXIT parameter to the address of the application subsystem code to process incoming ALLOCATEs.

QUEUE_DEPTH specifies the maximum number of incoming ALLOCATEs for the LU to queue. APPC/PC rejects incoming ALLOCATEs that occur when it already has this number of ALLOCATEs in the queue. This parameter applies only if QUEUE_ALLOCATES = YES is also specified.

PARTNER_LU_NAME specifies the network name (in EBCDIC) of the remote LU that the application subsystem is defining as a partner. If the name is less than 8 characters long, the application subsystem must pad it with EBCDIC blanks on the right.

PARTNER_LU_SESSION_LIMIT specifies the maximum number of sessions, across all mode names, you want to permit between the local LU and the specified partner LU. This value should be greater than 1 only if the partner LU supports parallel sessions.

PARTNER_LU_MAX_MC_SEND_LL specifies the maximum size of a logical record (as indicated by the LL field of the logical record) you want the mapped conversation function to construct (used only in a mapped conversation). Use the default of zero, which implies the largest possible value (32767), unless the partner requires a smaller value. The minimum value is 4. A value of 1, 2, or 3 is treated as if it were a 4.

PARTNER_LU_DLC_NAME specifies the name of the DLC used to communicate with the partner LU. This parameter must be the name specified on the APPC/PC

configuration menu for DLC parameters. Use "SDLC" for SDLC operation or "ITRN" for operation with the IBM Token-Ring Network. Use ASCII characters for this name; it is used locally only.

PARTNER_LU_ADAPTER_NUMBER specifies whether the IBM Token-Ring Network adapter used for this partner LU is primary or secondary. The value of this is 0 if the adapter is configured as the primary adapter, and 1 if it is configured as the secondary adapter. The adapter is primary or secondary if switch 2 on the adapter card is set to the primary/secondary position. The parameter must match the specification on the APPC/PC configuration menu for the IBM Token-Ring Network. The value of this parameter must be 0 for SDLC.

PARTNER_LU_ADAPTER_ADDRESS specifies the adapter address of the partner LU. If the partner node is running under APPC/PC on the IBM Token-Ring Network, this value is the one entered in the Local Node Address field of the APPC/PC configuration menus. This address is not used for SDLC operation.

PARTNER_LU_SECURITY_CAPABILITIES specifies the LU-LU verification and conversation security support levels of the partner LU. There are three security functions that you may or may not choose to support:

1. LU-LU Verification Security:

   Indicates that the application subsystem provides the LU-LU password for APPC LU-LU verification. APPC/PC provides the verification logic.

2. Conversation Level Security:

   Indicates that the application subsystem validates the conversation-level password and user ID.

3. Already Verified:

   Indicates automatic acceptance of the incoming ALLOCATE where the partner LU has already verified

the password and user ID (permitted only if the conversation supports conversation level security).

MODE_NAME specifies the mode name (in EBCDIC) designating the network properties (such as the maximum RU size and pacing windows) for these sessions. If the name is less than 8 characters long, the application subsystem must pad the name with EBCDIC blanks on the right. The partner LU must have the same mode name defined.

MAX_RU_SIZE(low:high) specifies the range of permissible maximum RU sizes to be used on sessions with this mode name. When the local LU is the session initiator, APPC/PC tries to use the high value, but the partner LU may negotiate the value to a lower value during session activation. APPC/PC rejects the session activation if the negotiated value falls outside the specified range. When the local LU does not initiate the session, APPC/PC accepts unchanged the incoming maximum RU size value if it falls within the range. If it does not fall within the range, APPC/PC negotiates to the nearest value that is within the range (that is, either the high or low value).

Although the low value can be as low as 16, APPC/PC gains no advantage from values less than 256. You should specify a value less than 256 only if the partner LU gains some efficiency from the smaller value. In the most general environment, you should specify the value according to the following expression:

$$\text{low value} \leq 256 \leq \text{high value}$$

APPC/PC transmits the MAX_RU_SIZE to the remote LU in an encoded form. The formula for this encoding is:

$$a * 2^b \qquad 8 \leq a < 16$$

where "a" is the mantissa and "b" is the exponent of the MAX_RU_SIZE value. APPC/PC rounds down any value that it cannot encode in this form. The following table lists the values that can be encoded in this form.

	Mantissa (a)							
Exponent (b)	8	9	A (10)	B (11)	C (12)	D (13)	E (14)	F (15)
0	8	9	10	11	12	13	14	15
1	16	18	20	22	24	26	28	30
2	32	36	40	44	48	52	56	60
3	64	72	80	88	96	104	112	120
4	128	144	160	176	192	208	224	240
5	256	288	320	352	384	416	448	480
6	512	576	640	704	768	832	896	960
7	1024	1152	1280	1408	1536	1664	1792	1920
8	2048	2304	2560	2816	3072	3328	3584	3840
9	4096							

PACING_SIZE specifies the largest permissible receive pacing window size on sessions of the specified mode name. The largest possible value is 63. A value of 0 implies no pacing (specifying an infinite window size). When the local LU initiates the session, APPC/PC tries to use this value, but the partner LU may negotiate the value to a lower value during session activation.

When the local LU is not the session initiator, APPC/PC accepts unchanged the incoming receive pacing value if it is less than the specified value. If the incoming receive pacing value is greater than the specified value, APPC/PC negotiates the incoming value down to the specified value.

MODE_MAX_NEGOTIABLE_SESSION_LIMIT specifies the maximum number of sessions you want to permit during CNOS negotiation. This parameter sets an upper bound on the maximum number of sessions that can be specified between the local LU and the partner LU on this mode for a locally issued CNOS verb with SET_NEGOTIABLE(NO). This parameter also limits the maximum number of sessions. See "CNOS (Change Number of Sessions)" on page 5-23 for information on the effects of session limits.

Returned Parameters:

RETURN_CODE indicates whether the verb is acceptable (0) or not acceptable (non-0). If the verb is not acceptable, APPC/PC does not define any part of the local LU. To indicate errors detected in the definition of a particular partner LU or mode name, APPC/PC overwrites the erroneous field with dollar signs ($).

See Appendix C, "Verb Return Codes," for more information on return codes.

The ATTACH_LU return codes are as follows:

- NO_PU_ATTACHED: The application subsystem has not yet issued an ATTACH_PU verb.
- ALREADY_ACTIVE_LU: The LU_NAME is defined already.
- BAD_PART_SESS: The session limit for an individual partner LU (PARTNER_LU_SESSION_LIMIT) is greater than the session limit permitted for all partner LUs (LU_SESSION_LIMIT).
- BAD_RU_SIZES: The second value of the MAX_RU_SIZE parameter is smaller than the first value. Reversing the values is the most likely solution for this error.
- BAD_MODE_SESS: The session limit for an individual mode name (MODE_SESSION_LIMIT) is greater than the session limit permitted for all mode names used for sessions with the specific partner LU (LU_SESSION_LIMIT).
- BAD_PACING_CNT: The PACING_SIZE is not between 0 and 63, inclusive.
- EXTREME_RUS: The upper bound for the MAX_RU_SIZE is too large or the lower bound is too small.

- SNASVCMG_1: APPC/PC does not accept "SNASVCMG" as the mode name for a single session connection to communicate data between transaction programs.

- UNRECOGNIZED_DLC: The specified DLC name and ID could not be found in the configuration file.

  For detailed information on the following return codes, see "Understanding Control Verb Return Codes" on page 5-3.

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- OK.

LU_ID specifies the LU identifier. This value is used in subsequent application subsystem and conversation verbs to identify the local LU.

Notes:

1. After the application subsystem defines the local and partner LUs, it must still issue the CNOS verb before APPC/PC permits any incoming session activations or outgoing ALLOCATEs, for both parallel and single session connections.

2. The application subsystem can redefine the connections between the local LU and its partner LU by issuing the DETACH_LU verb, and then supplying the new parameters with another ATTACH_LU verb. The CHANGE_LU verb can redefine only a few of these parameters.

3. A session activation request received from a remote LU might not specify a local LU name. If the application subsystem specifies a null (all EBCDIC blanks) local LU name in an ATTACH_LU verb, APPC/PC directs the session activation request to that local LU. If the application subsystem does not specify a null local LU name on an ATTACH_LU verb,

APPC/PC directs the session activation request to the first local LU defined through an ATTACH_LU verb.

4. When defining an LU that can perform parallel sessions, you should define the "SNASVCMG" (in EBCDIC) mode name for any partner LUs allowing parallel sessions (that is, where the value of the PARTNER_LU_SESSION_LIMIT parameter is greater than 1). If you do not explicitly define this mode, APPC/PC does it implicitly, using default settings for RU size (256), pacing window (1), and session limit maximum (2). APPC/PC subtracts the number of sessions reserved for the "SNASVCMG" mode from the number of sessions allowed by the local-LU and partner-LU session limits when changing the number of sessions with CNOS.

5. You must provide the LU_NAME, PARTNER_LU_NAME, and MODE_NAME parameters in EBCDIC because APPC/PC transmits them to the remote LU. You can use the CONVERT verb to convert these parameters from ASCII to EBCDIC.

ATTACH_PU

Requests APPC/PC to define a local PU with the specified parameters.

ATTACH_PU	Supplied Parameters: PU_NAME (variable) NET_NAME (variable) SYSTEM_LOG_EXIT (variable) RETURN_CONTROL (COMPLETE) (INCOMPLETE)
	Returned Parameters: RETURN_CODE (variable) VERSION (variable) RELEASE (variable)
	;

Supplied Parameters:

PU_NAME specifies the local PU's network name; that is, the name of the PU, as known in its network. APPC/PC returns the PU_NAME on error messages logged to the PU SYSLOG exit. The PU_NAME is not used elsewhere.

NET_NAME specifies the name (in EBCDIC) of the network containing this PU.

SYSTEM_LOG_EXIT specifies the address of the application subsystem code to process log messages destined for the PU; for example, detected protocol or hardware errors, or network management messages. If you do not want APPC/PC to call the application subsystem to process any of these types of messages, set this parameter to all 1s (X'FFFFFFFF').

Warning: You must provide a log exit if you want APPC/PC to provide notification or error information.

RETURN_CONTROL indicates whether to return control for certain verbs to the verb issuer with an INCOMPLETE return code if APPC/PC cannot immediately finish processing the verb. This option enables APPC/PC to execute verbs from other transaction programs to prevent deadlock problems. For more information, see "System Deadlocks" on page 10-2.

- COMPLETE: Specifies that each verb is to be processed to completion.

- INCOMPLETE: Specifies that the verbs that return an INCOMPLETE return code must be re-issued without change. Each verb description shows, under "Returned Parameters", whether INCOMPLETE is a possible return code for that verb.

Returned Parameters:

RETURN_CODE indicates whether the verb is acceptable (0) or not acceptable (non-0). If the verb is not acceptable, APPC/PC does not define any part of the local PU. See Appendix C, "Verb Return Codes," for more information on return codes. The ATTACH_PU return codes are as follows:

- ALREADY_ACTIVE_PU: The PU is already active and cannot be redefined at this time.

  For detailed information on the following return codes, see "Understanding Control Verb Return Codes" on page 5-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- OK.

VERSION specifies the version number of this APPC/PC implementation.

RELEASE specifies the release number of this APPC/PC implementation.

Notes:

1. After initially loading the system or issuing DETACH_PU, the application subsystem must issue the ATTACH_PU verb before issuing any other application subsystem or conversation verbs.

2. You must provide the NET_NAME parameter in EBCDIC because APPC/PC transmits it to the remote LU. You can use the CONVERT verb to convert this parameter from ASCII to EBCDIC.

CHANGE_LU

Alters parameters for an existing local LU.

CHANGE_LU	Supplied Parameters: LU_ID (variable) CREATE_TP_EXIT (variable) SYSTEM_LOG_EXIT (variable) LU_LU_PASSWORD_EXIT (variable) MAX_TPS (variable) QUEUE_ALLOCATES (STOP) (RESUME)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

LU_ID specifies the identifier for the local LU that the application subsystem is altering. APPC/PC returned this value when it identified and initialized the LU using ATTACH_LU. For more information, see "ATTACH_LU" on page 5-7.

CREATE_TP_EXIT specifies the address of the application subsystem code that processes incoming ALLOCATEs from partner LUs. If you do not want the application subsystem to accept any incoming ALLOCATEs, set this parameter to all 1s (X'FFFFFFFF'). In this case, APPC/PC rejects any incoming ALLOCATEs.

The application subsystem must not change the method of handling incoming ALLOCATEs. If the application subsystem specified synchronous management (queueing) of incoming ALLOCATEs in the ATTACH_LU verb (by setting CREATE_TP_EXIT to 0), it must also specify this synchronous management with CHANGE_LU. If the application subsystem specified asynchronous management using the CREATE_TP exit of incoming

ALLOCATEs in the ATTACH_LU verb, then it must specify asynchronous management with CHANGE_LU.

SYSTEM_LOG_EXIT specifies the address of the application subsystem code that processes log messages sent from the partner LU or transaction program, or generated locally as a result of detected protocol errors. If you do not want APPC/PC to call the application subsystem to process these types of errors, set this parameter to all 1s (X'FFFFFFFF').

LU_LU_PASSWORD_EXIT specifies the address of the application subsystem code that is to supply the LU-LU password for the given partner LU. If the session does not support LU-LU verification level security, the application subsystem should set this parameter to all 1s (X'FFFFFFFF').

MAX_TPS specifies the maximum number of transaction programs that APPC/PC can run under this LU. If you try to initiate a transaction program when this maximum number has already been reached, APPC/PC rejects incoming ALLOCATEs and TP_STARTED verbs.

QUEUE_ALLOCATES applies only if the application subsystem specified synchronous management of incoming ALLOCATEs (QUEUE_ALLOCATES = YES with the ATTACH_LU verb).

- STOP specifies that APPC/PC should stop queueing incoming ALLOCATEs, and reject the already queued ALLOCATEs.

- RESUME specifies that APPC/PC should resume queueing incoming ALLOCATEs.

Returned Parameters:

RETURN_CODE indicates whether the verb is acceptable (0) or not acceptable (non-0). If the verb is not acceptable, APPC/PC does not change any part of the local LU definition. See Appendix C, "Verb Return Codes," for more information on return codes. The CHANGE_LU return codes are as follows:

- BAD_LU_ID: APPC/PC does not recognize the specified LU_ID.

- INVALID_CHANGE: The application subsystem has made an invalid change in the management of incoming ALLOCATEs.

  For detailed information on the following return codes, see "Understanding Control Verb Return Codes" on page 5-3, and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- OK.

CNOS (Change Number of Sessions)

Establishes the initial mode session limit for single- or parallel-session connections. You also use the CNOS verb to reset the limit to 0.

The CNOS verb defines a protocol boundary for use by programs that perform LU control functions.

Before the detailed description of the CNOS verb is a discussion of LU-LU sessions and return codes, as well as a summary of the functions that APPC/PC supports.

Understanding LU-LU Session Characteristics

The two characteristics of LU-LU sessions that affect the CNOS verb are:

- The method of connecting the LUs—single session or parallel sessions
- The contention-winner polarities of LU-LU sessions—which LU has priority over the other LU.

A connection between two LUs can be either single or parallel but not both.

For single session connections, APPC/PC limits the number of active sessions between the two LUs to one. That is, an LU cannot activate another session until it deactivates the active session.

The value of the PARTNER_LU_SESSION_LIMIT parameter of the ATTACH_LU verb implicitly specifies parallel- or single-session support.

For parallel-session connections, the application subsystem can group sessions by mode names. A CNOS verb issued by either side causes the corresponding LUs to negotiate a limit for the number of active sessions for the specified mode name group. Additional sessions within

the mode name group can then be activated up to this negotiated limit.

Relating the Mode Name to Network Properties

Each single session or group of parallel sessions has associated with it a set of similar network properties and a mode name that identifies this set of network properties. Transaction programs use the mode name to select the set of network properties it wants to use for a conversation.

The network properties grouped under a single mode name include, for example, the highest synchronization level for conversations on the sessions, the class of service for the sessions, and the session routing and delay characteristics. The correlation of mode names to the sets of network properties is established at system-definition time by the administrator of the network.

Understanding Contention-Winner Polarity

The contention-winner polarity for a session (single or parallel) indicates which of the corresponding LUs has priority when both LUs try to allocate a conversation on the session at the same time. For each single or parallel LU-LU session, only one LU is the contention winner of the session. The other LU is the contention loser of the session.

The contention-winner LU can allocate a conversation on a session without requesting permission from the contention-loser LU. Conversely, the contention-loser LU must request permission from the contention-winner LU to allocate a conversation on the session. The contention-winner LU either grants or rejects the request. The two LUs establish contention-winner polarity of a session when they request APPC/PC to activate a session.

For single sessions, the LU starting the session can request that it be the contention winner or loser. The LU responding to the session activation can accept the

requested polarity or change the polarity, depending on the requested polarity.

If the initiating LU requests that it be the contention winner, the responding LU can accept the polarity or change the polarity making the responding LU the contention winner. If the initiating LU requests that it be the contention loser, the responding LU always accepts the contention-winner polarity.

For parallel sessions, you can further divide each mode-name group of sessions based on contention-winner polarities. You can designate the minimum number of contention-winner sessions for one LU, and all or part of the remaining sessions as the minimum number of contention-winner sessions for the other LU. This partitioning enables two LUs to divide a group of parallel sessions between them to ensure that each LU becomes the contention winner of a minimum number of the sessions.

The LU activating a parallel session can request that it be the contention winner or contention loser. The initiating LU can become the contention winner of a parallel session only if making the initiating LU the contention winner does not infringe on the partner LU's minimum number of contention winners. Otherwise, the initiating LU becomes the contention loser. The LU responding to the activation of a parallel session always accepts the initiating LU's choice of polarity.

Changing the Mode Session Limit

CNOS changes the session limit for a mode group of a given partner LU. This limit is called the **mode session limit**. APPC/PC uses this limit to restrict the number of LU-LU sessions per mode name that are available between two LUs for allocation to conversations. You can use CNOS to change the session limits for both single- and parallel-session connections.

For single sessions, CNOS changes the mode session limit only at the local LU. The remote LU does not take part in processing the change.

For parallel sessions, a CNOS verb issued by either partner changes the mode session limits as well as other CNOS parameters of both LUs. These other CNOS parameters control the minimum number of contention-winner sessions for each LU:

- Which LU is responsible for selecting and deactivating LU-LU sessions when you reset the mode session limit

- Whether or not the LU drains allocation requests when you reset the mode session limits.

If you issue a CNOS verb and parallel session support is specified, APPC/PC initiates and manages a conversation in order to perform negotiation between the two LUs. APPC/PC allocates this conversation on a session with the SNA-defined mode name, SNASVCMG. Therefore, you must first issue a CNOS verb for the SNASVCMG mode to increase the session limit.

The LU that sends the CNOS request is referred to as the *source LU*; the LU that receives the CNOS request is referred to as the *target LU*. Two LUs executing a CNOS verb are considered a CNOS transaction. The role of the LU as a source LU or target LU lasts for the duration of the CNOS transaction. After either side issues the CNOS verb, APPC/PC assumes responsibility for creating and managing the CNOS transaction.

Understanding the CNOS Verb

You must issue this verb to set the mode session limit for either parallel- or single-session connections for a partner-LU/mode name combination before a transaction program issues the first ALLOCATE for that destination. Otherwise, the ALLOCATE fails (ALLOC_FAIL_NO_RETRY).

When setting limits for a parallel session connection, the two LUs negotiate the mode session limits, drain settings, and responsibility values. APPC/PC updates these parameters in the CNOS command to reflect the settings agreed to by both LUs during negotiation. The application subsystem can issue the DISPLAY verb to obtain the negotiated mode session limit values.

No CNOS negotiation occurs when setting the limits for single sessions (that is, the two LUs do not negotiate drain settings or responsibility values). Therefore, you must use another method to coordinate the CNOS parameter settings between partner LUs using a single-session connection.

As part of setting up the initial limits, CNOS also sets the guaranteed (that is, the minimum) number of contention-winner polarities and contention-loser polarities, as well as setting initial activation counts for both polarities. The action of the CNOS verb normally affects only the group of sessions with the specified mode name between the source LU and the target LU. Alternately, with one CNOS command, you can reset the session limits of all modes for a partner LU to 0.

APPC/PC enforces the new mode session limit and contention-winner polarity maximums until one side or the other changes them by issuing a subsequent CNOS verb. However, the partner LU may issue the subsequent CNOS verb, which could mean that the request is invisible at the source API.

After the CNOS verb raises the session limit above 0, it can only reset the limit to 0. It *cannot* set the session

limit to a different non-0 value, and it can not redistribute the number of sessions allocated as the contention winners and losers. Therefore, you cannot change the mode session limits if the partner LU has already set the limits to a non-0 value.

APPC/PC may activate one or more LU-LU sessions with the specified mode name as a result of initializing the session limit. APPC/PC terminates all LU-LU sessions for the specified mode name (or for all mode names) as a result of resetting the session limit to 0. APPC/PC terminates each session as it becomes free and does not interrupt active conversations.

The application subsystem can specify the session limit for use in CNOS negotiations when defining the local LU using the ATTACH_LU verb (see "ATTACH_LU" on page 5-7). You can make this limit higher than the current session limit to handle potential requests from the partner LU for a higher session limit.

You can redefine the mode session limit (but not the local LU session limit or the partner LU session limit) using the CNOS verb. Ifthe SET_NEGOTIABLE parameter value is YES, the corresponding value given in this CNOS verb overrides the MODE_MAX_NEGOTIABLE_SESSION_LIMIT value from the ATTACH_LU verb.

	Supplied Parameters:
CNOS	LU_ID (variable) PARTNER_LU_NAME (variable) MODE_NAME_SELECT (ALL) (ONE, 'SNASVCMG') (ONE, variable) SET_NEGOTIABLE (NO) (YES) PARTNER_LU_MODE_SESSION_LIMIT (variable) MIN_CONWINNERS_SOURCE (variable) MIN_CONWINNERS_TARGET (variable) AUTO_ACTIVATE (variable) RESPONSIBLE (SOURCE) (TARGET) DRAIN_SOURCE (NO) (YES) DRAIN_TARGET (NO) (YES)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

LU_ID specifies the identifier for the local LU for which you are issuing this CNOS verb. APPC/PC returns this value when the application subsystem defines the LU with the ATTACH_LU verb. For more information on ATTACH_LU, see "ATTACH_LU" on page 5-7.

PARTNER_LU_NAME specifies the name of the target LU for which the initialization of session limits and polarities applies.

MODE_NAME_SELECT specifies the mode name for which you are setting or resetting the session limits and polarities.

- **ALL** specifies that the CNOS verb is to change the session limits and polarities for all mode names that apply to the target LU. The single mode name that CNOS does not change when you specify ALL is the SNA-defined mode name, SNASVCMG. You can use the ALL setting for MODE_NAME_SELECT only when resetting session limits to 0.

- **ONE, 'SNASVCMG'** specifies the SNA-defined mode name (in EBCDIC). This mode is only for use by a CNOS transaction when the source LU and target LU are using parallel sessions.

- **ONE, variable** (where **variable** represents a mode name) specifies changes to the session limit and polarities for only the specified mode name.

SET_NEGOTIABLE specifies whether APPC/PC also uses the PARTNER_LU_MODE_SESSION_LIMIT specified in the CNOS verb to override the current settings for MODE_MAX_NEGOTIABLE_SESSION_LIMIT. The application subsystem sets this limit first in the ATTACH_LU verb but a later CNOS verb, with the SET_NEGOTIABLE parameter equal to YES, can override this value. When you specify SET_NEGOTIABLE (YES), a normal CNOS negotiation still takes place. If the CNOS verb contains no parameter errors, the new local negotiation values take effect, even if the partner LU (in the parallel-session case) negotiates the suggested values down.

PARTNER_LU_MODE_SESSION_LIMIT specifies the mode session limit for parallel-session connections. That is, it specifies the maximum number of parallel sessions allowed between the source LU and target LU, for the group of sessions under the specified mode name.

If the mode session limit is currently greater than 0, the value of this parameter must be 0. That is, the CNOS verb can raise the limit above 0, but the next CNOS verb must set the value to 0; you *cannot* change the mode session limit from one non-0 number to another.

When raising the mode session limit above 0, the target LU can negotiate this parameter to a value greater than 0 and less than the specified session limit. The specified session limit, or the negotiated session limit if it is negotiated, becomes the new mode session limit.

The value specified for this parameter must be greater than or equal to the sum of the values specified on the CNOS MIN_CONWINNERS_SOURCE and MIN_CONWINNERS_TARGET parameters.

For single-session connections, the value of this parameter must not be greater than 1.

For the SNASVCMG mode name, the specified mode session limit must be 0, 1, or 2. The specified limit depends on whether you are preparing the SNASVCMG mode for more CNOS commands (one or two) or you are shutting down the mode (0).

If the ATTACH_PU verb specifies RETURN_CONTROL(INCOMPLETE), the mode session limit should be large enough to accommodate all active conversations on the mode for any one transaction program. If the ATTACH_PU verb specifies RETURN_CONTROL(COMPLETE), the mode session limit should be large enough to accommodate all active conversations on the mode for all transaction programs.

MIN_CONWINNERS_SOURCE specifies the minimum number of sessions of which the source LU is guaranteed to be the contention winner. The specified number must be 0 or greater. The specified number, or the negotiated number, becomes the new minimum number of contention-winner sessions for the source LU. The sum of this number and the target LU's new minimum number of contention-winner sessions cannot exceed the new session limit.

When the specified number is greater than half the new session limit (rounded up), the target LU can negotiate this parameter down to half the new session limit. When the specified number is less than or equal to half the new session limit, the target LU cannot negotiate this parameter.

For single-session connections, this parameter specifies the desired contention-winner polarity of the session for the source LU.

For the SNASVCMG mode name with a mode session limit of 2, the specified minimum number of contention-winner sessions for the source LU must be 1. For the SNASVCMG mode name with a mode session limit of 1, the specified minimum number of contention-winner sessions for the source LU must be 0.

This parameter is valid only when the application subsystem is setting the session limit to a non-0 value.

The target LU may negotiate this value for a parallel-session connection; APPC/PC returns the new value to the calling program in this parameter.

MIN_CONWINNERS_TARGET specifies the minimum number of sessions of which the target LU is guaranteed to be the contention winner. The specified number must be 0 or greater. The specified number, or the negotiated number if it is negotiated, becomes the new minimum number of contention-winner sessions for the target LU. The sum of this number and the source LU's new minimum number of contention-winner sessions cannot exceed the new session limit.

The target LU can negotiate this parameter to a number less than or equal to the new session limit minus the new minimum number of contention-winner sessions for the source LU.

For single-session connections, this parameter specifies the desired contention-winner polarity for the target LU.

For the SNASVCMG mode name, the specified minimum number of contention-winner sessions for the target LU must be 1.

This parameter is valid only when the application subsystem is setting the mode session limit to a non-0 value.

AUTO_ACTIVATE specifies the number of contention-winner sessions for APPC/PC to activate automatically, rather than by allocation requests from the transaction program.

Warning: Do not specify AUTO_ACTIVATE if the Link Take-Down option for an SDLC switched line was specified on the APPC/PC configuration menu that sets the parameters for SDLC operation. Specifying both Link Take-Down and AUTO_ACTIVATE can cause thrashing when APPC/PC attempts to activate sessions until the AUTO_ACTIVATE limit is reached.

This parameter is valid only when the application subsystem is setting the mode session limit to a non-0 value. If the parameter is greater than the MIN_CONWINNERS_SOURCE parameter (after any negotiation in the parallel-session connection case), APPC/PC decreases the parameter to the negotiated value.

On a busy IBM Token-Ring Network, this parameter may conflict with the congestion algorithm that APPC/PC uses to terminate unused sessions and links when congestion rises past a certain threshold. Auto-activation by either peer partner may re-establish sessions and links, possibly resulting in a thrashing situation. Therefore, the application subsystem should not specify auto-activation between peer nodes for a large congested network. The congestion algorithm does not apply to a host connection, and you can specify auto-activation for sessions to an SNA network, using a boundary function.

RESPONSIBLE, DRAIN_SOURCE, and DRAIN_TARGET are session-terminating control

parameters. These parameters are valid only when using CNOS to set the mode session limit to 0.

RESPONSIBLE specifies which LU is responsible for deactivating the sessions as a result of resetting the session limit for parallel-session connections. This parameter does not apply to single-session connections or the SNASVCMG sessions. The target LU may negotiate this value.

- **SOURCE** specifies that the source LU is responsible. The target LU cannot negotiate this specification.

- **TARGET** specifies that the target LU is responsible. The target LU can negotiate this specification to SOURCE, in which case the source LU becomes responsible.

Whether an LU deactivates a session immediately after the current conversation or after all queued conversations are complete depends on the DRAIN_SOURCE and DRAIN_TARGET parameters.

- If an LU is to drain its allocation requests, it continues to allocate conversations to active sessions. The responsible LU deactivates a session only when the conversation allocated to the session is deallocated and no request is waiting for allocation to any session with the specified mode name. The allocation of a waiting request takes precedence over the deactivation of a session.

- If an LU is not to drain its allocation requests, the responsible LU deactivates a session as soon as the conversation allocated to the session is deallocated. If no conversation is allocated to the session, the responsible LU deactivates the session immediately.

However, this verb cannot force deallocation of active conversations.

The RESPONSIBLE and MODE_NAME_SELECT parameters are interrelated as follows:

- APPC/PC ignores the RESPONSIBLE parameter for mode names for which the session limit is currently 0 if the application subsystem specifies MODE_NAME_SELECT(ALL).

- If the application subsystem specifies MODE_NAME_SELECT(ONE, variable) with a session limit of 0 and the current session limit for that mode name is already 0, the RESPONSIBLE parameter must specify the same LU (source or target) as is currently responsible for deactivating sessions. You can use this procedure to change the DRAIN option specified in an earlier CNOS command.

DRAIN_SOURCE specifies whether the source LU can drain its allocation requests. For parallel-session connections, the target LU cannot negotiate this parameter. This parameter does not apply to the SNASVCMG sessions.

- **NO** specifies that the source LU cannot drain its allocation requests. APPC/PC rejects all requests currently waiting for allocation, or subsequently issued requests, at the source LU and issues a return code of ALLOCATION_ERROR.

- **YES** specifies that the source LU can drain its allocation requests. The source LU continues to allocate conversations to the sessions until no requests are waiting for allocation, at which time its draining is ended. APPC/PC rejects all allocation requests issued at the source LU with a return code of ALLOCATION_ERROR.

For parallel-session connections, the DRAIN_SOURCE and MODE_NAME_SELECT parameters are interrelated, as follows:

- If the application subsystem specifies MODE_NAME_SELECT(ALL) and DRAIN_SOURCE(YES), APPC/PC ignores the DRAIN_SOURCE parameter for those mode names for which the session limit is currently 0.

- If the application subsystem specifies MODE_NAME_SELECT(ALL) and DRAIN_SOURCE(NO), APPC/PC accepts the DRAIN_SOURCE parameter for all mode names. APPC/PC terminates draining for any mode currently draining its requests.

- If the application subsystem specifies MODE_NAME_SELECT(ONE, variable) and DRAIN_SOURCE(YES) is currently in effect, DRAIN_SOURCE(NO) directs APPC/PC to terminate the draining of the source LU.

- If the application subsystem specifies MODE_NAME_SELECT(ONE, variable) and DRAIN_SOURCE(NO) is currently in effect, the application subsystem must specify DRAIN_SOURCE(NO) again.

DRAIN_TARGET specifies whether the target LU can drain its allocation requests. This parameter does not apply to the SNASVCMG sessions.

For a parallel-session connection, the partner LU may negotiate this value; APPC/PC returns the new value to the calling program in this parameter.

- **NO** specifies that the target LU cannot drain its allocation requests. Requests currently waiting for allocation, or issued later at the target LU, will be rejected. The target LU cannot negotiate this specification for parallel-session connections.

- **YES** specifies that the target LU can drain its allocation requests. The target LU continues to allocate conversations to the sessions until no requests are waiting for allocation, at which time its draining is ended. Allocation requests issued at the target LU after draining ends are rejected. For parallel-session connections, the target LU can negotiate this specification to NO, in which case the target LU cannot drain its allocation requests.

For parallel-session connections, this parameter and MODE_NAME_SELECT are interrelated, as follows:

- If the application subsystem specifies MODE_NAME_SELECT(ALL) and DRAIN_TARGET(YES), APPC/PC ignores the DRAIN_TARGET parameter for the mode names for which the session limit is currently 0.

- If the application subsystem specifies MODE_NAME_SELECT(ALL) and DRAIN_TARGET(NO), APPC/PC accepts the DRAIN_TARGET parameter for all mode names, regardless of the current session limit. Any draining of allocation requests at the target LU is terminated.

- If the application subsystem specifies MODE_NAME_SELECT(ONE, variable) and DRAIN_TARGET(YES) is currently in effect, DRAIN_TARGET(NO) terminates the target LU's draining.

- If the application subsystem specifies MODE_NAME_SELECT(ONE,variable) and DRAIN_TARGET(NO) is currently in effect, the target LU can either accept the DRAIN_TARGET(YES) parameter or negotiate the parameter to NO. After the target LU accepts the DRAIN_TARGET(YES) parameter, it can drain any remaining allocation requests.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. It consists of a 2-byte "primary" code and a 4-byte "secondary" code. See Appendix C, "Verb Return Codes," for more information on return codes. The CNOS return codes are as follows:

- OK: Indicates that APPC/PC executed the CNOS verb successfully.

The following secondary codes indicate whether APPC/PC accepts the parameters as specified in the CNOS verb or as negotiated by the partner LU.

- CNOS_ACCEPTED: APPC/PC accepts the session limits and responsibility as specified.

- CNOS_NEGOTIATED: APPC/PC accepts the session limits and responsibility as negotiated by the partner LU.

- PARAMETER CHECK

  - BAD_LU_ID: APPC/PC does not recognize the value specified for the LU_ID parameter.

  - ALL_MODE_MUST_RESET: APPC/PC does not permit the specification of a non-0 session limit when MODE_NAME_SELECT indicates 'ALL'.

  - BAD_SNASVCMG_LIMITS: The application subsystem has specified invalid settings for the PARTNER_LU_MODE_SESSION_LIMIT, MIN_CONWINNERS_SOURCE, or MIN_CONWINNERS_TARGET parameters when MODE_NAME('SNASVCMG') is indicated.

    The three groups of valid settings are as follows:

    - PARTNER_LU_MODE_SESSION_LIMIT(2)
      MIN_CONWINNERS_SOURCE(1)
      MIN_CONWINNERS_TARGET(1)

    - PARTNER_LU_MODE_SESSION_LIMIT(1)
      MIN_CONWINNERS_SOURCE(0)
      MIN_CONWINNERS_TARGET(1)

    - PARTNER_LU_MODE_SESSION_LIMIT(0)
      MIN_CONWINNERS_SOURCE(0)
      MIN_CONWINNERS_TARGET(0)

  - MINS_GT_TOTAL: The sum of MIN_CONWINNERS_SOURCE and MIN_CONWINNERS_TARGET specifies a number

greater than
PARTNER_LU_MODE_SESSION_LIMIT.

- MODE_CLOSED: CNOS cannot set a non-0 limit because the local maximum negotiable session limit is currently 0 for the specified mode.

- BAD_MODENAME: The specified partner LU does not support the specified mode name.

- RESET_SNA_DRAINS: The SNASVCMG mode does not support the DRAIN settings.

- SINGLE_NOT_SRC_RESP: For a single-session CNOS verb (for which an ATTACH_LU verb was issued with PARTNER_LU_SESSION_LIMIT = 1), APPC/PC permits only the local (source) LU to be responsible for deactivating sessions. Set the RESPONSIBLE parameter to indicate the source LU.

- BAD_PARTNER_LU: APPC/PC does not recognize the specified partner LU name.

- EXCEEDS_MAX_ALLOWED: The local maximum negotiable session limit is less than the session limit specified with the CNOS verb.

- CHANGE_SRC_DRAINS: APPC/PC does not permit you to specify MODE_NAME_SELECT(ONE) and DRAIN_SOURCE(YES) when DRAIN_SOURCE(NO) is currently in effect for the specified mode.

- ALLOCATION_ERROR

  - ALLOCATION_FAILURE_NO_RETRY: APPC/PC cannot allocate the conversation because of a permanent error condition.

  - ALLOCATION_FAILURE_RETRY: APPC/PC cannot allocate the conversation because of a temporary error condition.

- STATE_CHECK

  - CANT_RAISE_LIMITS: APPC/PC does not permit you to set session limits to a non-0 value unless the limits are currently 0.

  - LU_DETACHED: A DETACH_LU verb has reset the definition of the local LU before the CNOS verb tried to specify that LU.

- CNOS_PARTNER_REJECT: The partner LU rejected a CNOS request from the local LU.

  - CNOS_MODE_CLOSED: The local LU cannot negotiate a non-0 session limit because the local maximum session limit of the partner LU is 0.

  - CNOS_BAD_MODENAME: The partner LU does not recognize the specified mode name.

  - CNOS_COMMAND_RACE_REJECT: APPC/PC is currently processing a CNOS verb issued by the partner LU. You should retry the CNOS command later.

  APPC/PC includes an internal CNOS service transaction program that executes CNOS verbs. While using some of the conversation verbs (ALLOCATE, SEND, RECEIVE_AND_WAIT, and DEALLOCATE) to perform CNOS protocol, the CNOS service transaction program may receive any error return code defined for the conversation verb that it issued. In this case, the CNOS verb returns the received return code.

For detailed information on the following return codes, see "Understanding Control Verb Return Codes" on page 5-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED.

Notes (when setting session limits above 0):

1. APPC/PC initializes the mode session limit for a single-session connection to a target LU only at the source LU; no CNOS transaction occurs. The mode session limit for the specified target LU and mode name must be initialized at both the source LU and target LU before either LU can activate the corresponding session. APPC/PC can then activate the session automatically after completing this verb, or in response to an allocation request. In this case, you may want to issue CNOS before you issue ACTIVATE_DLC.

2. Initializing the SNASVCMG mode is the first step in setting up a parallel-session connection to a target LU. The application subsystem must issue the CNOS verb to initialize the mode session limit and contention-winner polarities for the SNASVCMG mode for a target LU before it can initialize any other mode for that target LU. The application subsystem issues the ATTACH_LU verb to define the target LU and issues the CNOS verb to enable a session with the SNASVCMG mode name.

3. The mode session limit and contention-winner polarities for the SNASVCMG mode name must be initialized at both the source and target LUs before either LU can activate the corresponding sessions. APPC/PC can then activate the sessions automatically after completion of this verb, or in response to an allocation request.

4. When initializing parallel-session limits, the CNOS verb operates on groups of sessions with the same mode name. The application subsystem uses the CNOS verb to initialize the limits on the number of active parallel sessions that can exist concurrently within a mode-name group between the source and target LUs. The limits imposed on the number of active parallel sessions within a mode-name group are:

   - The number of active sessions cannot exceed the mode session limit.

- The number of active contention-winner sessions for the source LU cannot exceed the mode session limit minus the minimum number of contention-winner sessions for the target LU.

- The number of active contention-winner sessions for the target LU cannot exceed the mode session limit minus the new minimum number of contention-winner sessions for the source LU.

5. When a CNOS verb specifies the AUTO_ACTIVATE option, the source LU will attempt to activate sessions to conform to the new limits.

   An LU can activate sessions in response to either a CNOS or an ALLOCATE verb according to the following conditions:

   - If the minimum number of contention-winner sessions for the source LU is greater than 0, the source LU can activate contention-winner sessions up to this minimum number.

   - If the sum of the minimum number of contention-winner sessions for both the source LU and the target LU is less than the mode session limit, both LUs can activate additional contention-winner sessions. APPC/PC provides these sessions on a first-come, first-served basis up to the new session limit. The LU can activate these sessions in response to ALLOCATE requests.

   - An LU can activate additional contention-loser sessions up to the mode session limit when activating additional contention-winner sessions does not leave enough sessions to satisfy the minimum number of contention-winner sessions for the partner LU. The LU can activate these contention-loser sessions in response to ALLOCATE requests.

   - Control may return to the user before APPC/PC finishes activating any automatically activated

sessions; that is, when APPC/PC has initiated, but not necessarily completed, the activation process.

Notes (when setting session limits to 0):

1. Only the source LU can reset the mode session limit for a single-session connection to a target LU. In this case, a CNOS transaction does not occur. The source LU deactivates active sessions according to the DRAIN_SOURCE and DRAIN_TARGET parameters.

2. The responsible LU deactivates the group of parallel sessions associated with a specified mode name (other than SNASVCMG) according to the DRAIN_SOURCE and DRAIN_TARGET parameters. If the MODE_NAME_SELECT parameter specifies ALL, the responsible LU deactivates all sessions associated with all modes other than SNASVCMG. The application subsystem must reset the mode session limits and contention-winner polarities for all mode names other than SNASVCMG before issuing the CNOS verb with the SNASVCMG mode name specified.

3. When the application subsystem resets the mode session limit and contention-winner polarities for parallel sessions associated with the SNA-defined mode name (SNASVCMG), APPC/PC resets these parameters only at the source LU. A CNOS transaction does not occur. The source LU deactivates the sessions associated with the SNASVCMG mode name as soon as all other active sessions between the source LU and target LU are deactivated. If no other sessions between the two LUs are active, the source LU immediately deactivates the sessions associated with the SNASVCMG mode name.

4. Control may return to the program that issues the CNOS verb before the LU finishes deactivating the session; that is, when the deactivation process has started but not necessarily completed.

5. You must specify the PARTNER_LU_NAME and the mode name (including SNASVCMG) in EBCDIC. You

can use the CONVERT verb to convert these parameters from ASCII to EBCDIC.

DETACH_LU

Terminates a local LU. Transaction programs must not be using any sessions or conversations attached to this local LU when you issue this verb. You must make sure that all sessions between the named local LU and the partner LUs are first terminated by issuing a previous CNOS verb that sets session limits to 0. After DETACH_LU, APPC/PC does not accept incoming session activation requests for the local LU. Also, APPC/PC will not honor locally issued verbs (such as ALLOCATE) for the specified local LU (that is, the previously assigned LU_ID from ATTACH_LU is now invalid).

DETACH_LU	Supplied Parameters: LU_ID (variable)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

LU_ID specifies the identifier for the local LU you want to deactivate. APPC/PC returned this value when ATTACH_LU identified and initialized the LU. For more information, see "ATTACH_LU" on page 5-7.

If a transaction program issues DETACH_LU before all conversations and sessions for the LU are terminated, a deadlock may occur because the sessions may never be terminated.

The application subsystem may issue DETACH_LU only for an **independent LU**, that is, an LU that is not in a session with an SNA host SSCP.

Returned Parameters:

RETURN_CODE indicates the result of APPC/PC's execution of the DETACH_LU verb. See Appendix C, "Verb Return Codes," for more information

on return codes. The DETACH_LU return codes are as follows:

- BAD_LU_ID: APPC/PC does not recognize the specified LU_ID.

- SSCP_CONNECTED_LU: The LU is in a session with an SNA host System Services Control Point (SSCP). Peer-to-peer sessions have been deactivated but the LU has not been terminated.

  For detailed information on the following return codes, see "Understanding Control Verb Return Codes" on page 5-3, and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- OK.

Note:

APPC/PC terminates all transaction programs still attached to the local LU (that is, APPC/PC generates TP_ENDED automatically).

DETACH_PU

Terminates the local PU. If any LUs are active (not DETACHed), APPC/PC deactivates them through an implicit (internal) invocation of DETACH_LU. For DETACH_PU(TYPE = SOFT), the rules for detaching sessions and local transaction programs associated with the LUs are the same as for DETACH_LU.

DETACH_PU also deactivates the link-level connections for the IBM PC. If you continue issuing verbs, you must first issue the ATTACH_PU, ATTACH_LU, and ACTIVATE_DLC verbs to re-open the communication adapters.

APPC/PC returns control to the program issuing the verb after it terminates all resources of the node.

DETACH_PU	Supplied Parameters: TYPE (HARD) (SOFT)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

TYPE specifies whether APPC/PC should terminate the sessions and conversations associated with the LUs immediately or after the transaction programs terminate their conversations.

- **HARD** directs APPC/PC to stop the sessions and conversations immediately, without waiting for the transaction programs to terminate the sessions and conversations normally. The HARD option may cause APPC/PC to send network management error messages to a network manager (if there is a session with the SSCP at the host). You can use this option when the program is "hung" and the user presses the Ctrl-Break key (for more information see Chapter 2, "Developing an Application Subsystem").

- **SOFT** directs APPC/PC to wait for the transaction programs to terminate their conversations. After all conversations are terminated, APPC/PC terminates the sessions those conversations were using.

Returned Parameters:

RETURN_CODE indicates the result of APPC/PC's execution of the DETACH_PU verb. See Appendix C, "Verb Return Codes," for more information on return codes. The DETACH_PU return codes are as follows:

- NO_PU_ATTACHED: The application subsystem has not yet issued ATTACH_PU to define the PU being detached.

- ADAPTER_CLOSE_FAILURE: APPC/PC has experienced a failure while trying to close an adapter.

  For detailed information on the following return codes, see "Understanding Control Verb Return Codes" on page 5-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- OK.

Notes:

1. The application subsystem can gain control at an interrupt and then issue DETACH_PU (TYPE = HARD) while APPC/PC is processing a conversation verb. In this case, DETACH_PU overrides the current verb, and APPC/PC returns control to the point at which the application subsystem issued DETACH_PU.

   You may also issue DETACH_PU (TYPE = HARD) after APPC/PC has abnormally terminated to shut down the adapter. This action may not succeed for some kinds of abnormal termination.

DETACH_PU (TYPE = HARD) is the only control or transaction verb that APPC/PC will process while it is executing another verb.

2. A short delay may occur after issuing the DETACH_PU verb before the SDLC adapter shuts down.

DISPLAY

Returns the current operating parameter values that control the operation of the local LU. DISPLAY does not cause any information to flow outside the local LU; it reads information from the local LU only.

DISPLAY	Supplied Parameters: LU_ID (variable) PARTNER_LU_NAME (variable) MODE_NAME (variable)
	Returned Parameters: RETURN_CODE (variable) LU_SESSION_LIMIT (variable) PARTNER_LU_SESSION_LIMIT (variable) MODE_MAX_NEGOTIABLE_SESSION_LIMIT (variable) CURRENT_SESSION_LIMIT (variable) MIN_NEGOTIATED_WINNER_LIMIT (variable) MIN_NEGOTIATED_LOSER_LIMIT (variable) ACTIVE_SESSION_COUNT (variable) ACTIVE_CONWINNER_SESSION_COUNT (variable) ACTIVE_CONLOSER_SESSION_COUNT (variable) SESSION_TERMINATION_COUNT (variable) SESSION_TERMINATION_SOURCE_DRAIN (NO) (YES) SESSION_TERMINATION_TARGET_DRAIN (NO) (YES)
	;

Supplied Parameters:

LU_ID specifies the identifier for the local LU for which APPC/PC is to retrieve operating parameters. APPC/PC returns the LU_ID value when the application subsystem defines the LU with the ATTACH_LU verb. (For more information, see "ATTACH_LU" on page 5-7.)

PARTNER_LU_NAME specifies the name of the remote LU for which DISPLAY is to retrieve operating parameters.

MODE_NAME specifies the name of the mode for which DISPLAY is to retrieve operating parameters.

Returned Parameters:

RETURN_CODE indicates whether the verb is acceptable (0) or not acceptable (non-0). See Appendix C, "Verb Return Codes," for more information on return codes. The DISPLAY return codes are as follows:

- BAD_LU_ID: APPC/PC does not recognize the supplied LU_ID parameter value.
- BAD_PART_LUNAME: APPC/PC does not recognize the supplied PARTNER_LU_NAME parameter value.
- BAD_MODENAME: APPC/PC does not recognize the supplied MODE_NAME parameter value.

  For detailed information on the following return codes, see "Understanding Control Verb Return Codes" on page 5-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- OK.

LU_SESSION_LIMIT returns the maximum number of sessions permissible for the local LU, as specified in ATTACH_LU. (For more information, see "ATTACH_LU" on page 5-7.)

PARTNER_LU_SESSION_LIMIT returns the maximum number of sessions permissible between the local LU and the partner LU, as specified in ATTACH_LU. (For more information, see "ATTACH_LU" on page 5-7.)

MODE_MAX_NEGOTIABLE_SESSION_LIMIT returns the maximum number of sessions permissible between the local LU and the partner LU for the designated mode name, as specified in ATTACH_LU or altered by a CNOS transaction. For more information, see "ATTACH_LU" on page 5-7.)

CURRENT_SESSION_LIMIT returns the currently agreed-upon maximum session limit between the local LU and the partner LU for the designated mode name (as specified or negotiated by executing a CNOS verb).

MIN_NEGOTIATED_WINNER_LIMIT returns the minimum number of contention-winner sessions permissible between the local LU and the partner LU for the designated mode name (as specified or negotiated by executing a CNOS verb).

MIN_NEGOTIATED_LOSER_LIMIT returns the minimum number of contention-loser sessions permissible between the local LU and the partner LU for the designated mode name (as specified or negotiated by executing a CNOS verb).

ACTIVE_SESSION_COUNT returns the number of currently active sessions between the local LU and the partner LU for the designated mode name.

ACTIVE_CONWINNER_SESSION_COUNT returns the number of currently active contention-winner sessions between the local LU and the partner LU for the designated mode name.

ACTIVE_CONLOSER_SESSION_COUNT returns the number of currently active contention loser sessions between the local LU and the partner LU for the designated mode name.

SESSION_TERMINATION_COUNT returns the number of currently active sessions that the local LU is responsible for terminating (as the result of a previous CNOS verb).

SESSION_TERMINATION_SOURCE_DRAIN indicates whether the local LU drains the queue of ALLOCATE requests before terminating sessions in response to a CNOS verb issued by either side. For more information, see "Understanding the CNOS Verb" on page 5-27.

SESSION_TERMINATION_TARGET_DRAIN indicates whether the remote LU drains the queue of ALLOCATE requests before terminating sessions as a result of a CNOS verb. For more information, see "Understanding the CNOS Verb" on page 5-27.

Note:

You must specify the PARTNER_LU_NAME and MODE_NAME in EBCDIC. You can use the CONVERT verb to convert these parameters from ASCII to EBCDIC.

GET_ALLOCATE

Requests APPC/PC to dequeue the next queued incoming ALLOCATE if you have specified synchronous management of incoming ALLOCATEs.

Alternatively, the application subsystem can test for queued ALLOCATEs, without dequeuing one.

In both cases, APPC/PC supplies a return code of OK if the queue holds an incoming ALLOCATE, and a return code of UNSUCCESSFUL if the queue does not hold an incoming ALLOCATE.

The application subsystem should issue the GET_ALLOCATE verb only if it specified synchronous management of incoming ALLOCATEs (QUEUE_ALLOCATES = YES in the ATTACH_LU verb).

GET_ALLOCATE	Supplied Parameters: LU_ID (variable) TYPE (DEQUEUE) (TEST)
	Returned Parameters: RETURN_CODE (variable) CREATE_TP_PTR (variable)
	;

Supplied Parameters:

LU_ID specifies the identifier for the local LU that the application subsystem wants to access. APPC/PC returns this value when the application subsystem identifies and initializes the LU by issuing the ATTACH_LU verb. For more information, see "ATTACH_LU" on page 5-7.

TYPE specifies whether APPC/PC is to dequeue a queued incoming ALLOCATE or only test for the presence of an incoming ALLOCATE in the queue.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. See Appendix C, "Verb Return Codes," for more information on return codes. The GET_ALLOCATE return codes are as follows:

- BAD_LU_ID: APPC/PC does not recognize the specified LU_ID.

- GET_ALLOC_BAD_TYPE: APPC/PC does not recognize the parameter specified in the TYPE field.

- UNSUCCESSFUL: The LU is not currently holding any incoming ALLOCATEs in its queue.

  For detailed information on the following return codes, see "Understanding Control Verb Return Codes" on page 5-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- OK.

CREATE_TP_PTR specifies a pointer to a CREATE_TP record identical in format to the record APPC/PC passes to the user exit for incoming ALLOCATEs when you use asynchronous management of incoming allocates. This record contains the TP_ID and CONV_ID of the new transaction program and the new conversation. CREATE_TP_PTR is set only if TYPE = DEQUEUE and RETURN_CODE = OK. For more information see "CREATE_TP" on page 5-66.

TP_ENDED

Notifies APPC/PC of the end of the identified transaction program. APPC/PC responds by performing a DEALLOCATE (TYPE = ABEND_PROG) to free only conversations associated with that program. For more information, see "DEALLOCATE" on page 7-32.

TP_ENDED	Supplied Parameters: TP_ID (variable)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program that you want to terminate. APPC/PC returns this value when the application subsystem initiates the program with the TP_STARTED verb (see "TP_STARTED" on page 5-58) or with the CREATE_TP verb (see "CREATE_TP" on page 5-66).

Returned Parameters:

RETURN_CODE indicates whether APPC/PC accepts the TP_ENDED request. See Appendix C, "Verb Return Codes," for more information on return codes. The TP_ENDED return codes are as follows:

- BAD_TP_ID: APPC/PC does not recognize the supplied TP_ID as an assigned transaction program ID.

  For detailed information on the following return codes, see "Understanding Control Verb Return Codes" on page 5-3, and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED

- INCOMPLETE (possible only if all conversations are not deallocated)
- OK.

Note:

After the application subsystem issues TP_ENDED, the TP_ID is no longer a legal value for any basic or mapped conversation verb.

TP_STARTED

Notifies APPC/PC that the application subsystem has requested resources for a transaction program initiated as a result of a local command, rather than initiated from an incoming ALLOCATE. APPC/PC responds by generating a transaction program identifier (TP_ID) and returning it to the application subsystem. The initiated transaction program must use this TP_ID with all later conversation verbs.

TP_STARTED	Supplied Parameters: LU_ID (variable)
	Returned Parameters: TP_ID (variable) RETURN_CODE (variable)
	;

Supplied Parameters:

LU_ID specifies the identifier for the local LU under which the application subsystem wants to initiate this transaction program. APPC/PC returns this value when the application subsystem identifies and initializes the LU by issuing the ATTACH_LU verb; see "ATTACH_LU" on page 5-7.

Returned Parameters:

TP_ID specifies the identifier for this transaction program. Similarly, APPC/PC also provides a TP_ID for a transaction program started by an incoming ALLOCATE.

RETURN_CODE indicates whether APPC/PC accepts the TP_STARTED request. See Appendix C, "Verb Return Codes," for more information on return codes. The TP_STARTED return codes are as follows:

- BAD_LU_ID: APPC/PC does not recognize the LU_ID.

- TOO_MANY_TPS: APPC/PC is already running the maximum number of transaction programs that can be run concurrently on this LU (as defined with the MAX_TPS parameter in the ATTACH_LU verb).

  For detailed information on the following return codes, see "Understanding Control Verb Return Codes" on page 5-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- OK.

Note:

If an incoming ALLOCATE is in the process of being rejected, it counts against the MAX_TPS parameter until rejection is complete. Therefore, for the TOO_MANY_TPS return code, you may wish to retry the TP_STARTED verb.

TP_VALID

Notifies APPC/PC of the status of an incoming ALLOCATE that the application subsystem has dequeued with the GET_ALLOCATE verb. A valid status indicates that the application subsystem has validated and loaded the new transaction program. The application subsystem must issue a TP_VALID verb before issuing any conversation verbs for the new transaction program.

TP_VALID	Supplied Parameters: TP_ID (variable) CREATE_TP_PTR (variable)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for this transaction program. APPC/PC returns this value in the CREATE_TP structure when the application subsystem uses the GET_ALLOCATE verb. The transaction program must use this identifier when it issues any conversation verbs for APPC/PC to perform.

CREATE_TP_PTR specifies the pointer to the CREATE_TP structure that APPC/PC returned in response to the previous GET_ALLOCATE verb. The application subsystem should specify whether the TP name is valid or invalid in the SENSE CODE field of the CREATE_TP record.

For a list of the valid sense codes, see "CREATE_TP" on page 5-66.

Returned Parameters:

RETURN_CODE indicates whether APPC/PC accepts the TP_VALID request. See Appendix C, "Verb Return Codes," for more information on return codes. The TP_VALID return codes are as follows:

- BAD_TP_ID: APPC/PC does not recognize the TP_ID specified with the TP_VALID verb.

- BAD CONV_ID: APPC/PC does not recognize the CONV_ID in the CREATE_TP record.

- BAD_STATE: TP_VALID does not follow a GET_ALLOCATE.

  For detailed information on the following return codes, see "Understanding Control Verb Return Codes" on page 5-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- INCOMPLETE (possible only if the incoming ALLOCATE is rejected with a non-0 sense code)
- OK.

Note:

APPC/PC, not the transaction program, manages the storage for the CREATE_TP structure; the transaction program should not allocate or free storage for this record.

APPC/PC to Application Subsystem

APPC/PC uses the following verbs to request actions from an application subsystem.

In the descriptions of these verbs, APPC/PC provides the supplied parameters and the application subsystem provides the returned parameters.

ACCESS_LU_LU_PW

Requests the application subsystem to provide APPC/PC with an LU-LU password for a specified partner LU. The application subsystem configures the local LU security capabilities for each partner LU by using the ATTACH_LU verb. The ATTACH_LU verb tells whether the session with a partner LU requires the validation of session-level security. APPC/PC requests an LU-LU password for the validation of session-level security only when both partner LUs are configured to use session-level security.

The application subsystem should maintain system-defined LU-LU passwords for each partner LU in a manner as secure as possible. The application subsystem specifies the user exit routine for the ACCESS_LU_LU_PW call in the ATTACH_LU verb.

Warning: You may not use DOS (or BIOS) functions in this exit unless APPC/PC is disabled when such functions are issued. For more information, see Chapter 2, "Developing an Application Subsystem."

ACCESS_LU_LU_PW	Supplied Parameters: LU_ID (variable) LU_NAME (variable) PARTNER_LU_NAME (variable) PARTNER_FULLY_QUALIFIED_LU_NAME (variable)
	Returned Parameters: PASSWORD_AVAILABLE (YES) (NO) PASSWORD (variable)
	;

Supplied Parameters:

LU_ID indicates the identifier of the local LU for which APPC/PC is processing the session security. APPC/PC returns this value when the application subsystem identifies and initializes the LU by using the ATTACH_LU verb. (See "ATTACH_LU" on page 5-7 for more details.)

LU_NAME indicates the local LU name for which APPC/PC is processing the session security.

PARTNER_LU_NAME indicates the partner LU name for which the application subsystem should return an LU-LU password.

PARTNER_FULLY_QUALIFIED_LU_NAME indicates the fully qualified name of the LU for which the transaction program should return an LU-LU password. The fully qualified name consists of the name of the network (if any), a period (EBCDIC X'4B'), and the LU_NAME. If the network name is blank the period is omitted and the fully qualified name is the same as the LU name.

Returned Parameters:

PASSWORD_AVAILABLE specifies whether an LU-LU password is available for this LU-LU session.

PASSWORD specifies the LU-LU password that the application subsystem returns for the specified partner LU. The LU-LU password must be a byte string consisting of 8 bytes of binary values 0 through 255. If the password is shorter than 8 bytes, the application subsystem must pad this field to the right with 0s.

Notes:

1. APPC/PC, not the application subsystem, manages storage for ACCESS_LU_LU_PW; the application subsystem should not allocate or free storage for this record.

2. APPC/PC supplies all names exactly as you specified them (in EBCDIC). You can use the CONVERT verb to convert EBCDIC names to ASCII.

CREATE_TP

Notifies the application subsystem that APPC/PC has received an incoming ALLOCATE, and requests that the application subsystem initiate a new transaction program and attach it to the conversation being requested by the incoming ALLOCATE. APPC/PC assigns a TP_ID for the newly initiated transaction program and a CONV_ID identifier for the conversation.

If the application subsystem does not accept the parameters of the conversation (such as the transaction program name or synchronization level), it returns a non-0 value in the SENSE_CODE parameter of the verb. This non-0 value indicates that APPC/PC should reject the incoming ALLOCATE with that sense code (sent to the partner LU).

Warning: You may not use DOS (or BIOS) functions in this exit unless APPC/PC is disabled when such functions are issued by the transaction program. For more information, see Chapter 2, "Developing an Application Subsystem."

CREATE_TP	Supplied Parameters: LU_ID (variable) TP_ID (variable) PARTNER_FULLY_QUALIFIED_LU_NAME (variable) PARTNER_LU_NAME (variable) MODE_NAME (variable) CONV_ID (variable) TPN (variable) TYPE (BASIC_CONVERSATION) (MAPPED_CONVERSATION) SYNC_LEVEL (NONE) (CONFIRM) PASSWORD (variable) USER_ID (variable) ALREADY_VERIFIED (YES) (NO)
	Returned Parameters: SENSE_CODE (variable) ERROR_DATA_LENGTH (variable) ERROR_DATA (variable)
	;

Supplied Parameters:

LU_ID indicates the identifier for the local LU under which the application subsystem should initiate the transaction program. APPC/PC returns this value when the application subsystem identifies and initializes the LU with the ATTACH_LU verb. (See "ATTACH_LU" on page 5-7 for more details.)

TP_ID indicates the identifier for the new instance of the transaction program.

PARTNER_FULLY_QUALIFIED_LU_NAME indicates (if known) the fully qualified name of the LU where the remote transaction program is located. The fully qualified name consists of the network identifier, a period (EBCDIC X'4B'), and the LU_NAME. If the network name is blank the period is omitted and the fully qualified name is the same as the LU name.

PARTNER_LU_NAME indicates the name of the remote LU from which the request came. For allocating a session, the local LU knows the remote LU by this name; that is, the PARTNER_LU_NAME is the LU network name.

MODE_NAME indicates the mode name for the session on which the application subsystem received the ALLOCATE.

CONV_ID indicates the identifier of the new conversation.

TPN indicates the name of the transaction program that the application subsystem is to initiate and connect at this end of the conversation.

TYPE indicates the type of conversation that the application subsystem is to allocate.

- **BASIC_CONVERSATION** indicates the allocation of a basic conversation.
- **MAPPED_CONVERSATION** indicates the allocation of a mapped conversation.

SYNC_LEVEL indicates the synchronization level that the remote program indicated for this conversation.

- **NONE** indicates that the programs do not perform confirmation processing on this conversation. The programs do not issue verbs and do not recognize any returned parameters relating to the synchronization function.

- **CONFIRM** indicates that the programs can perform confirmation processing on this conversation. The programs can issue verbs and recognize returned parameters relating to confirmation.

PASSWORD indicates the conversation-level security password that the remote transaction program provided to access the specified local transaction program.

USER_ID indicates the conversation-level user ID that the remote program provided to identify itself.

ALREADY_VERIFIED indicates whether the partner LU has already validated the conversation-level password and user ID.

Returned Parameters:

SENSE_CODE specifies the value that indicates to APPC/PC whether the initiation request is acceptable (0) or not acceptable (non-0). If the application subsystem does not accept the request, it does not initiate a new instance of the specified transaction program. APPC/PC does not perform any validity checking of the SENSE_CODE to guarantee that it is a legal sense code, it only returns the sense code to the partner LU. The currently defined APPC sense codes are:

- OK (X'00000000')

- TP_NAME_NOT_RECOGNIZED(X'10086021'): The application subsystem does not recognize the transaction program name. Although the application subsystem may actually recognize the transaction program name, it may not be able to initiate the transaction program using the designated partner LU or mode name.

- TP_NOT_AVAIL_RETRY(X'084B6031'): The TPN exists but is temporarily unavailable. The remote program can try again later.

- TP_NOT_AVAIL_NO_RETRY(X'084C0000'): The TPN exists but is permanently unavailable. The

remote program cannot try again later without further operator action.

- SYNC_LEVEL_NOT_SUPPORTED(X'10086041'): The remote program specified an unrecognized or unacceptable SYNC_LEVEL type for this transaction program.

- CONVERSATION_TYPE_MISMATCH(X'10086034'): The remote program specified an unrecognized or unacceptable conversation type for this transaction program.

- SECURITY_NOT_VALID(X'080F6051'): The application subsystem rejects the password or user ID, or the local transaction program requires security but the remote program did not supply a user ID.

ERROR_DATA_LENGTH specifies the length of the error data APPC/PC adds to the error notification it sends to the partner LU.

ERROR_DATA specifies the data that APPC/PC adds to the error notification it sends to the partner LU, if the parameters and characteristics for this initiation request are unacceptable to the application subsystem. Note that this data *must* be an appropriately formatted error log data GDS variable, with the correct GDS ID. (For more information, see "GDS Variables" in the *SNA Reference Summary*.) APPC/PC does not check the validity of this data, but if the application subsystem improperly forms this variable, the partner LU may deactivate the session.

State Changes (when SENSE_CODE indicates OK):

APPC/PC places the conversation in receive state for the local transaction program to receive information from the remote partner program.

Notes:

1. APPC/PC does *not* permit the transaction program to start issuing verbs from within the CREATE_TP exit routine. The exit code of the application subsystem

must verify the parameters of the initiation request and save all appropriate information. The application subsystem can then use this information to initiate the transaction program later.

2. After the application subsystem initiates the transaction program, the conversation enters receive state.

3. APPC/PC, not the application subsystem, manages the storage for the CREATE_TP structure; the application subsystem should not allocate or free the storage for this structure.

4. You must specify the PARTNER_FULLY_QUALIFIED_LU_NAME, PARTNER_LU_NAME, MODE_NAME, TPN, PASSWORD, and USER_ID in EBCDIC. You can use the CONVERT verb to convert these parameters from ASCII to EBCDIC.

SYSLOG

Notifies the application subsystem that APPC/PC detected an error or that another program detected an error and informed APPC/PC. These types of errors include a DLC-detected transmission error, abnormal session terminations resulting from protocol errors detected at either the local or remote LU, and a locally or remotely detected conversation-level error. The application subsystem processes and handles these errors by reporting the errors to the human operator, saving the information for later analysis, or discarding the information.

Errors detected in an LU use the system log exit specified in the ATTACH_LU issued for that LU. Errors detected in the PU and network management messages (NMVTs) use the system log exit specified in ATTACH_PU.

Warning: You may not use DOS (or BIOS) functions in this exit unless APPC/PC is disabled when such functions are issued by the transaction program before issuing the macro. For more information, see Chapter 2, "Developing an Application Subsystem."

	Supplied Parameters:
SYSLOG	PU_OR_LU_NAME (variable)
	TP_ID (variable)
	CONV_ID (variable)
	TYPE (variable)
	SUBTYPE (variable)
	DATA_LENGTH (variable)
	DATA (variable)
	ADDITIONAL_INFO (variable)
	;

Supplied Parameters:

PU_OR_LU_NAME (in EBCDIC) incidates the name of the local LU or PU that detected the error.

TP_ID indicates the identifier for the transaction program, if any, that detected the error. APPC/PC passes the TP_ID to the application subsystem when the application subsystem initiates the program using the TP_STARTED verb (see "TP_STARTED" on page 5-58) or using the CREATE_TP verb (see "CREATE_TP" on page 5-66).

CONV_ID indicates the identifier of the conversation, if any, on which APPC/PC detected the error.

TYPE identifies the major category of error detected. For a list of type codes, see Appendix D, "SYSLOG Type Codes."

SUBTYPE identifies the specific error type within the major TYPE category. For locally or remotely detected session errors or conversation-level errors, the SUBTYPE contains the specific sense code. For a list of the subtype codes, see Appendix D, "SYSLOG Type Codes."

DATA_LENGTH indicates the length of the data conveyed with the error notification.

DATA is any character data conveyed along with the error notification. APPC/PC passes data associated with a conversation-level error to the application subsystem using this parameter.

ADDITIONAL_INFO identifies a pointer to additional information about the error. For more detail, see "Understanding Link Error Data" on page D-9.

Notes:

1. Not all parameters are available for all types of error notifications. If a TP_ID, CONV_ID, or SUBTYPE is unavailable in a particular log message, the parameter value is 0. For example, if APPC/PC detects a DLC

hardware error, the PU_OR_LU_NAME contains the PU name, but the TP_ID and CONV_ID parameters contain a value of 0 to indicate that the application subsystem should ignore them.

2. The DATA and ADDITIONAL_INFO parameters are meaningful for certain types of SYSLOG. Appendix D, "SYSLOG Type Codes," identifies these types.

3. APPC/PC directs abnormal termination logs only to the PU system log exit.

4. APPC/PC, not the application subsystem, manages storage for SYSLOG; the application subsystem should not allocate or free the storage for this structure.

5. You must specify the PU_OR_LU_NAME in EBCDIC. You can use the CONVERT verb to convert this parameter value from ASCII to EBCDIC.

Activating and Deactivating a Node

The following procedures and examples describe how to activate and deactivate an APPC/PC node.

Activating a Node

Follow these steps to prepare for transaction program conversations and then activate a transaction program.

1. Define the local PU using ATTACH_PU.

2. Define local LUs and their potential partner LUs using ATTACH_LU.

3. Activate the DLCs using ACTIVATE_DLC.

4. If you are using an SDLC switched connection, dial the number to establish the connection. If you are issuing a CNOS verb for a single session, you do not need to dial until you are ready to start the conversation. You must dial before issuing CNOS if you are using parallel sessions or you have specified the AUTO_ACTIVATE option on the CNOS verb.

5. Use CNOS to set the mode session limit (for a specified partner LU and mode) so that there are enough sessions to satisfy session activation requests (that is, both incoming and local session activation requests).

   For parallel sessions, use CNOS to set the mode session limit for the SNASVCMG mode so that the corresponding LUs can perform CNOS negotiation. Next, set the mode session limit for the mode you want to use for a conversation. The application subsystem must raise both single- and parallel-session limits from 0.

6. Initiate a transaction program resulting from an incoming ALLOCATE or start a local transaction

program by issuing the TP_STARTED verb from the application subsystem. Use the passthrough verbs to set up the transaction program (For more information see "Communicating Identifiers," on page 2-6).

7. Issue conversation verbs from within the transaction program.

Example: Activating a Simple Configuration

The following examples show the configuration of a single LU (APPCLU1) with a single partner LU (APPCLU2) at ring address ADDR2. An IBM Token-Ring Network connects these LUs, which use only a single session with a mode name of APPCMODE.

```
ATTACH_PU
   PU_NAME (APPCPU)
ATTACH_LU
   LU_NAME (APPCLU1)
   LU_SESSION_LIMIT (1)
     PARTNER_LU_NAME (APPCLU2)
     PARTNER_LU_SESSION_LIMIT (1)
     PARTNER_LU_DLC_NAME (ITRN)
     PARTNER_LU_ADAPTER_NUMBER(0)
     PARTNER_LU_ADAPTER_ADDRESS(ADDR2)
       MODE_NAME (APPCMODE)
       MODE_MAX_NEGOTIABLE_SESSION_LIMIT (1)
   LU_ID (LU_ID)     /*** RETURNED ***/
ACTIVATE_DLC
   DLC_NAME (ITRN)
```

At this point the partner LUs are defined and the adapter is open. Communication to partner LU APPCLU2 can begin.

```
CNOS
   LU_ID (LU_ID)     /*** FROM ATTACH_LU ***/
   PARTNER_LU_NAME (APPCLU2)
   MODE_NAME_SELECT (ONE, APPCMODE)
   PARTNER_LU_MODE_SESSION_LIMIT (1)
```

If the transaction is to be locally initiated, enter the name of the transaction program at the DOS prompt or at the prompt displayed by the application subsystem. The transaction program then uses the passthrough verbs to

direct the application subsystem to issue the TP_STARTED verb.

Note:

You may want to issue a single-session CNOS verb before the ACTIVATE_DLC verb to ensure that both sides have raised their session limits before you attempt to start conversations.

Deactivating a Node

Follow these steps to terminate a transaction program, release the APPC/PC conversations allocated to this program, detach the PU, and unload APPC/PC.

1. After all conversations have ended, issue TP_ENDED to indicate the termination of the transaction program.

2. Reset the mode session limits to 0 using the CNOS verb. Resetting these limits causes APPC/PC to deactivate the sessions used by the transaction program for conversations.

3. Undefine the local PU using the DETACH_PU(TYPE=SOFT) verb.

4. Enter APPCUNLD at the DOS prompt to unload APPC/PC.

Example: Deactivating a Simple Configuration

```
TP_ENDED
   TP_ID (TP_ID)     /*** FROM TP_STARTED ***/
CNOS
   LU_ID (LU_ID)     /*** FROM ATTACH_LU ***/
   PARTNER_LU_NAME (APPCLU2)
   MODE_NAME_SELECT (ONE, APPCMODE)
   PARTNER_LU_MODE_SESSION_LIMIT (0)
DETACH_PU
   TYPE(SOFT)
```

Chapter 6. Using Transaction Mapped Conversation Verbs

This chapter describes the category of verbs called *mapped conversation verbs*. Mapped conversation verbs define the Mapped Conversation API for end-user program-to-program support. In particular, the mapped conversation protocol is for use by transaction programs.

Before the detailed descriptions of the mapped conversation verbs is a discussion of the conversation states and common error codes at the mapped conversation protocol boundary and a description of the mapped conversation return codes and abnormal termination conditions. These subjects apply generally to all the mapped conversation verbs.

Understanding Mapped Conversation States

The selection of verbs that a program can issue for a particular mapped conversation depends on the state of the mapped conversation. As the program issues verbs, the state of the mapped conversation can change. This state change is a result of the function of the verb, a verb issued by the remote program, or of network errors.

APPC/PC defines the state of a mapped conversation in terms of the local program's view of the local end of the mapped conversation. The states of other mapped conversations allocated to the program can be different. For example, one mapped conversation can be in receive state and another in send state, concurrently.

The state of the mapped conversation determines the verbs that APPC/PC allows a program to issue. The table below correlates the verbs, and parameters if applicable, to the mapped conversation states.

Verb	Conversation States at Mapped Conversation Protocol Boundary			
	Reset	**Send**	**Re-ceive**	**Con-firm**
MC_ALLOCATE	Yes	n/a	n/a	n/a
MC_CONFIRM	n/a	Yes	No	No
MC_CONFIRMED	n/a	No	No	Yes
MC_DEALLOCATE with TYPE(FLUSH) or TYPE(SYNC_LEVEL)	n/a	Yes	No	No
MC_DEALLOCATE with TYPE(ABEND)	n/a	Yes	Yes	Yes
MC_FLUSH	n/a	Yes	No	No
MC_GET_ATTRIBUTES	n/a	Yes	Yes	Yes
GET_TYPE	n/a	Yes	Yes	Yes
MC_PREPARE_TO_RECEIVE	n/a	Yes	No	No
MC_RECEIVE_AND_WAIT	n/a	Yes	Yes	No
MC_RECEIVE_IMMEDIATE	n/a	No	Yes	No
MC_REQUEST_TO_SEND	n/a	No	Yes	Yes
MC_SEND_DATA	n/a	Yes	No	No
MC_SEND_ERROR	n/a	Yes	Yes	Yes
MC_TEST	n/a	Yes	Yes	Yes

At the intersection of each verb row and state column, the table indicates Yes, No, or n/a. **Yes** means that APPC/PC allows the program to issue the verb when the mapped conversation is in that state.

No means the program cannot issue the verb because APPC/PC disallows the verb in that state. APPC/PC treats a verb issued for a mapped conversation in a disallowed state as a state-check condition. The individual verb descriptions list the applicable state-check conditions.

n/a means the state is not applicable either because it cannot exist when the verb is issued or because the state is not relevant to the verb.

A mapped conversation enters a particular state when the program issues a verb that causes a state transition or when the program receives a returned value that indicates a state transition has occurred. This chapter defines the specific state transitions in the individual verb descriptions under the heading "State Changes" in the return code descriptions and under the following heading, "Understanding Mapped Conversation Return Codes."

Understanding Mapped Conversation Return Codes

All conversation verbs have a parameter called RETURN_CODE that APPC/PC uses to pass a return code back to the transaction program after the LU finishes executing a verb. The return code indicates the results of verb execution, including any state changes to the specified mapped conversation. For information on which verbs a program can issue in each state, see "Understanding Mapped Conversation States" on page 6-1.

The structure of a RETURN_CODE parameter is a 2-byte primary code identifying the error type, and a 4-byte secondary code which provides more detailed error information.

Some of the return codes indicate results of the local LU's processing of a verb; APPC/PC returns these return codes with the verb that initiated the local processing. Other return codes indicate results of processing initiated at the remote end of the mapped conversation.

Depending on the verb, APPC/PC returns these return codes with the verb that initiated the remote processing or with a subsequent verb. Still other return codes report events occurring at the remote end of the mapped conversation. In any case, APPC/PC returns only one return code at a time.

Appendix C, "Verb Return Codes" provides detailed information on all return codes, including the actions you should take.

The following return codes can be returned on one or more mapped conversation verbs:

ALLOCATION_ERROR
APPC_ABENDED
APPC_BUSY
APPC_DISABLED
CONV_FAILURE_NO_RETRY
CONV_FAILURE_RETRY
CONVERSATION_TYPE_MIXED
DEALLOCATE_ABEND
DEALLOCATE_NORMAL
INCOMPLETE
INCOMPLETE_ALTERED_VERB
INVALID_VERB
OK
PROG_ERROR_NO_TRUNC
PROG_ERROR_PURGING.

Detailed descriptions of these return codes follow. Brief references to these return codes appear in the individual verb descriptions later in this chapter.

ALLOCATION_ERROR indicates that the local transaction program issued an MC_ALLOCATE and APPC/PC could not complete the allocation of the specified mapped conversation. The ALLOCATION_ERROR indication and one of the following secondary return codes form the complete return code that APPC/PC returns to the transaction program; the secondary return code identifies the specific error.

The remote LU and remote transaction program referred to in the following secondary return code definitions are the LU specified in the PARTNER_LU_NAME parameter and the transaction program specified in the TPN parameter, respectively, of the MC_ALLOCATE verb.

APPC/PC reports allocation errors caused by the local LU failing to obtain a session for the conversation on the

MC_ALLOCATE verb, and by the remote LU rejecting the allocation request on a subsequent verb.

Whenever APPC/PC returns an ALLOCATION_ERROR return code to the program, it places the mapped conversation in reset state.

The ALLOCATION_ERROR secondary return codes are:

- **ALLOCATION_FAILURE_NO_RETRY** indicates that APPC/PC cannot allocate the mapped conversation on a session because of a permanent condition. For example, APPC/PC cannot activate the session to be used for the mapped conversation because the current mode session limit for the specified partner LU is 0; or because of a system definition error or a session-activation protocol error; or because APPC/PC deactivated the session in response to a session protocol error before it could allocate the mapped conversation. The transaction program should not try the conversation again until it corrects the condition.

- **ALLOCATION_FAILURE_RETRY** indicates that APPC/PC cannot allocate the mapped conversation on a session because of a temporary condition. For example, APPC/PC cannot allocate a session for the mapped conversation because of a temporary lack of resources at the local LU or remote LU, or because APPC/PC deactivated the session due to a line or modem failure before it could allocate the mapped conversation.

  The condition is temporary; the transaction program can try the conversation again. However, to avoid congesting the network with attempted allocation requests, the transaction program should pause or wait for a keystroke before retrying the conversation.

- **CONVERSATION_TYPE_MISMATCH** indicates that the remote LU rejects the allocation request because either it does not support mapped conversations, or the remote transaction program does not support the mapped conversation protocol

boundary. APPC/PC returns this return code on a subsequent verb.

- **PIP_NOT_ALLOWED** indicates that the remote LU rejects the allocation request because the local program specified program initialization parameters (by setting the PIP_DATA_LENGTH parameter to a non-0 value) and either the remote LU does not support PIP data, or the remote transaction program has no PIP variables defined. APPC/PC returns this return code on a subsequent verb.

- **PIP_NOT_SPECIFIED_CORRECTLY** indicates that the remote LU rejects the allocation request because the remote transaction program has one or more PIP variables defined and the local transaction program has specified that there are no program initialization parameters (by setting the PIP_DATA_LENGTH parameter to 0).

  This error can also indicate that the local transaction program has specified program initialization parameters (by setting the PIP_DATA_LENGTH parameter to a non-0 value) that do not correspond in number to those defined for the remote transaction program. APPC/PC returns this return code on a subsequent verb.

- **SECURITY_NOT_VALID** indicates that the remote LU rejects the allocation request because the access security information (specified using the MC_ALLOCATE SECURITY parameter) is invalid. APPC/PC returns this return code on a subsequent verb.

- **SYNC_LEVEL_NOT_SUPPORTED** indicates that the remote LU rejects the allocation request because the local transaction program specified a synchronization level (using the SYNC_LEVEL parameter) that the remote transaction program does not support. APPC/PC returns this return code on a subsequent verb.

- **TPN_NOT_RECOGNIZED** indicates that the remote LU rejects the allocation request because the local transaction program specified a remote transaction program name that the remote LU does not recognize. APPC/PC returns this return code on a subsequent verb.

- **TRANS_PGM_NOT_AVAIL_NO_RETRY** indicates that the remote LU rejects the allocation request because the local transaction program specified a remote transaction program that the remote LU recognizes but cannot start. The condition is permanent; the transaction program should not try the conversation again. APPC/PC returns this return code on a subsequent verb.

- **TRANS_PGM_NOT_AVAIL_RETRY** indicates that the remote LU rejects the allocation request because the local transaction program specified a remote transaction program that the remote LU recognizes but currently cannot start. The condition is temporary; the transaction program can try the conversation again. APPC/PC returns this return code on a subsequent verb.

APPC_ABENDED indicates that APPC/PC has been abnormally terminated.

APPC_BUSY indicates that APPC/PC is currently executing another verb and cannot execute this verb. This error can occur if a verb is issued after APPC/PC execution is interrupted (for example, by a Ctrl-Break or timer interrupt).

APPC_DISABLED indicates that APPC/PC is disabled as a result of the DISABLE/ENABLE_APPC verb.

CONV_FAILURE_NO_RETRY indicates that a failure occurred that caused APPC/PC to terminate the mapped conversation prematurely. For example, APPC/PC deactivated the session that the transaction programs were using for the mapped conversation because of a session protocol error, or APPC/PC deallocated the mapped conversation because of protocol error between

the mapped conversation components of the LUs. The condition is permanent; the transaction program should not try the conversation again until the condition is corrected. APPC/PC can report this return code to the local transaction program with a verb it issues in any state other than reset. APPC/PC places the mapped conversation in reset state.

CONV_FAILURE_RETRY indicates that a failure occurred that caused APPC/PC to terminate the mapped conversation prematurely. For example, APPC/PC terminates conversations when it must deactivate the associated sessions in response to a line or modem failure. The condition is temporary; the transaction program can try the conversation again. However, to avoid congesting a network with attempted allocation requests, the transaction program should pause or wait for a keystroke before retrying the transaction. APPC/PC can report this return code to the local transaction program on a verb that the transaction program issues in any state other than reset. APPC/PC places the mapped conversation in reset state.

CONVERSATION_TYPE_MIXED indicates that the local transaction program issued both basic and mapped conversation verbs for the same conversation. APPC/PC reports this return code with the verb issued. The state of the mapped conversation remains unchanged.

DEALLOCATE_ABEND indicates that the remote transaction program issued an MC_DEALLOCATE verb specifying the TYPE(ABEND) parameter. The remote LU can also issue an MC_DEALLOCATE verb specifying the TYPE(ABEND) parameter in response to a remote transaction program abnormal termination condition. If the mapped conversation for the remote transaction program was in receive state when the remote transaction program or LU issued an MC_DEALLOCATE, information sent by the local transaction program and not yet received by the remote transaction program is purged.

APPC/PC returns the DEALLOCATE_ABEND return code to the local program with a verb the program issues in either send or receive state. After APPC/PC issues this

return code, it places the mapped conversation in reset state.

DEALLOCATE_NORMAL indicates that the remote transaction program issued an MC_DEALLOCATE verb specifying the TYPE(SYNC_LEVEL) where the synchronization level of the conversation is NONE or TYPE(FLUSH) parameter. APPC/PC reports this return code to the local transaction program on a verb the program issues in receive state. APPC/PC leaves the mapped conversation in reset state.

INCOMPLETE indicates that the verb has not finished and must be re-issued unchanged before any other verb with the same TP_ID. Before re-issuing the verb, you should try to issue verbs on other transaction programs, including other unfinished verbs. If you are queueing incoming ALLOCATEs in the LUs, you should also periodically issue GET_ALLOCATE. This return code is returned only if ATTACH_PU (RETURN_CONTROL = INCOMPLETE) was issued. For more information on this return code, see "System Deadlocks" on page 10-2.

INCOMPLETE_ALTERED_VERB indicates that a verb was issued with the same TP_ID as that of an unfinished verb, or the unfinished verb was altered before it was re-issued.

Note: You may change the first 12 bytes of an incomplete verb so that you can place list pointers in this area to create a list of incomplete verbs.

INVALID_VERB indicates that APPC/PC did not recognize the verb operation code of the issued verb. APPC/PC reports this return code on the verb issued. The state of the mapped conversation remains unchanged.

OK indicates that APPC/PC executed the verb successfully. That is, APPC/PC performed the function defined for the verb, up to the point at which it returns control to the transaction program. The state of the mapped conversation is as defined for the verb.

PROG_ERROR_NO_TRUNC indicates that the remote program issued an MC_SEND_ERROR verb and the mapped conversation for the remote transaction program was in send state. No truncation of data occurs at the mapped conversation API.

APPC/PC reports this return code to the local transaction program on an MC_RECEIVE_AND_WAIT or MC_RECEIVE_IMMEDIATE verb that the transaction program issues before receiving data records or after receiving one or more complete data records. The mapped conversation remains in receive state.

PROG_ERROR_PURGING indicates that the remote transaction program issued an MC_SEND_ERROR verb, and the mapped conversation for the remote transaction program was in receive or confirm state. The MC_SEND_ERROR may cause data sent by the local transaction program to be purged.

Purging occurs when a transaction program issues MC_SEND_ERROR in receive state before receiving all the information sent by its partner transaction program. That is, APPC/PC purges the information sent before it reports the error.

The purging can occur at the local LU, the remote LU, or both. No purging occurs when a transaction program issues MC_SEND_ERROR in confirm state or receive state after receiving all the information sent by its partner program.

APPC/PC normally reports this return code to the local transaction program with a verb that the local transaction program issues after sending some information to the remote transaction program. However, APPC/PC can report the return code with a verb that the program issues before sending any information, depending on the verb and when the program issues it. APPC/PC leaves the mapped conversation in receive state.

Note:

The PARAMETER_CHECK and STATE_CHECK errors can also occur on any verb. When APPC/PC detects these error conditions, it does not try to execute the verb. A PARAMETER_CHECK error occurs when APPC/PC detects an invalid parameter value. A STATE_CHECK error occurs when the conversation is not in the correct state for the verb the transaction program is issuing. The verb descriptions list the secondary return codes that identify the causes of these errors.

The table below correlates the return codes to the verbs on which they can be returned.

Return Codes	Verbs													
	MC ALLOCATE	MC CONFIRM	MC CONFIRMED	MC DEALLOCATE	MC FLUSH	MC GET ATTRIBUTES	GET TYPE	MC PREPARE TO RECEIVE	MC RECEIVE AND WAIT	MC RECEIVE IMMEDIATE	MC REQUEST TO SEND	MC SEND DATA	MC SEND ERROR	MC TEST
ALLOCATION_ERROR	X	X		X				X	X	X		X	X	
APPC_ABENDED	X	X	X	X	X	X	X	X	X	X	X	X	X	X
APPC_BUSY	X	X	X	X	X	X	X	X	X	X	X	X	X	X
APPC_DISABLED	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CONV_FAILURE_NO_RETRY		X		X				X	X	X		X	X	
CONV_FAILURE_RETRY		X		X				X	X	X		X	X	
CONVERSATION_TYPE_MIXED		X	X	X	X	X		X	X	X	X	X	X	X
DEALLOCATE_ABEND		X		X				X	X	X		X	X	
DEALLOCATE_NORMAL									X	X			X	
INCOMPLETE	X	X		X				X	X	X		X	X	
INCOMPLETE_ALTERED_VERB	X	X		X				X	X	X		X	X	
OK	X	X	X	X	X	X	X	X	X	X	X	X	X	X
PARAMETER_CHECK	X	X	X	X	X	X	X	X	X	X	X	X	X	X
PROG_ERROR_NO_TRUNC									X	X				
PROG_ERROR_PURGING		X		X				X	X	X		X	X	
STATE_CHECK		X	X	X	X			X	X	X	X	X		
UNSUCCESSFUL	X									X				X

Each X in the table means that APPC/PC can return the return code with the corresponding verb.

The individual verb descriptions list the applicable return codes. The descriptions for most verbs do not explicitly list the secondary return codes of ALLOCATION_ERROR because they can occur at any time and are not necessarily a result of the verb on which they are returned. The description of the MC_ALLOCATE verb, however, lists the two secondary return codes that indicate an allocation failure.

Verb Descriptions

Detailed descriptions of the mapped conversation verbs follow.

MC_ALLOCATE

Allocates a session, if one is available, between the local LU and a remote LU, and establishes a mapped conversation between the local transaction program and a remote transaction program on that session. APPC/PC assigns a CONV_ID to the conversation. Except for the first conversation of a remotely initiated transaction program, the transaction program must issue this verb before it can issue any verbs that refer to the mapped conversation.

MC_ALLOCATE	Supplied Parameters: TP_ID (variable) PARTNER_LU_NAME (variable) MODE_NAME (variable) TPN (variable) RETURN_CONTROL (WHEN_SESSION_ALLOCATED) (IMMEDIATE) (WHEN_SESSION_FREE) SYNC_LEVEL (CONFIRM) (NONE) SECURITY (NONE) (SAME) (PGM (USER_ID (variable) PASSWORD (variable)) PIP_DATA_LENGTH (variable) PIP_DATA (variable)
	Returned Parameters: CONV_ID (variable) RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

PARTNER_LU_NAME specifies the name (in EBCDIC) of the remote LU at which the remote transaction program is located. The local LU uses the PARTNER_LU_NAME to identify the remote LU when allocating a session. That is, the PARTNER_LU_NAME is the remote LU network name.

MODE_NAME specifies the mode name (in EBCDIC) designating the network properties for the session APPC/PC is to allocate for the mapped conversation. For example, one network property is the class of service APPC/PC provides for the session used. Mapped conversations cannot use the SNASVCMG mode.

TPN specifies the name (in EBCDIC) of the remote transaction program that manages the other end of the mapped conversation. TPN cannot specify an SNA service transaction program name at the mapped conversation API. (SNA service transaction program names begin with a character value in the range X'00' to X'3F'.)

RETURN_CONTROL specifies when the local LU is to return control to the local transaction program, after the program requests the LU to allocate a session for the conversation. APPC/PC also reports allocation errors caused by the local LU's failure to obtain a session for the conversation on this verb. APPC/PC reports allocation errors caused by the remote LU rejecting the allocation request on a subsequent verb. The valid values for the RETURN_CONTROL parameter are:

- **WHEN_SESSION_ALLOCATED** directs the LU to allocate a session for the conversation before returning control to the transaction program. Except for instances of conversation failure, the LU does not return control to the program until a session becomes available.

  This option may cause a deadlock situation if there are not enough sessions for the currently active conversations of all transaction programs. Specifying the RETURN_CONTROL(INCOMPLETE) option on the ATTACH_PU verb can confine the problem to the currently executing transaction program, but if this program has itself used up all sessions, the program can still cause a deadlock situation by using the RETURN_CONTROL(WHEN_SESSION_ALLOCATED) option. You must analyze your operating environment to determine if this option is safe. For more

information on this problem, see "System Deadlocks" on page 10-2.

- **IMMEDIATE** directs the LU to allocate a session for the conversation if a contention-winner session is immediately available. This option returns control to the transaction program immediately with a return code indicating whether a session is allocated. In this case, the local LU does not have to wait for a response from the partner LU, and the MC_ALLOCATE verb does not require information to be sent to the partner LU.

  - A return code of OK indicates that a session is immediately available and is allocated for the conversation. A session is immediately available when it is active, it is not allocated to another conversation, and the local LU is the contention winner for the session.

  - A return code of UNSUCCESSFUL indicates that a session is not immediately available and that the LU did not perform the allocation.

- **WHEN_SESSION_FREE** directs the LU to allocate a session for the conversation only if a session (either a contention winner or loser) is available or able to be activated, and then to return control to the transaction program with a return code indicating whether a session is allocated.

  - A return code of OK indicates that a session is available and is allocated for the conversation. If activation of a session is necessary, the LU performs the session activation before returning control to the transaction program.

  - A return code of ALLOCATION_ERROR indicates that a session is unavailable and that the LU cannot activate a new session. No session is allocated.

SYNC_LEVEL specifies the synchronization level that the local and remote transaction programs can use for this conversation.

- **NONE** specifies that the transaction programs do not perform confirmation processing on this conversation. The programs do not issue verbs and do not recognize returned parameters relating to the synchronization function.

- **CONFIRM** specifies that the transaction programs can perform confirmation processing on this conversation. The programs can issue verbs and recognize returned parameters relating to confirmation.

SECURITY specifies access security information that the remote LU uses to validate access to the remote transaction program and its conversations. The access security information includes a user ID and a password provided using the following arguments:

- **NONE** specifies that this conversation does not use conversation-level security.

- **SAME** informs the remote LU that the user ID has already been verified as part of the allocation request that initiated execution of the local transaction program. APPC/PC saved the user ID when the local transaction program was remotely initiated. This option also causes APPC/PC to send this saved user ID to the remote LU with an 'already verified' indication. If there was no user ID, APPC/PC sends the allocation request with no security information and a 'not already verified' indication.

- **PGM** directs the remote LU to use the security information that the local transaction program provides on this parameter. The local transaction program uses the following arguments to provide this security information:

  - **USER_ID** specifies the ID (in EBCDIC) of the end user. The USER_ID is 8 bytes long. The remote LU uses this value and the password to verify the

identity of the end user making the allocation request. In addition, the remote LU may use the user ID for auditing or accounting to associate conversation accesses with the end user.

- **PASSWORD** specifies the password (in EBCDIC) of the end user. The PASSWORD can be up to 8 bytes long. The remote LU uses this value and the user ID to verify the identity of the end user making the allocation request.

PIP_DATA_LENGTH specifies the length of the program initialization parameters for the remote program. Set this parameter to 0 if there is no PIP data.

PIP_DATA specifies the variable containing the PIP data that the local program is sending to the remote program. The transaction program *must* format the PIP data according to the GDS format (see "FM Headers" in *SNA Reference Summary*).

Returned Parameters:

CONV_ID indicates the ID of the new conversation.

RETURN_CODE indicates the result of verb execution. The MC_ALLOCATE return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - DATA_AREA_ACROSS_SEGMENT: APPC/PC does not permit PIP data to cross a segment boundary.
  - BAD_TPN_LEN: The value that TPN specifies is too short (less than 1) or too long (greater than 64).
  - BAD_SYNC_LEVEL: APPC/PC does not recognize the specified SYNC_LEVEL.

- BAD_SECURITY_SELEC: APPC/PC does not recognize the specified SECURITY.

- BAD_RETURN_CONTROL: APPC/PC does not recognize the specified RETURN_ CONTROL.

- TOO_BIG_SEC_TOKENS: APPC/PC does not accept a password or user ID that is longer than 8 bytes.

- PIP_LEN_INCORRECT: APPC/PC does not accept PIP data that is longer than 32767 bytes.

- NO_USE_OF_SNASVCMG: APPC/PC does not accept SNASVCMG as the value for the MODE_NAME parameter.

- UNKNOWN_PARTNER_MODE: APPC/PC does not recognize the specified PARTNER_LU_NAME or MODE_NAME.

- ALLOCATION_ERROR

  - ALLOCATION_FAILURE_NO_RETRY: APPC/PC cannot allocate the conversation because of a permanent error condition.

  - ALLOCATION_FAILURE_RETRY: APPC/PC cannot allocate the conversation because of a temporary error condition.

- UNSUCCESSFUL: The program specified RETURN_CONTROL(IMMEDIATE) and APPC/PC could not allocate the conversation because no contention-winner sessions were available.

For detailed information on the following return codes, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- INCOMPLETE

- INCOMPLETE_ALTERED_VERB
- OK.

State Changes:

The conversation enters send state if the RETURN_CODE indicates OK.

Notes:

1. The local LU does not send PIP data immediately unless the local transaction program issues a verb (other than MC_SEND_DATA) that explicitly directs the LU to flush its send buffer. Otherwise, the LU retains the PIP parameter of the MC_ALLOCATE verb and accumulates data from subsequent MC_SEND_DATA verbs.

   The LU sends this data to the partner LU when it accumulates enough data for transmission. The amount of information that is sufficient for transmission depends on the characteristics of the session allocated for the conversation, and can vary from session to session.

2. The local transaction program can ensure that APPC/PC connects the remote transaction program as soon as possible by issuing the MC_FLUSH verb after the MC_ALLOCATE verb.

3. Contention for a session can occur when two LUs connected by a session both try to allocate a conversation on the session at the same time. APPC/PC resolves contention by making one LU the contention winner of the session and the other LU the contention loser of the session.

   The contention-winner LU allocates a conversation on a session without asking permission from the contention-loser LU. Conversely, the contention-loser LU requests permission from the contention-winner LU to allocate a conversation on the session. Then the contention-winner LU either grants or rejects the request.

4. The remote transaction program starts the conversation in receive state.

5. For an IBM Token-Ring Network, one link must be reserved for outgoing calls to enable the MC_ALLOCATE verb to complete. For more information on reserving links, see "Entering Information for an IBM Token-Ring DLC" in the *APPC/PC Installation and Configuration Guide*

6. You must specify the PARTNER_LU_NAME, MODE_NAME, TPN, USER_ID, and PASSWORD in EBCDIC. You can use the CONVERT verb to convert these parameters from ASCII to EBCDIC.

MC_CONFIRM

Sends a confirmation request to a remote transaction program and waits for a reply. This verb enables the local and remote transaction programs to synchronize their processing with one another. MC_CONFIRM also causes the LU to flush its send buffer. To use this verb, the transaction program must allocate the conversation with a synchronization level of CONFIRM.

MC_CONFIRM	Supplied Parameters: TP_ID (variable) CONV_ID (variable)
	Returned Parameters: RETURN_CODE (variable) REQUEST_TO_SEND_RECEIVED (YES) (NO)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which the local transaction program is requesting confirmation.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The MC_CONFIRM return codes are:

- OK: The remote program replied CONFIRMED.
- PARAMETER_CHECK

- BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.

- BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

- CONFIRM_ON_SYNC_NONE: APPC/PC does not permit the program to use this verb if it allocated the conversation with SYNC_LEVEL(NONE).

- STATE_CHECK

  - CONFIRM_BAD_STATE: The conversation is not in send state.

  - CONFIRM_NOT_LL_BDY: The conversation is in send state, and the program started, but did not finish, sending a logical record.

For detailed information on the following return codes, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes."

- ALLOCATION_ERROR
- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DEALLOCATE_ABEND
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- PROG_ERROR_PURGING.

REQUEST_TO_SEND_RECEIVED indicates whether the local LU received an MC_REQUEST_TO_SEND verb. The two indications APPC/PC can return are YES and NO.

- YES indicates that the local LU received an MC_REQUEST_TO_SEND verb from the remote transaction program. The remote transaction program

issues an MC_REQUEST_TO_SEND verb to request the local transaction program to enter receive state and place the remote transaction program in send state.

- NO indicates that the local LU has not received an MC_REQUEST_TO_SEND notification from the remote transaction program

State Changes:

If the RETURN_CODE parameter indicates OK, the state of the conversation remains the same.

For more information on state changes when the RETURN_CODE indicates other than OK, see "Understanding Mapped Conversation Return Codes" on page 6-3.

Notes:

1. The transaction program can use this verb for various application-level functions. For example:

   - The transaction program can issue this verb immediately after an MC_ALLOCATE to determine whether the allocation of the conversation is successful before sending data

   - The transaction program can issue this verb to request acknowledgment of data it sent to the remote program. The remote program can respond by issuing MC_CONFIRMED to indicate that it received and processed the data without errors. Alternatively, the remote program can issue an MC_SEND_ERROR to indicate that it found an error.

2. When the REQUEST_TO_SEND_RECEIVED parameter indicates YES, the remote transaction program is requesting that the local transaction program enter receive state and place the remote program in send state. A transaction program enters receive state by issuing the

MC_PREPARE_TO_RECEIVE verb or the MC_RECEIVE_AND_WAIT verb. The transaction partner program enters send state after it issues the MC_RECEIVE_AND_WAIT verb or the MC_RECEIVE_IMMEDIATE verb, and receives the SEND indication from its partner transaction program on the WHAT_RECEIVED parameter.

MC_CONFIRMED

Sends a confirmation reply to the remote transaction program. This verb enables the local and remote transaction programs to synchronize their processing. The local transaction program can issue this verb when it receives a confirmation request. (For more information on confirmation requests, see the WHAT_RECEIVED parameter of the MC_RECEIVE_AND_WAIT verb.)

MC_CONFIRMED	Supplied Parameters: TP_ID (variable) CONV_ID (variable)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which the local transaction program is returning confirmation.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The MC_CONFIRMED return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

- STATE_CHECK
  - CONFIRMED_BAD_STATE: The conversation is not in confirm state.

For detailed information on the following return codes, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- OK.

State Changes:

The conversation enters **receive** state if the program received CONFIRM on the preceding MC_RECEIVE_AND_WAIT or MC_RECEIVE_IMMEDIATE.

The conversation enters **send** state if the program received CONFIRM_SEND on the preceding MC_RECEIVE_AND_WAIT or MC_RECEIVE_IMMEDIATE.

The conversation enters **reset** state if the program received CONFIRM_DEALLOCATE on the preceding MC_RECEIVE_AND_WAIT or MC_RECEIVE_IMMEDIATE.

Note:

The program can issue this verb only as a reply to a confirmation request.

MC_DEALLOCATE

Deallocates the specified conversation. MC_DEALLOCATE can perform the function of the MC_CONFIRM verb before it deallocates the conversation. APPC/PC discards the conversation ID after deallocating the conversation.

MC_DEALLOCATE	Supplied Parameters: TP_ID (variable) CONV_ID (variable) TYPE (SYNC_LEVEL) (FLUSH) (ABEND)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation that the local transaction program is deallocating.

TYPE specifies the type of deallocation APPC/PC is to perform.

- **SYNC_LEVEL** directs APPC/PC to deallocate the conversation based on the synchronization level allocated to this conversation.
  - If the SYNC_LEVEL is NONE, APPC/PC performs the function of the MC_FLUSH verb and then deallocates the conversation normally.

- If the SYNC_LEVEL is CONFIRM, APPC/PC performs the function of the MC_CONFIRM verb and, if it is successful (as indicated by a return code of OK on this MC_ALLOCATE verb), APPC/PC deallocates the conversation normally. If it is unsuccessful, the return code determines the state of the conversation.

- **FLUSH** directs APPC/PC to perform the function of the FLUSH verb and then to deallocate the conversation normally. When you specify FLUSH as the type, the remote partner receives a DEALLOCATE_NORMAL following the successful deallocation of a conversation.

- **ABEND** directs APPC/PC to perform the function of the FLUSH verb when the conversation is in send state, and then to deallocate the conversation abnormally by purging data, if necessary. APPC/PC purges data when a program deallocates a conversation in receive state.

  The remote program is informed of the deallocation by a DEALLOCATE_ABEND indication unless the remote program has issued an MC_SEND_ERROR. If the remote program has issued an MC_SEND_ERROR, it may receive a DEALLOCATE_NORMAL return code instead of DEALLOCATE_ABEND.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The MC_DEALLOCATE return codes are:

- PARAMETER_CHECK

  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.

  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

- DEALLOCATE_BAD_TYPE: APPC/PC does not recognize the specified TYPE.

- STATE_CHECK

  - DEALLOC_FLUSH_BAD_STATE: The program specified the TYPE(SYNC_LEVEL) parameter for a conversation specified with SYNC_LEVEL(NONE) and the conversation is not in send state. Alternatively, the program may have specified TYPE(FLUSH) when the conversation was not in send state.

  - DEALLOC_CONFIRM_BAD_STATE: The program specified the TYPE(SYNC_LEVEL) parameter for a conversation specified with SYNC_LEVEL(CONFIRM) when the conversation was not in send state.

For detailed information on the following return codes, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- OK.

APPC/PC can report the following return codes if the program specifies TYPE(SYNC_LEVEL) and the synchronization level allocated to this conversation is CONFIRM. For detailed information on these return codes, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes."

- ALLOCATION_ERROR
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DEALLOCATE_ABEND
- PROG_ERROR_PURGING.

State Changes:

The conversation enters **reset** state when RETURN_CODE indicates OK.

For information on state changes when RETURN_CODE indicates other than OK, see "Understanding Mapped Conversation Return Codes" on page 6-3.

Notes:

1. A transaction program can use the TYPE(SYNC_LEVEL) parameter to deallocate the conversation according to the synchronization level allocated to the conversation.

   - If the synchronization level is NONE, APPC/PC deallocates the conversation unconditionally.

   - If the synchronization level is CONFIRM, APPC/PC deallocates the conversation after the remote program responds to the confirmation request by issuing MC_CONFIRMED. APPC/PC does not deallocate the conversation if the remote program responds to the confirmation request by issuing MC_SEND_ERROR.

2. A transaction program can use the TYPE(FLUSH) parameter to deallocate the conversation unconditionally (that is, regardless of its synchronization level).

3. A transaction program can use the TYPE(ABEND) parameter to deallocate the conversation unconditionally regardless of its synchronization level and its current state. Specifically, a transaction program uses this parameter when it detects an error that prevents useful communication (that is, communication leading to successful completion of the transaction). The transaction programs define the specific use and meaning of ABEND.

MC_FLUSH

Flushes the send buffer of the local LU by sending all buffered information to the remote LU. APPC/PC sends all of the information stored in the local LU to the remote LU. Information buffered by the LU can come from MC_ALLOCATE, MC_SEND_DATA, MC_PREPARE_TO_RECEIVE, and MC_SEND_ERROR. Refer to the descriptions of these verbs for more details on this buffered information, including when APPC/PC places information in the buffer.

MC_FLUSH	Supplied Parameters: TP_ID (variable) CONV_ID (variable)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which APPC/PC is to flush the local LU's send buffer.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The MC_FLUSH return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.

- BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

- STATE_CHECK

  - FLUSH_NOT_SEND_STATE: The conversation is not in send state. The MC_FLUSH verb can only be issued in send state.

For detailed information on the following return codes, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- OK.

State Changes:

None

Notes:

1. This verb is useful for reducing the time a remote transaction program must wait for data. The LU normally buffers the data from consecutive MC_SEND_DATA verbs until it has enough to transmit. Then it transmits the buffered data to the remote LU. However, the local transaction program can issue MC_FLUSH to force the local LU to transmit the buffered data, eliminating the wait for the buffer to fill. In this way, the local transaction program can minimize the time that the remote transaction program has to wait to receive the data it is to process.

2. The LU flushes its send buffer only when it has some information to transmit. If the LU has no information in its send buffer, it does not transmit anything to the remote LU.

3. The local transaction program can ensure that APPC/PC connects the remote transaction program as soon as possible by issuing the MC_FLUSH verb immediately after the MC_ALLOCATE verb.

MC_GET_ATTRIBUTES

Returns information pertaining to the specified mapped conversation.

MC_GET_ATTRIBUTES	Supplied Parameters: TP_ID (variable) CONV_ID (variable)
	Returned Parameters: RETURN_CODE (variable) LU_ID (variable) OWN_NET_NAME (variable) OWN_LU_NAME (variable) PARTNER_LU_NAME (variable) PARTNER_FULLY_QUALIFIED_LU_NAME (variable) MODE_NAME (variable) SYNC_LEVEL (variable) USER_ID (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation for which the attributes are desired.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The MC_GET_ATTRIBUTES return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

For detailed information on the following return codes, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- OK.

LU_ID indicates the identifier for the local LU under which the transaction program is executing. APPC/PC returns this value to the application subsystem of the transaction program when it defines the LU after the application subsystem issues ATTACH_LU. For more information, see "ATTACH_LU" on page 5-7.

OWN_NET_NAME indicates the name (in EBCDIC) of the network containing the LU at which the local transaction program is located.

OWN_LU_NAME indicates the network name (in EBCDIC) of the LU at which the local transaction program is located.

PARTNER_LU_NAME indicates the name (in EBCDIC) of the LU at which the remote transaction program is located. The local LU uses this name to identify the remote LU when allocating a conversation. For more details, see the description of the LU_NAME parameter

of MC_ALLOCATE under "MC_ALLOCATE" on page 6-13.

PARTNER_FULLY_QUALIFIED_LU_NAME indicates the fully qualified name (in EBCDIC) of the LU at which the remote transaction program is located. If APPC/PC does not know the partner's fully-qualified LU name, it returns a null (0-length) value.

MODE_NAME indicates the mode name (in EBCDIC) for the session on which APPC/PC allocated the conversation.

SYNC_LEVEL indicates the level of synchronization processing that the programs are using for the conversation. The synchronization levels are NONE and CONFIRM.

USER_ID indicates the user ID (in EBCDIC) if the program specified conversation-level security. An incoming ALLOCATE specifies this USER_ID when it requests the application subsystem to start a transaction program. If the program did not specify conversation-level security, or if the transaction program was initiated locally, APPC/PC returns a null (0-length) value for the USER_ID parameter.

State Changes:

None

Notes:

1. The program issues this verb to obtain the attributes of a conversation.

2. The following parameters are returned in EBCDIC:

   - OWN_NET_NAME
   - OWN_LU_NAME
   - PARTNER_LU_NAME
   - PARTNER_LU_FULLY_QUALIFIED_NAME
   - MODE_NAME
   - USER_ID in EBCDIC

You can use the CONVERT verb to convert these parameter values from ASCII to EBCDIC.

GET_TYPE

Returns the conversation type (basic or mapped) of the specified conversation. A transaction program can use this verb for both basic conversations or mapped conversations.

GET_TYPE	Supplied Parameters: TP_ID (variable) CONV_ID (variable)
	Returned Parameters: RETURN_CODE (variable) TYPE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation for which the conversation type is desired.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The GET_TYPE return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

For detailed information on the following return codes, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- OK.

TYPE indicates the conversation type. The conversation types are:

- BASIC_CONVERSATION indicates that a basic conversation was initiated by one of the following methods:

  - The conversation was initiated with an ALLOCATE (CONV_TYPE = BASIC_CONVERSATION) verb.

  - An incoming ALLOCATE specified a basic conversation.

- MAPPED_CONVERSATION indicates that a mapped conversation was initiated by one of the following methods:

  - The conversation was initiated with an MC_ALLOCATE verb.

  - The conversation was initiated with an ALLOCATE (CONV_TYPE = MAPPED_CONVERSATION) verb.

  - An incoming ALLOCATE specified a mapped conversation.

State Changes:

None

Note:

A program that APPC/PC can process at either the basic conversation API or the mapped conversation API uses this verb to determine which category of verbs, basic conversation or mapped conversation, to issue.

MC_PREPARE_TO_RECEIVE

Changes a mapped conversation from send to receive state in preparation to receive data. Also performs the function of the MC_FLUSH or MC_CONFIRM verbs.

MC_PREPARE_TO_RECEIVE	Supplied Parameters: TP_ID (variable) CONV_ID (variable) TYPE (SYNC_LEVEL) (FLUSH) LOCKS (SHORT) (LONG)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which the local transaction program is preparing to receive data.

TYPE specifies how APPC/PC prepares the conversation to receive data.

- **SYNC_LEVEL** directs APPC/PC to prepare the conversation to receive data based on the synchronization level allocated to this conversation:
  - If SYNC_LEVEL is NONE, execute the function of the MC_FLUSH verb and then place the conversation in receive state.

- If SYNC_LEVEL is CONFIRM, execute the function of the MC_CONFIRM verb (including flushing the local LU's send buffer) and then place the conversation in receive state if the MC_CONFIRM function is successful (as indicated by a return code of OK with this MC_PREPARE_TO_RECEIVE verb). If the execution of the MC_CONFIRM function is unsuccessful, the return code determines the state of the conversation.

- **FLUSH** directs APPC/PC to flush the local LU's send buffer before placing the conversation in receive state.

LOCKS specifies when APPC/PC is to return control to the local program after executing the MC_CONFIRM function of this verb. APPC/PC ignores this parameter unless the verb also specifies TYPE(SYNC_LEVEL) and the synchronization level for this conversation is CONFIRM.

- **SHORT** directs APPC/PC to return control to the local transaction program after it receives an MC_CONFIRMED reply. With the short option, control is returned to the transaction program more quickly; however, more data flows on the line.

- **LONG** informs APPC/PC that the MC_CONFIRMED reply is to be returned with any returned data that is available. With the LONG option, there is less line flow but APPC/PC takes longer to return control to the local transaction program.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. For detailed information on return codes listed below that do not include descriptions, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes." The MC_PREPARE_TO_RECEIVE return codes are:

- PARAMETER_CHECK

- BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.

- BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

- P_TO_R_INVALID_TYPE: APPC/PC does not recognize the specified TYPE.

- STATE_CHECK

  - P_TO_R_NOT_SEND_STATE: The conversation is not in send state.

For detailed information on the following return codes, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- OK.

APPC/PC can report the following return codes if the program specifies TYPE(SYNC_LEVEL) and the synchronization level allocated to this conversation is CONFIRM.

- ALLOCATION_ERROR
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DEALLOCATE_ABEND
- PROG_ERROR_PURGING.

State Changes:

The conversation enters receive state when RETURN_CODE indicates OK.

For information on state changes when RETURN_CODE indicates other than OK, see "Understanding Mapped Conversation Return Codes" on page 6-3.

Notes:

1. The conversation for the remote transaction program enters send state when the remote transaction program issues an MC_RECEIVE_AND_WAIT or MC_RECEIVE_IMMEDIATE verb and receives the SEND indication on the WHAT_RECEIVED parameter. The remote transaction program can then send data to the local transaction program.

2. If the local transaction program issues MC_PREPARE_TO_RECEIVE with a SYNC_LEVEL of CONFIRM, the remote transaction program enters send state after issuing MC_CONFIRMED.

MC_RECEIVE_AND_WAIT

Waits for information to arrive on the specified conversation and then receives the information. If information is already available, the transaction program receives it without waiting. The information can be data, conversation status, or a request for confirmation. APPC/PC returns control to the transaction program and indicates the type of information received.

The transaction program can issue this verb when the conversation is in either receive or send state. If the conversation is in send state, the LU first flushes its send buffer, sending all buffered information and the SEND indication to the remote transaction program. This action places the local conversation in receive state. The LU then waits for information to arrive from the remote transaction program. The remote transaction program sends data to the local transaction program after it receives the SEND indication.

MC_RECEIVE_AND_WAIT	Supplied Parameters: TP_ID (variable) CONV_ID (variable) DATA_PTR (variable) MAX_LENGTH (variable)
	Returned Parameters: RETURN_CODE (variable) DATA_LENGTH (variable) DATA (see DATA_PTR) WHAT_RECEIVED (variable) REQUEST_TO_SEND_RECEIVED (YES) (NO)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the conversation ID of the conversation on which the local transaction program is to receive data.

DATA_PTR specifies the address of the buffer that is to accept the received data.

MAX_LENGTH specifies the maximum amount of data (in bytes) that the program is to receive. Values between 0 and 65535 are valid for this parameter, but the sum of this value and the offset portion of DATA_PTR must not exceed 65535. This limit keeps the incoming data from crossing a segment boundary.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The MC_RECEIVE_AND_WAIT return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.
  - DATA_AREA_ACROSS_SEGMENT: The data receiving area crosses a segment boundary.
- STATE_CHECK
  - RCV_AND_WAIT_BAD_STATE: The conversation is not in send or receive state.

For detailed information on the following return codes, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes."

- ALLOCATION_ERROR
- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DEALLOCATE_NORMAL
- DEALLOCATE_ABEND
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- OK
- PROG_ERROR_NO_TRUNC
- PROG_ERROR_PURGING.

DATA_LENGTH indicates the actual amount of data the transaction program received up to the maximum (MAX_LENGTH). The value of this variable is 0 if the program receives information other than data, or no information at all.

DATA from the partner transaction program is received in the buffer specified by the address in the DATA_PTR parameter. APPC/PC does not place any information in this buffer when the program receives information other than data, as indicated by the WHAT_RECEIVED parameter.

WHAT_RECEIVED indicates what the transaction program received. The transaction program should examine this variable only when RETURN_CODE indicates OK. APPC/PC does not place any information in this variable when RETURN_CODE indicates other than OK.

- DATA_COMPLETE indicates that the transaction program has received a complete data record, or the last remaining portion of a data record.

- DATA_INCOMPLETE indicates that the transaction program has received an incomplete data record. The transaction program can issue another MC_RECEIVE_AND_WAIT or MC_RECEIVE_IMMEDIATE (or more than one) to receive the remaining data.

- SEND indicates that the remote transaction program has entered receive state, thereby placing the local transaction program in send state. After receiving the SEND indication, the local transaction program can issue the MC_SEND_DATA verb to send data to the remote program.

- CONFIRM indicates that the remote transaction program has issued MC_CONFIRM, requesting the local transaction program to respond by issuing MC_CONFIRMED. The program may respond, instead, by issuing MC_SEND_ERROR.

- CONFIRM_SEND indicates that the remote transaction program has issued MC_PREPARE_TO_RECEIVE with TYPE(SYNC_LEVEL), and the synchronization level is CONFIRM. The local transaction program can respond by issuing MC_CONFIRMED or MC_SEND_ERROR.

- CONFIRM_DEALLOCATE indicates that the remote transaction program has issued MC_DEALLOCATE with TYPE(SYNC_LEVEL), and the synchronization level is CONFIRM. The local transaction program can respond by issuing MC_CONFIRMED or MC_SEND_ERROR.

REQUEST_TO_SEND_RECEIVED indicates whether the local LU has received a REQUEST_TO_SEND. The indication is either YES or NO.

- YES indicates that the local LU has received a REQUEST_TO_SEND notification from the remote transaction program. The remote transaction program has issued MC_REQUEST_TO_SEND, requesting the local transaction program to enter receive state and

thereby place the remote transaction program in send state.

- NO indicates that the local LU has not received a REQUEST_TO_SEND notification from the remote transaction program.

State Changes:

The following state changes occur when the RETURN_CODE parameter indicates OK. For information on state changes when RETURN_CODE indicates other than OK, see "Understanding Mapped Conversation Return Codes" on page 6-3.

- The conversation enters receive state when the verb is issued in send state and WHAT_RECEIVED indicates DATA_COMPLETE or DATA_INCOMPLETE.

- The conversation enters send state when WHAT_RECEIVED indicates SEND.

- The conversation enters confirm state when WHAT_RECEIVED indicates CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE.

- No state change occurs when the transaction program issues MC_RECEIVE_AND_WAIT in receive state and WHAT_RECEIVED indicates DATA_COMPLETE or DATA INCOMPLETE.

Notes:

1. When the transaction program issues MC_RECEIVE_AND_WAIT in send state, the LU implicitly executes an MC_PREPARE_TO_RECEIVE with TYPE(FLUSH) before executing the MC_RECEIVE_AND_WAIT. See the description of MC_PREPARE_TO_RECEIVE for details of this function.

2. The mapped conversation protocol boundary provides for sending and receiving only data records. Unlike the logical records defined for the basic conversation

protocol boundary, data records contain only data; they do not contain the logical record length field.

3. The MC_RECEIVE_AND_WAIT verb can receive only as much of the data record as the program specifies using the MAX_LENGTH parameter. The WHAT_RECEIVED parameter indicates whether the transaction program has received a complete or incomplete data record, as follows:

   - The WHAT_RECEIVED parameter indicates DATA_COMPLETE after the transaction program receives a complete data record or the last portion of a data record. The length of the record (or portion of the record) is equal to (or less than) the length specified on the MAX_LENGTH parameter.

   - The WHAT_RECEIVED parameter indicates DATA_INCOMPLETE after the transaction program receives a portion of a data record other than the last remaining portion. APPC/PC detects that the data record is incomplete because the remaining length of the record is greater than the length specified on the MAX_LENGTH parameter. The amount received equals the length specified. The transaction program must issue another MC_RECEIVE_AND_WAIT (or possibly more than one) to receive the rest of the data record.

4. A program can use MC_RECEIVE_AND_WAIT with MAX_LENGTH(0) to determine the type of information that is available without actually receiving data. The RETURN_CODE and WHAT_RECEIVED parameters provide this information as usual. However, the transaction program receives no data in this case.

5. The transaction program receives only one kind of information at a time. For example, it may receive data or a CONFIRM request, but it cannot receive both at the same time. The RETURN_CODE and WHAT_RECEIVED parameters indicate to the program the kind of information the program receives.

6. The local transaction program usually receives a REQUEST_TO_SEND notification when it is in send state. APPC/PC reports this request to the local transaction program on an MC_SEND_DATA verb or an MC_SEND_ERROR verb issued in send state. However, the local transaction program can receive notification when the program is in receive state in the following cases:

   - When the local transaction program just entered receive state and the remote transaction program issues MC_REQUEST_TO_SEND before it enters send state.

   - When the remote transaction program has just entered receive state by using the MC_PREPARE_TO_RECEIVE verb (not MC_RECEIVE_AND_WAIT), and then issuing MC_REQUEST_TO_SEND before the local transaction program responds by entering send state. This ambiguity can occur because MC_REQUEST_TO_SEND is an expedited request and this request can, therefore, arrive ahead of the request carrying the SEND indication.

     The local transaction program might not be able to distinguish this case from the preceding case. The remote transaction program can avoid this ambiguity by waiting until it receives information from the local transaction program before it issues the MC_REQUEST_TO_SEND.

   - The remote program issues the MC_REQUEST_TO_SEND in send state.

MC_RECEIVE_IMMEDIATE

Receives information that is available from the specified mapped conversation, but does not wait for information to arrive. The information (if any) can be data, conversation status, or a request for confirmation. APPC/PC returns control to the transaction program with an indication of whether information was received and, if so, the type of information.

MC_RECEIVE_IMMEDIATE	Supplied Parameters: TP_ID (variable) CONV_ID (variable) DATA_PTR (variable) MAX_LENGTH (variable)
	Returned Parameters: RETURN_CODE (variable) DATA_LENGTH (variable) DATA (see DATA_PTR) WHAT_RECEIVED (variable) REQUEST_TO_SEND_RECEIVED (YES) (NO)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which the local transaction program is to receive data.

DATA_PTR specifies the address of the data buffer that is to accept the received data.

MAX_LENGTH specifies the most data (in bytes) the program is to receive when APPC/PC executes this verb. Values between 0 and 65535 are valid for this parameter but the sum of this value and the offset portion of DATA_PTR must not exceed 65535. This limit keeps the incoming data from crossing a segment boundary.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The MC_RECEIVE_IMMEDIATE return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.
  - DATA_AREA_ACROSS_SEGMENT: The data receiving area crosses a segment boundary.
- STATE_CHECK
  - RCV_IMMD_NOT_RCV_STATE: The conversation is not in receive state.
  - UNSUCCESSFUL: No data was available to be received.

For detailed information on the following return codes, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes."

- ALLOCATION_ERROR
- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- CONV_FAILURE_NO_RETRY

- CONV_FAILURE_RETRY
- DEALLOCATE_NORMAL
- DEALLOCATE_ABEND
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- OK
- PROG_ERROR_NO_TRUNC
- PROG_ERROR_PURGING.

DATA_LENGTH indicates the actual amount of data the transaction program received up to the maximum (MAX_LENGTH). If the program receives information other than data, or no information at all, this variable is 0.

DATA from the partner transaction program is received in the buffer that the DATA_PTR parameter specifies. When the program receives information other than data, as indicated by the WHAT_RECEIVED parameter, APPC/PC does not place any information in this buffer.

WHAT_RECEIVED indicates what kind of information the transaction program received. The program should examine this variable only if the RETURN_CODE parameter indicates OK. APPC/PC does not place any information in this variable if the RETURN_CODE is other than OK. The WHAT_RECEIVED indications are:

- DATA_COMPLETE indicates that the transaction program has received a complete data record, or the last portion of a data record.

- DATA_INCOMPLETE indicates that the transaction program has received less than a complete data record. The transaction program can issue another MC_RECEIVE_IMMEDIATE or MC_RECEIVE_AND_WAIT (or possibly more than one) to receive the remaining data.

- CONFIRM indicates that the remote transaction program has issued MC_CONFIRM, requesting the local transaction program to respond by issuing MC_CONFIRMED. The program can respond by issuing MC_CONFIRMED or MC_SEND_ERROR.

- CONFIRM_SEND indicates that the remote transaction program has issued MC_PREPARE_TO_RECEIVE with TYPE(SYNC_LEVEL), and the synchronization level is CONFIRM. The local transaction program can respond by issuing MC_CONFIRMED or MC_SEND_ERROR.

- CONFIRM_DEALLOCATE indicates that the remote transaction program has issued MC_DEALLOCATE with TYPE(SYNC_LEVEL), and the synchronization level is CONFIRM. The local transaction program can respond by issuing MC_CONFIRMED or MC_SEND_ERROR.

- SEND indicates that the remote transaction program has entered receive state, placing the local transaction program in send state. After receiving the SEND indication, the local transaction program can issue the MC_SEND_DATA verb to send data to the remote transaction program.

REQUEST_TO_SEND_RECEIVED indicates whether the local LU has received a REQUEST_TO_SEND from the remote transaction program. The indication is either YES or NO.

- YES indicates that the local LU has received a REQUEST_TO_SEND notification from the remote transaction program. The remote transaction program has issued MC_REQUEST_TO_SEND, requesting the local transaction program to enter receive state and thereby place the remote transaction program in send state.

- NO indicates that the local LU has not received a REQUEST_TO_SEND notification.

State Changes:

The following state changes occur when the RETURN_CODE parameter indicates OK. For information on state changes when RETURN_CODE

indicates other than OK, see "Understanding Mapped Conversation Return Codes" on page 6-3.

- The conversation enters send state when WHAT_RECEIVED indicates SEND.

- The conversation enters confirm state when WHAT_RECEIVED indicates CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE.

- No state change occurs when the transaction program issues MC_RECEIVE_IMMEDIATE and WHAT_RECEIVED indicates DATA_COMPLETE or DATA_INCOMPLETE.

Notes:

1. The mapped conversation API provides for sending and receiving data records only. Unlike the logical records defined for the basic conversation API, data records contain only data; they do not contain the logical record length field.

2. The MC_RECEIVE_IMMEDIATE verb can receive only as much of the data record as the transaction program specifies using the MAX_LENGTH parameter. The WHAT_RECEIVED parameter indicates whether the program has received a complete or incomplete data record, as follows:

   - The WHAT_RECEIVED parameter indicates DATA_COMPLETE after the transaction program receives a complete data record or the last remaining portion of a data record. The length of the record (or portion of the record) is equal to (or less than) the length specified on the MAX_LENGTH parameter.

   - The WHAT_RECEIVED parameter indicates DATA_INCOMPLETE after the transaction program receives part of a data record other than the last portion. The program issues another MC_RECEIVE_IMMEDIATE (or possibly more than one) to receive the rest of the data record.

3. A program can use MC_RECEIVE_IMMEDIATE with MAX_LENGTH(0) to determine the type of information that is available without actually receiving data. The RETURN_CODE and WHAT_RECEIVED parameters provide this information, as usual. However, the program receives no data in this case.

4. The transaction program receives only one kind of information at a time. For example, it may receive data or a CONFIRM request, but it cannot receive both at the same time. The RETURN_CODE and WHAT_RECEIVED parameters indicate to the program the kind of information the program receives.

5. The local transaction program usually receives a REQUEST_TO_SEND notification when it is in send state. APPC/PC reports this request to the local transaction program in the RETURN_CODE of an MC_SEND_DATA verb or an MC_SEND_ERROR verb that the local transaction program issues in send state. However, the local transaction program can receive the notification when the program is in receive state in the following cases:

   - When the local transaction program just entered receive state and the remote transaction program issues MC_REQUEST_TO_SEND before it enters send state.

   - When the remote transaction program has just entered receive state by issuing the MC_PREPARE_TO_RECEIVE verb (not MC_RECEIVE_AND_WAIT), and then issuing MC_REQUEST_TO_SEND before the local transaction program responds by entering send state. This ambiguity can occur because MC_REQUEST_TO_SEND is an expedited request and this request can arrive ahead of the request carrying the SEND indication.

     The local transaction program might not be able to distinguish this case from the preceding case. The remote transaction program can avoid this

ambiguity by waiting until it receives information from the local transaction program before it issues the MC_REQUEST_ TO_SEND.

- When the remote transaction program issues the MC_REQUEST_TO_SEND in send state.

MC_REQUEST_TO_SEND

Notifies the remote transaction program that the local transaction program is requesting to enter send state for the mapped conversation. APPC/PC places the mapped conversation in send state when the local transaction program subsequently receives a SEND indication from the remote transaction program

MC_REQUEST_TO_SEND	Supplied Parameters: TP_ID (variable) CONV_ID (variable)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which the local transaction program is requesting to send data.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The MC_REQUEST_TO_SEND return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

- STATE_CHECK
  - R_T_S_NOT_RCV_STATE: The conversation is not in receive or confirm state.

For detailed information on the following return codes, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- OK.

State Changes:

None

Notes:

1. A transaction program enters receive state by issuing the MC_RECEIVE_AND_WAIT verb or the MC_PREPARE_TO_RECEIVE verb. The partner transaction program enters the corresponding send state when it issues an MC_RECEIVE_AND_WAIT or MC_RECEIVE_IMMEDIATE verb and receives the SEND indication (on the WHAT_RECEIVED parameter).

2. The REQUEST_TO_SEND_RECEIVED indication of YES is normally given to the remote transaction program when the conversation is in send state; that is, on an MC_SEND_DATA verb or on an MC_SEND_ERROR verb issued in send state. However, the YES indication can also be given on an MC_RECEIVE_AND_WAIT or MC_RECEIVE_IMMEDIATE verb. For more information, see "MC_RECEIVE_AND_WAIT" on page 6-46 or "MC_RECEIVE_IMMEDIATE" on page 6-53.

3. When the remote LU receives the MC_REQUEST_TO_SEND notification, it retains the

notification until the remote transaction program issues a verb with the REQUEST_TO_SEND_RECEIVED parameter.

The remote LU retains only one MC_REQUEST_TO_SEND notification at a time (for each conversation). The remote LU discards additional notifications until the LU can indicate the retained notification to the remote transaction program. Therefore the local transaction program can issue the MC_REQUEST_TO_SEND verb more times than the remote LU indicates to the remote transaction program.

MC_SEND_DATA

Sends one data record to the remote transaction program. The data format consists entirely of data.

MC_SEND_DATA	Supplied Parameters: TP_ID (variable) CONV_ID (variable) DATA_LENGTH (variable) DATA (variable)
	Returned Parameters: RETURN_CODE (variable) REQUEST_TO_SEND_RECEIVED (YES) (NO)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which the local transaction program is sending the data.

DATA_LENGTH specifies the length of the data record to be sent. The length of the data record can range from 0 to 65535 bytes. The local LU sends a null data record if the local transaction program specifies a data record length of 0. The sum of the DATA_LENGTH value and the offset portion of the DATA address cannot exceed 65535. This limit keeps the data from crossing a segment boundary.

DATA specifies the data record that the local transaction program is sending.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The MC_SEND_DATA return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.
  - DATA_AREA_ACROSS_SEGMENT: The data to be sent crosses a segment boundary.
- STATE_CHECK
  - SEND_DATA_NOT_SEND_STATE: The conversation is not in send state.

For detailed information on the following return codes, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes."

- ALLOCATION_ERROR
- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DEALLOCATE_ABEND
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- OK
- PROG_ERROR_PURGING.

REQUEST_TO_SEND_RECEIVED indicates whether the local LU has received a REQUEST_TO_SEND notification. The indication is either YES or NO.

- YES indicates that the local LU has received a REQUEST_TO_SEND notification from the remote

transaction program. The remote transaction program issues MC_REQUEST_TO_SEND to request the local transaction program to enter receive state and place the remote transaction program in send state.

- NO indicates that the local LU has not received a REQUEST_TO_SEND notification.

State Changes:

The state does not change when the RETURN_CODE indicates OK.

For information on state changes when the RETURN_CODE indicates other than OK, see "Understanding Mapped Conversation Return Codes" on page 6-3.

Notes:

1. The mapped conversation API provides for sending and receiving data records only. Unlike the logical records defined for the basic conversation API, data records contain only data; they do not contain the logical record length field.

2. The MC_SEND_DATA verb sends a complete data record. The sending program, therefore, cannot truncate a data record.

3. The local LU stores in a send buffer the data the local transaction program is sending. The local LU sends the data to the remote LU when the send buffer accumulates enough data for transmission. The local transaction program can force the LU to flush its send buffer by issuing the MC_FLUSH verb. The amount of data that is enough for transmission depends on the characteristics of the session allocated for the mapped conversation, and can vary from session to session.

4. When REQUEST_TO_SEND_RECEIVED indicates YES, the remote transaction program is requesting the local transaction program to enter receive state and thereby place the remote transaction program in send

state. A transaction program uses the MC_RECEIVE_AND_WAIT verb or the MC_PREPARE_TO_RECEIVE verb to enter receive state. The partner transaction program enters send state when it issues an MC_RECEIVE_AND_WAIT or MC_RECEIVE_IMMEDIATE verb and receives the SEND indication on the WHAT_RECEIVED parameter.

MC_SEND_ERROR

Informs the remote transaction program that the local transaction program found an error. If the conversation is in send state, the local LU flushes its send buffer.

After the successful completion of this verb, the local transaction program is in send state and the remote transaction program is in receive state. The transaction program must take the appropriate actions to correct the problem.

MC_SEND_ERROR	Supplied Parameters: TP_ID (variable) CONV_ID (variable)
	Returned Parameters: RETURN_CODE (variable) REQUEST_TO_SEND_RECEIVED (YES) (NO)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameters, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which the local transaction program is sending the error.

Returned Parameters:

RETURN_CODE indicates the result of verb execution.

APPC/PC returns the following MC_SEND_ERROR return codes independently of the state of the conversation:

- PARAMETER_CHECK

  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.

  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

  - DATA_AREA_ACROSS_SEGMENT: The data to be sent crosses a segment boundary.

  For detailed information on return codes listed below, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes."

  - APPC_ABENDED
  - APPC_BUSY
  - APPC_DISABLED
  - CONV_FAILURE_NO_RETRY
  - CONV_FAILURE_RETRY
  - CONVERSATION_TYPE_MIXED
  - INCOMPLETE
  - INCOMPLETE_ALTERED_VERB
  - OK.

If the transaction program issues MC_SEND_ERROR in send state, APPC/PC can report the following return codes:

- ALLOCATION_ERROR
- DEALLOCATE_ABEND
- PROG_ERROR_PURGING.

If the transaction program issues MC_SEND_ERROR in receive state, APPC/PC can report the following return code:

- DEALLOCATE_NORMAL: The remote transaction program issued a DEALLOCATE specifying the TYPE(SYNC_LEVEL) where the SYNC_LEVEL of the conversation is NONE or TYPE(FLUSH) parameter. This return code does not indicate an error condition.

REQUEST_TO_SEND_RECEIVED indicates whether the local LU has received a REQUEST_TO_SEND notification. The indication is either YES or NO.

- YES indicates that the local LU has received a REQUEST_TO_SEND notification from the remote transaction program. The remote transaction program has issued MC_REQUEST_TO_SEND, requesting the local transaction program to enter receive state and thereby place the remote transaction program in send state.

- NO indicates that the local LU has not received a REQUEST_TO_SEND notification.

State Changes:

The following state changes occur when the RETURN_CODE parameter indicates OK. For information on state changes when the RETURN_CODE indicates other than OK, see "Understanding Mapped Conversation Return Codes" on page 6-3.

The conversation enters send state after the local transaction program issues MC_SEND_ERROR in receive or confirm state.

No state change occurs when the local transaction program issues the verb in send state.

Notes:

1. The local transaction program can ensure that the remote transaction program receives the complete error notification as soon as possible by issuing MC_FLUSH immediately after MC_SEND_ERROR.

2. An MC_SEND_ERROR is reported to the remote transaction program as one of the following return codes:

   - PROG_ERROR_NO_TRUNC: The local transaction program issued MC_SEND_ERROR in

send state. No data truncation occurs at the mapped conversation API.

- PROG_ERROR_PURGING: The local transaction program issued MC_SEND_ERROR in receive state and all data sent by the remote transaction program and not yet received by the local transaction program is purged. This return code can also occur if the local transaction program issued MC_SEND_ERROR in confirm state, in which case no data is purged.

3. When a transaction program issues MC_SEND_ERROR in receive state, all incoming information is purged. The incoming information that is purged includes the following return code indications:

   - ALLOCATION_ERROR
   - DEALLOCATE_ABEND
   - PROG_ERROR_PURGING.

   APPC/PC reports the DEALLOCATE_NORMAL return code instead of ALLOCATION_ERROR or DEALLOCATE_ABEND. It reports OK instead of PROG_ERROR_PURGING.

   The other kinds of incoming information that are purged are:

   - Data sent by a transaction program issuing the MC_SEND_DATA verb.

   - Confirmation requests sent by a transaction program issuing the MC_CONFIRM verb.

   If a transaction program sends a confirmation request in conjunction with the MC_DEALLOCATE verb by specifying the TYPE(SYNC_LEVEL) parameter, then the deallocation request is also purged.

   However, incoming REQUEST_TO_SEND indications are not purged.

4. When the returned REQUEST_TO_SEND_RECEIVED parameter indicates YES, the remote transaction program is requesting the local transaction program to enter receive state and thereby place the remote program in send state. A transaction program enters receive state by issuing the MC_RECEIVE_AND_WAIT or MC_PREPARE_TO_RECEIVE verb.

   The partner transaction program enters send state when it issues an MC_RECEIVE_AND_WAIT or MC_RECEIVE_IMMEDIATE verb and receives the SEND indication on the WHAT_RECEIVED parameter.

5. The local transaction program can use MC_SEND_ERROR for various application-level functions. For example, the local transaction program can issue this verb to inform the remote transaction program of an error it found in the data records it received, or to reject a confirmation request.

MC_TEST

Tests the specified conversation to determine if the local LU has received a REQUEST_TO_SEND notification from a remote transaction program. The return code indicates the result of the test.

MC_TEST	Supplied Parameters: TP_ID (variable) CONV_ID (variable)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation the local transaction program is checking for a REQUEST_TO_SEND indication.

Returned Parameters:

RETURN_CODE indicates the result of the test. The MC_TEST return codes are:

- OK: The local LU has received a REQUEST_TO_SEND notification. The remote program has issued MC_REQUEST_TO_SEND, requesting the local program to enter receive state and thereby place the remote program in send state.
- PARAMETER_CHECK

- BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.

- BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

- UNSUCCESSFUL: The local LU has not received a REQUEST_TO_SEND notification.

For detailed information on the following return codes, see "Understanding Mapped Conversation Return Codes" on page 6-3 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB.

State Changes:

None

Notes:

1. When the local LU receives a REQUEST_TO_SEND notification, the remote transaction program is requesting the local transaction program to enter receive state and thereby place the remote transaction program in send state. A transaction program enters receive state by issuing the MC_RECEIVE_AND_WAIT or MC_PREPARE_TO_RECEIVE verb.

2. The partner transaction program enters send state after it issues an MC_RECEIVE_AND_WAIT or MC_RECEIVE_IMMEDIATE verb and then receives the SEND indication on the WHAT_RECEIVED parameter.

Chapter 7. Using Transaction Basic Conversation Verbs

This chapter describes the category of verbs called *basic conversation verbs* that are available for use by transaction programs. Basic conversation verbs define the basic conversation API for end-user program-to-program support.

Before the detailed descriptions of the basic conversation verbs is a discussion of the conversation states at the basic conversation API, and a description of the basic conversation return codes. These subjects apply generally to all of the basic conversation verbs.

Note:

Every conversation is either a basic or a mapped conversation. A transaction program cannot use the basic conversation verbs and mapped conversation verbs on the same conversation. However, a transaction program providing its own mapped conversation layer can use a basic conversation to perform mapped conversation operations.

Understanding Basic Conversation States

The selection of verbs that a program can issue for a particular conversation depends on the state of the conversation. As the program issues verbs, the state of the conversation can change. This state change occurs in response to a verb issued by the local program, a verb issued by the remote program, or as a result of network errors.

APPC/PC defines the state of a conversation in terms of the local program's view of the local end of the conversation. The local end of the conversation is the end to which the local program is connected. The states of other conversations allocated to the program can be different. For example, one conversation can be in receive state and another in send state, concurrently.

The state of the basic conversation determines which verbs APPC/PC allows a program to issue. The folowing table correlates the verbs, and parameters if applicable, to the basic conversation states.

Verb	Conversation States at Basic Conversation Protocol Boundary			
	Reset	**Send**	**Re-ceive**	**Con-firm**
ALLOCATE	Yes	n/a	n/a	n/a
CONFIRM	n/a	Yes	No	No
CONFIRMED	n/a	No	No	Yes
DEALLOCATE with TYPE(FLUSH) or TYPE(SYNC_LEVEL)	n/a	Yes	No	No
DEALLOCATE with TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER)	n/a	Yes	Yes	Yes
FLUSH	n/a	Yes	No	No
GET_ATTRIBUTES	n/a	Yes	Yes	Yes
GET_TYPE	n/a	Yes	Yes	Yes
POST_ON_RECEIPT	n/a	No	Yes	No
PREPARE_TO_RECEIVE	n/a	Yes	No	No
RECEIVE_AND_WAIT	n/a	Yes	Yes	No
RECEIVE_IMMEDIATE	n/a	No	Yes	No
REQUEST_TO_SEND	n/a	No	Yes	Yes
SEND_DATA	n/a	Yes	No	No
SEND_ERROR	n/a	Yes	Yes	Yes
TEST with TEST(POSTED)	n/a	No	Yes	No
TEST with TEST(REQUEST_TO_SEND_RECEIVED)	n/a	Yes	Yes	Yes
WAIT	n/a	No	Yes	No

At the intersection of each verb row and state column, the table indicates Yes, No, or n/a. Yes means that APPC/PC allows the program to issue the verb when the conversation is in that state.

No means the program cannot issue the verb because APPC/PC disallows the verb in that state. APPC/PC treats a verb issued for a conversation in a disallowed state as a state-check condition and issues an appropriate return code. The individual verb descriptions list the applicable state-check conditions.

n/a means the state is not applicable either because it cannot exist when the verb is issued or because the state is not relevant to the verb.

A conversation enters a particular state when the transaction program issues a verb that causes a state transition or when the program receives a return code that indicates that a state transition has occurred. This chapter defines the specific state transitions under the following heading, "Understanding Basic Conversation Return Codes," and in each verb description under the headings "Returned Parameters" and "State Changes."

Understanding Basic Conversation Return Codes

All conversation verbs have a parameter called RETURN_CODE that APPC/PC uses to pass a return code back to the program when the LU finishes executing a verb. The return code indicates the result of verb execution, including any state changes to the specified basic conversation. For information on which verbs a program can issue in each state, see "Understanding Basic Conversation States" on page 7-2.

The structure of a return code is a 2-byte primary code that identifies the error type, and a 4-byte secondary code that provides more detailed error information.

Some of the return codes indicate results of the local LU's processing of a verb; APPC/PC returns these return codes on the verb that initiated the local processing. Other return codes indicate results of processing initiated at the remote end of the conversation..

Depending on the verb, APPC/PC returns these codes on the verb that initiated the remote processing or on a later verb. Still other return codes report events occurring at the remote end of the conversation. In any case, APPC/PC returns only one code at a time.

The following return codes can be returned on one or more basic conversation verbs:

- ALLOCATION_ERROR
- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DATA_POSTING_BLOCKED
- DEALLOCATE_ABEND_PROG
- DEALLOCATE_ABEND_SVC
- DEALLOCATE_ABEND_TIMER

- DEALLOCATE_NORMAL
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- INVALID_VERB
- OK
- POSTING_NOT_ACTIVE
- PROG_ERROR_NO_TRUNC
- PROG_ERROR_PURGING
- PROG_ERROR_TRUNC
- SVC_ERROR_NO_TRUNC
- SVC_ERROR_PURGING
- SVC_ERROR_TRUNC.

Detailed descriptions of these return codes follow. Brief references to these return codes appear in the individual verb descriptions later in this chapter. For a description of unique parameters or return codes, see the individual verb descriptions.

ALLOCATION_ERROR indicates that the local transaction program issued an ALLOCATE and APPC/PC could not complete the allocation of the specified conversation. The ALLOCATION_ERROR indication and one of the following secondary return codes form the complete return code that APPC/PC returns to the transaction program; the secondary return code identifies the specific error.

The remote LU and the remote transaction program referred to in the following secondary return code definitions are the LU specified in the LU_NAME parameter and the program specified in the TPN parameter, respectively, of the ALLOCATE verb.

APPC/PC reports allocation errors caused by the local LU failing to obtain a session for the conversation on the ALLOCATE verb, and by the remote LU rejecting the allocation request on a subsequent verb. When APPC/PC returns an ALLOCATION_ERROR to the transaction program, it places the basic conversation in reset state.

The ALLOCATION_ERROR secondary return codes are as follows:

- **ALLOCATION_FAILURE_NO_RETRY** indicates that APPC/PC cannot allocate the basic conversation on a session because of a permanent condition. For example, APPC/PC cannot activate the session to be used for the basic conversation because the current mode session limit for the specified partner LU is 0; because of a system definition error or a session-activation protocol error; or because APPC/PC deactivated the session in response to a session protocol error before it could allocate the basic conversation. The transaction program should not try the allocation request again until the condition is corrected.

- **ALLOCATION_FAILURE_RETRY** indicates that APPC/PC cannot allocate the basic conversation on a session because of a temporary condition. For example, APPC/PC cannot allocate the session for the basic conversation because of a temporary lack of sessions at the local LU or remote LU. The error can also occur if APPC/PC deactivates the session because of a line or modem failure before it could allocate the conversation.

  The condition is temporary; the transaction program can try the allocation request again. However, to avoid congesting the network with attempted allocation requests, the transaction program should pause or wait for a keystroke before retrying the transaction.

- **CONVERSATION_TYPE_MISMATCH** indicates that the remote LU rejects the allocation request because the remote transaction program does not support basic conversations. APPC/PC returns this return code on a subsequent verb.

- **PIP_NOT_ALLOWED** indicates that the remote LU rejects the allocation request because the local program specified program initialization parameters (by setting the PIP_DATA_LENGTH parameter to a non-0 value) and either the remote LU does not support PIP data, or the remote transaction program

has no PIP variables defined. APPC/PC returns this return code on a subsequent verb.

- **PIP_NOT_SPECIFIED_CORRECTLY** indicates that the remote LU rejects the allocation request because the remote transaction program has one or more PIP variables defined and the local transaction program has specified that there are no program initialization parameters (by setting the PIP_DATA_LENGTH parameter of the ALLOCATE verb to 0).

  This error can also indicate that the local transaction program has not specified program initialization parameters (by setting the PIP_DATA_LENGTH parameter to a non-0 value) that do not correspond in number to those defined for the remote transaction program. APPC/PC returns this return code on a subsequent verb.

- **SECURITY_NOT_VALID** indicates that the remote LU rejects the allocation request because access security information (specified using the ALLOCATE SECURITY parameter) is invalid. APPC/PC returns this return code on a subsequent verb.

- **SYNC_LEVEL_NOT_SUPPORTED** indicates that the remote LU rejects the allocation request because the local program specified a synchronization level (using the SYNC_LEVEL parameter) that the remote program does not support. APPC/PC returns this return code on a subsequent verb.

- **TPN_NOT_RECOGNIZED** indicates that the remote LU rejects the allocation request because the local transaction program specified a remote transaction program name that the remote LU does not recognize. APPC/PC returns this return code on a subsequent verb.

- **TRANS_PGM_NOT_AVAIL_NO_RETRY** indicates that the remote LU rejects the allocation request because the local transaction program specified a remote transaction program that the remote LU

recognizes but cannot start. The condition is permanent; the transaction program should not try the allocation request again. APPC/PC returns this return code on a subsequent verb.

- **TRANS_PGM_NOT_AVAIL_RETRY** indicates that the remote LU rejects the allocation request because the local transaction program specified a remote transaction program that the remote LU recognizes but currently cannot start. The condition is temporary; the transaction program can try the conversation again. APPC/PC returns this return code on a subsequent verb.

APPC_ABENDED indicates that APPC/PC has been abnormally terminated.

APPC_BUSY indicates that APPC/PC is currently executing another verb and cannot execute this verb. This error can occur if a verb is issued after APPC/PC execution is interrupted (for example, by a Ctrl-Break or timer interrupt).

APPC_DISABLED indicates that APPC/PC is disabled as a result of the DISABLE/ENABLE_APPC verb.

CONVERSATION_TYPE_MIXED indicates that the local program issued a combination of basic conversation verbs and mapped conversation verbs on the same conversation. APPC/PC reports this return code on the verb issued.

CONV_FAILURE_NO_RETRY indicates that a failure occurred that caused APPC/PC to terminate the conversation prematurely. For example, APPC/PC deactivated the session that the transaction programs were using for the conversation because of a session protocol error. The condition is permanent; the transaction program should not try the transaction again until the condition is corrected. APPC/PC can report this return code to the local transaction program on a verb the program issues in any state other than reset. APPC/PC leaves the conversation in reset state.

CONV_FAILURE_RETRY indicates that a failure occurred that caused APPC/PC to terminate the conversation prematurely. For example, APPC/PC terminates conversations when it must deactivate the associated sessions in response to a line or modem failure. The condition is temporary; the transaction program can try the conversation again. However, to avoid congesting a network with attempted allocation requests, the transaction program should pause or wait for a keystroke before retrying the transaction. APPC/PC can report this return code to the local program on a verb that the program issues in any state other than reset. APPC/PC places the conversation in reset state.

DATA_POSTING_BLOCKED indicates that a transaction program issued a WAIT or TEST verb and APPC/PC cannot post one of the active conversations because the APPC/PC workspace storage assigned to a session is filled, and the transaction program is unable to send a pacing response. The condition can be corrected by issuing a RECEIVE_IMMEDIATE or RECEIVE_AND_WAIT on one of the conversations specified in the TEST or WAIT verb. APPC/PC reports this return code only if receive pacing is active.

DEALLOCATE_ABEND_PROG indicates that the remote transaction program issued a DEALLOCATE verb specifying the TYPE(ABEND_PROG) parameter. The remote LU can also issue a DEALLOCATE verb specifying the TYPE(ABEND_PROG) parameter in response to a remote transaction program abnormal termination condition. If the conversation for the remote transaction program is in receive state when the remote transaction program or LU issues a DEALLOCATE verb, information sent by the local transaction program and not yet received by the remote transaction program is purged.

APPC/PC returns the DEALLOCATE_ABEND_PROG return code to the local transaction program on a verb the program issues in either send or receive state. After APPC/PC issues this return code, it deallocates the conversation, making the conversation unavailable to the program.

DEALLOCATE_ABEND_SVC indicates that the remote transaction program issued a DEALLOCATE verb specifying the TYPE(ABEND_SVC) parameter. If the conversation for the remote transaction program is in receive state when the remote transaction program issues a DEALLOCATE verb, information sent by the local transaction program and not yet received by the remote transaction program is purged.

APPC/PC returns the DEALLOCATE_ABEND_SVC return code to the local transaction program on a verb the program issues in either send or receive state. After APPC/PC issues this return code, it deallocates the conversation, making the conversation unavailable to the program.

DEALLOCATE_ABEND_TIMER indicates that the remote transaction program issued a DEALLOCATE verb specifying the TYPE(ABEND_TIMER) parameter. If the conversation for the remote transaction program is in receive state when the program issues a DEALLOCATE verb, information sent by the local transaction program and not yet received by the remote transaction program is purged.

APPC/PC returns the DEALLOCATE_ABEND_TIMER return code to the local transaction program on a verb the program issues in either send or receive state. After APPC/PC issues this return code, it deallocates the conversation, making the conversation unavailable to the program.

DEALLOCATE_NORMAL indicates that the remote transaction program issued a DEALLOCATE verb specifying the TYPE(SYNC_LEVEL) or TYPE(FLUSH) parameter. APPC/PC reports this return code to the local transaction program on a verb the program issues in receive state. After APPC/PC issues this return code, it deallocates the conversation, making the conversation unavailable to the program.

INCOMPLETE indicates that the verb has not finished and must be re-issued unchanged before any other verb with the same TP_ID. Before re-issuing the verb, you

should try to issue verbs on other transaction programs, including other unfinished verbs. If you are queueing incoming ALLOCATEs in the LUs, you should also periodically issue GET_ALLOCATE. This return code is returned only if ATTACH_PU (RETURN_CONTROL=INCOMPLETE) was issued. For more information on this return code, see "System Deadlocks" on page 10-2.

INCOMPLETE_ALTERED_VERB indicates that a verb was issued with the same TP_ID as that of an unfinished verb, or the unfinished verb was altered before it was re-issued.

Note: You may change the first 12 bytes of an incomplete verb so you can place list pointers in this area to create a list of incomplete verbs.

INVALID_VERB indicates that APPC/PC did not recognize the issued verb. APPC/PC reports this return code on the verb issued. The state of the conversation remains unchanged.

OK indicates that the verb issued by the local transaction program executed successfully. That is, APPC/PC performed the function defined for the verb, to the point where APPC/PC returns control to the transaction program. The state of the basic conversation is as defined for the verb.

POSTING_NOT_ACTIVE indicates that a transaction program issued a WAIT verb for which none of the listed conversations had an active POST_ON_RECEIPT outstanding. APPC/PC reports this return code on the WAIT verb.

PROG_ERROR_NO_TRUNC indicates that the remote transaction program issued a SEND_ERROR specifying the TYPE(PROG) parameter, the conversation for the remote transaction program was in send state, and the SEND_ERROR did not truncate a logical record. APPC/PC does not truncate a record if the remote transaction program issues SEND_ERROR before sending logical records or after sending a complete logical record.

APPC/PC reports this return code to the local transaction program on a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb which the local transaction program issues before receiving any logical records or after receiving one or more complete logical records. The conversation remains in receive state.

PROG_ERROR_PURGING indicates that the remote transaction program issued a SEND_ERROR specifying the TYPE(PROG) parameter, and the conversation for the remote transaction program was in receive or confirm state. The SEND_ERROR may cause data sent by the local transaction program to be purged.

Purging occurs when a transaction program issues SEND_ERROR in receive state before it receives all the information sent by its partner transaction program (that is, the information sent before APPC/PC reports the error). The purging can occur at the local LU, remote LU, or both. No purging occurs when a program issues SEND_ERROR in confirm state or receive state after receiving all the information sent by its partner transaction program.

APPC/PC normally reports this return code to the local transaction program on a verb the local transaction program issues after sending information to the remote program. However, APPC/PC can report this return code on a verb the transaction program issues before sending any information, depending on the verb and when the program issues it. APPC/PC leaves the conversation in receive state.

PROG_ERROR_TRUNC indicates that the remote transaction program issued a SEND_ERROR specifying the TYPE(PROG) parameter, the conversation for the remote transaction program was in send state, and the SEND_ERROR truncated a logical record. Truncation occurs when a transaction program begins sending a logical record and then issues SEND_ERROR before sending the complete logical record.

APPC/PC reports this return code to the local program on a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE

verb the program issues after receiving the truncated logical record. APPC/PC leaves the conversation in receive state.

SVC_ERROR_NO_TRUNC indicates that the remote transaction program issued a SEND_ERROR specifying the TYPE(SVC) parameter, the conversation for the remote transaction program was in send state, and the SEND_ERROR did not truncate a logical record. No truncation occurs when a transaction program issues SEND_ERROR before sending logical records or after sending a complete logical record.

APPC/PC reports this return code to the local transaction program on a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb the program issues before receiving any logical records or after receiving one or more complete logical records. APPC/PC places the conversation in receive state.

SVC_ERROR_PURGING indicates that the remote transaction program issued a SEND_ERROR specifying the TYPE(SVC) parameter, and that the conversation for the remote transaction program was in receive or confirm state. The SEND_ERROR may cause information to be purged.

Purging occurs when a transaction program issues SEND_ERROR in receive state before it receives all the information sent by its partner transaction program (that is, the information sent before APPC/PC reports the SVC_ERROR_PURGING return code to the partner program). The purging can occur at the local LU, remote LU, or both. No purging occurs when a transaction program issues SEND_ERROR in confirm or receive state after receiving all the information sent by its partner transaction program.

APPC/PC normally reports this return code to the local transaction program on a verb the program issues after sending information to the remote transaction program. However, APPC/PC can report the return code on a verb that the transaction program issues before it sends any information, depending on the verb and when the

transaction program issues it. APPC/PC leaves the conversation in receive state.

SVC_ERROR_TRUNC indicates that the remote transaction program issued a SEND_ERROR specifying the TYPE(SVC) parameter, the conversation for the remote transaction program was in send state, and the SEND_ERROR truncated a logical record. Truncation occurs when a transaction program begins sending a logical record and then issues SEND_ERROR before sending the complete logical record.

APPC/PC reports this return code to the local transaction program on a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb that the program issues after receiving the truncated logical record. APPC/PC leaves the conversation in receive state.

Note:

The PARAMETER_CHECK and STATE_CHECK errors can also occur on any verb. When APPC/PC detects these error conditions, it does not try to execute the verb. A PARAMETER_CHECK error occurs when APPC/PC detects an invalid parameter value. A STATE_CHECK error occurs when the conversation is not in the correct state for the verb the transaction program is issuing. The verb descriptions list the secondary return codes which identify the causes of these errors.

The following table correlates the verbs to the return codes on which they can be returned.

Return Codes	Verbs															
	ALLOCATE	CONFIRM	CONFIRMED	DEALLOCATE	FLUSH	GET ATTRIBUTES	GET TYPE	POST ON RECEIPT	PREPARE TO RECEIVE	RECEIVE AND WAIT	RECEIVE IMMEDIATE	REQUEST TO SEND	SEND DATA	SEND ERROR	TEST	WAIT
ALLOCATION_ERROR	X	X		X					X	X	X		X			
CONV_FAILURE_NO_RETRY		X		X					X	X	X		X	X		
CONV_FAILURE_RETRY		X		X					X	X	X		X	X		
CONVERSATION_TYPE_MIXED		X	X	X	X	X		X	X	X	X	X	X	X	X	X
DATA_POSTING_BLOCKED															X	X
DEALLOCATE_ABEND_PROG		X		X					X	X	X		X			
DEALLOCATE_ABEND_SVC		X		X					X	X	X		X	X		
DEALLOCATE_ABEND_TIMER		X		X					X	X	X		X	X		
DEALLOCATE_NORMAL										X	X			X		
INCOMPLETE	X	X		X					X	X	X		X	X	X	X
INCOMPLETE_ALTERED_VERB	X	X		X					X	X	X		X	X	X	X
OK	X	X		X					X	X	X		X	X	X	X
PARAMETER_CHECK	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
POSTING_NOT_ACTIVE																X
PROG_ERROR_NO_TRUNC										X	X					
PROG_ERROR_PURGING		X		X					X	X	X		X	X		
PROG_ERROR_TRUNC										X	X					
STATE_CHECK		X	X	X	X			X	X	X	X		X	X	X	X
SVC_ERROR_NO_TRUNC										X	X					
SVC_ERROR_PURGING		X		X					X	X	X		X	X		
SVC_ERROR_TRUNC										X	X					
UNSUCCESSFUL	X										X				X	X

Each X in the table means that APPC/PC can return the indicated return code with the corresponding verb.

The individual verb descriptions list the applicable return codes. The descriptions for most verbs do not explicitly list the secondary return code of the ALLOCATION_ERROR return code because they can occur at any time and are not necessarily a result of the verb on which they are returned. The description of the ALLOCATE verb, however, lists the two secondary codes that indicate an allocation failure.

Verb Descriptions

Detailed descriptions of the basic conversation verbs are as follows:

ALLOCATE

Allocates a session, if one is available, between the local LU and a remote LU, and establishes a basic or mapped conversation between the local transaction program and a remote transaction program on that session. APPC/PC assigns a CONV_ID to the conversation. Except for the first conversation of a remotely initiated transaction program, the transaction program must issue this verb before it can issue any verbs that refer to the conversation.

A program can request the allocation of a mapped conversation and then issue basic conversation verbs on a mapped conversation. However, the local transaction program is then responsible for providing its own mapping layer.

For example, you want to provide your mapping layer in order to do data mapping, or to permit MC_POST_ON_RECEIPT and MC_WAIT, neither of which are implemented by the APPC/PC mapped conversation API.

ALLOCATE	Supplied Parameters: TP_ID (variable) PARTNER_LU_NAME (variable) MODE_NAME (variable) TPN (variable) CONVERSATION_TYPE (BASIC_CONVERSATION) (MAPPED_CONVERSATION) RETURN_CONTROL (WHEN_SESSION_ALLOCATED) (IMMEDIATE) (WHEN_SESSION_FREE) SYNC_LEVEL (NONE) (CONFIRM) SECURITY (NONE) (SAME) (PGM (USER_ID (variable) PASSWORD (variable)) PIP_DATA_LENGTH (variable) PIP_DATA (variable)
	Returned Parameters: CONV_ID (variable) RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

PARTNER_LU_NAME specifies the name (in EBCDIC) of the remote LU at which the remote transaction program is located. The local LU uses the PARTNER_LU_NAME to identify the remote LU when allocating a session. That is, the PARTNER_LU_NAME is the remote LU network name.

MODE_NAME specifies the mode name (in EBCDIC) designating the network properties for the session APPC/PC is to allocate for the conversation. For example, one network property is the class of service APPC/PC provides for the session used. Basic conversations cannot use the SNASVCMG mode.

TPN specifies the name (in EBCDIC) of the remote transaction program that manages the other end of the conversation.

CONVERSATION_TYPE specifies the type of conversation APPC/PC is to allocate.

- **BASIC_CONVERSATION** directs APPC/PC to allocate a basic conversation.

- **MAPPED_CONVERSATION** directs APPC/PC to allocate a mapped conversation.

RETURN_CONTROL specifies when the local LU is to return control to the local transaction program after the program requests the LU to allocate a session for the conversation. APPC/PC reports allocation errors caused by the local LU failing to obtain a session for the conversation on this verb. APPC/PC also reports allocation errors caused by the remote LU rejecting the conversation request on a subsequent verb. The valid values for the RETURN_CONTROL parameter are:

- **WHEN_SESSION_ALLOCATED** directs the LU to allocate a session for the conversation before returning control to the transaction program. Except for instances of conversation failure, the LU does not return control to the transaction program until a session becomes available.

  This option may cause a deadlock situation if there are not enough sessions for the currently active conversations of all transaction programs. Specifying the RETURN_CONTROL(INCOMPLETE) option on the ATTACH_PU verb can confine the problem to the currently executing transaction program, but if this program has itself used up all sessions, the program

can still cause a deadlock situation by using the WHEN_SESSION_ALLOCATED option. You must analyze your operating environment to determine if this option is safe. For more information on this problem, see "System Deadlocks" on page 10-2.

- **IMMEDIATE** directs the LU to allocate a session for the conversation if a contention-winner session is immediately available. This option returns control to the transaction program immediately with a return code indicating whether a session is allocated. In this case, the local LU does not have to wait for a response from the partner LU, and the ALLOCATE verb does not require information to be sent to the partner LU.

  - A return code of OK indicates that a session is immediately available and is allocated for the conversation. A session is immediately available when it is active, it is not allocated to another conversation, and the local LU is the contention winner for the session.

  - A return code of UNSUCCESSFUL indicates that a session is not immediately available and that the LU did not perform the allocation.

- **WHEN_SESSION_FREE** directs the LU to allocate a session for the conversation only if a session (either a contention winner or loser) is available or able to be activated, and then to return control to the transaction program with a return code indicating whether a session is allocated.

  - A return code of OK indicates that a session is available and is allocated for the conversation. If activation of a session is necessary, the LU performs the session activation before returning control to the transaction program.

  - A return code of ALLOCATION_ERROR indicates that a session is unavailable and that the LU cannot activate a new session. No session is allocated.

SYNC_LEVEL specifies the synchronization level that the local and remote programs can use for this conversation.

- **NONE** specifies that the transaction programs do not perform confirmation processing on this conversation. The programs do not issue any verbs and do not recognize returned parameters relating to the synchronization function.

- **CONFIRM** specifies that the transaction programs can perform confirmation processing on this conversation. The programs can issue verbs and recognize returned parameters relating to confirmation.

SECURITY specifies access security information that the remote LU uses to validate access to the remote transaction program and its resources. The access security information includes a user ID and a password provided using the following arguments:

- **NONE** specifies that this conversation does not use conversation level security.

- **SAME** directs the remote LU to use the user ID it has already verified as part of the allocation request that initiated execution of the local program. This option also causes APPC/PC to send this saved user ID to the remote LU with an 'already verified' indication. If there was no user ID, APPC/PC sends the allocation request with no security information and a 'not already verified' indication.

- **PGM** directs the remote LU to use the security information that the local transaction program provides on this parameter. The local transaction program uses the following arguments to provide this security information:

  - **USER_ID** specifies the ID (in EBCDIC) of the end user. The USER_ID value is 8 bytes long. The remote LU uses this value and the password to verify the identity of the end user making the allocation request. In addition, the remote LU may

use the USER_ID for auditing or accounting to associate conversation accesses with the end user.

- **PASSWORD** specifies the password (in EBCDIC) of the end user. The PASSWORD value can be up to 8 bytes long. The remote LU uses this value and the USER_ID to verify the identity of the end user making the allocation request.

PIP_DATA_LENGTH specifies the length of the program initialization parameters for the remote program. Set this parameter to 0 if there is no PIP data.

PIP_DATA specifies the variable that contains the PIP data that the local program is sending to the remote program. The transaction program *must* format the PIP data according to the GDS format specified in "FM Headers" in the *SNA Reference Summary.*

Returned Parameters:

CONV_ID indicates the ID of the new conversation.

RETURN_CODE indicates the result of verb execution. The ALLOCATE return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - DATA_AREA_ACROSS_SEGMENT: APPC/PC does not permit PIP data to cross a segment boundary.
  - BAD_TPN_LEN: The value that TPN specifies is too short (less than 1) or too long (greater than 64).
  - BAD_CONV_TYPE: APPC/PC does not recognize the specified CONVERSATION_TYPE.
  - BAD_SYNC_LEVEL: APPC/PC does not recognize the specified SYNC_LEVEL.

- BAD_SECURITY_SELEC: APPC/PC does not recognize the specified SECURITY.

- BAD_RETURN_CONTROL: APPC/PC does not recognize the specified RETURN_CONTROL.

- TOO_BIG_SEC_TOKENS: APPC/PC does not accept a password or user ID longer than 8 bytes.

- PIP_LEN_INCORRECT: APPC/PC does not accept PIP data longer than 32767 bytes.

- NO_USE_OF_SNASVCMG: APPC/PC does not accept SNASVCMG as the value for the MODE_NAME parameter.

- UNKNOWN_PARTNER_MODE: APPC/PC does not recognize the specified PARTNER_LU_NAME or MODE_NAME.

- ALLOCATION_ERROR: The following two secondary return codes indicate the local transaction programs failure to obtain a conversation. For detailed information on these secondary return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

  - ALLOCATION_FAILURE_NO_RETRY: APPC/PC cannot allocate the conversation because of a permanent error condition.

  - ALLOCATION_FAILURE_RETRY: APPC/PC cannot allocate the conversation because of temporary error condition.

- UNSUCCESSFUL: The program specified RETURN_CONTROL(IMMEDIATE) and APPC/PC could not allocate the conversation because no contention-winner sessions were available.

For detailed information on the following return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- OK.

State Changes:

The conversation enters send state if the RETURN_CODE indicates OK.

Notes:

1. The local LU does not send PIP data immediately unless the local transaction program issues a verb (other than SEND_DATA) that explicitly directs the LU to flush its send buffer. Otherwise, the LU retains the PIP parameter of the ALLOCATE verb and accumulates data from subsequent SEND_DATA verbs.

   The LU sends this data to the partner LU when it accumulates enough data for transmission. The amount of information that is sufficient for transmission depends on the characteristics of the session allocated for the conversation, and can vary from session to session.

2. The local transaction program can ensure that APPC/PC connects the remote transaction program as soon as possible by issuing the FLUSH verb immediately after the ALLOCATE verb.

3. Contention for a session can occur when two LUs connected by a session both try to allocate a conversation on the session at the same time. APPC/PC resolves the contention by making one LU the contention winner of the session and the other LU the contention loser of the session.

   The contention-winner LU allocates a conversation on a session without asking permission from the contention-loser LU. Conversely, the contention-loser

LU requests permission from the contention-winner LU to allocate a conversation on the session. Then the contention-winner LU either grants or rejects the request.

4. The remote transaction program starts the conversation in receive state.

5. For an IBM Token-Ring Network, one link must be reserved for outgoing calls to enable the ALLOCATE verb to complete. For more information on reserving links, see "Entering Information for an IBM Token-Ring DLC" in the *APPC/PC Installation and Configuration Guide*

6. You must specify the PARTNER_LU_NAME, MODE_NAME, TPN, USER_ID, and PASSWORD in EBCDIC. You can use the CONVERT verb to convert these parameter values from ASCII to EBCDIC.

CONFIRM

Sends a confirmation request to a remote transaction program and waits for a reply. This verb enables the local and remote programs to synchronize their processing with one another. CONFIRM also causes the LU to flush its send buffer. To use this verb, the program must have allocated the conversation with a synchronization level of CONFIRM.

CONFIRM	Supplied Parameters: TP_ID (variable) CONV_ID (variable)
	Returned Parameters: RETURN_CODE (variable) REQUEST_TO_SEND_RECEIVED (YES) (NO)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which the transaction program is sending a confirmation request.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The CONFIRM return codes are as follows:

- OK: The remote transaction program replied CONFIRMED.
- PARAMETER_CHECK

- BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.

- BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

- CONFIRM_ON_SYNC_NONE: APPC/PC does not permit the transaction program to use this verb if it allocated the conversation with SYNC_LEVEL(NONE).

- STATE_CHECK

  - CONFIRM_BAD_STATE: The conversation is not in send state.

  - CONFIRM_NOT_LL_BDY: The conversation is in send state, and the transaction program started, but did not finish, sending a logical record.

For detailed information on the following return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- ALLOCATION_ERROR
- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DEALLOCATE_ABEND_PROG
- DEALLOCATE_ABEND_SVC
- DEALLOCATE_ABEND_TIMER
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- PROG_ERROR_PURGING
- SVC_ERROR_PURGING.

REQUEST_TO_SEND_RECEIVED indicates whether the local LU received a REQUEST_TO_SEND verb. The two indications APPC/PC can return are YES and NO.

- YES indicates that the local LU received a REQUEST_TO_SEND notification from the remote transaction program. The remote program issued a REQUEST_TO_SEND verb to request the local program to enter receive state and place the remote program in send state.

- NO indicates that the local LU has not received a REQUEST_TO_SEND notification from the remote program.

State Changes:

The state of the conversation does not change when the RETURN_CODE indicates OK. For information on state changes when the RETURN_CODE indicates a code other than OK, see "Understanding Basic Conversation Return Codes" on page 7-5.

Notes:

1. The transaction program can use this verb for various application-level functions. For example:

   - The transaction program can issue this verb immediately following an ALLOCATE to determine whether the allocation of the conversation is successful before sending data.

   - The transaction program can issue this verb to request acknowledgment of data it sent to the remote transaction program. The remote program can respond by issuing CONFIRMED to indicate that it received and processed the data without error, or by issuing SEND_ERROR to indicate that it found an error.

2. When the REQUEST_TO_SEND_RECEIVED parameter indicates YES, the remote transaction program is requesting that the local transaction program enter receive state and place the remote transaction program in send state. A transaction program enters receive state by issuing the PREPARE_TO_RECEIVE verb or the

RECEIVE_AND_WAIT verb. The partner transaction program enters send state after it issues the RECEIVE_AND_WAIT verb or the RECEIVE_IMMEDIATE verb, and receives the SEND indication from its partner transaction program on the WHAT_RECEIVED parameter.

CONFIRMED

Sends a confirmation reply to the remote transaction program. This verb enables the local and remote programs to synchronize their processing. The local program can issue this verb when it receives a confirmation request. (For more information on confirmation requests, see the WHAT_RECEIVED parameter of the RECEIVE_AND_WAIT or the RECEIVE_IMMEDIATE verbs).

CONFIRMED	Supplied Parameters: TP_ID (variable) CONV_ID (variable)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which the transaction program is sending a confirmation reply.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The CONFIRMED return codes are as follows:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.

- BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

- STATE_CHECK

  - CONFIRMED_BAD_STATE: The conversation is not in confirm state.

For detailed information on the following return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- OK.

State Changes:

The conversation enters receive state if the transaction program received CONFIRM on the preceding RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb.

The conversation enters send state if the transaction program received CONFIRM_SEND on the preceding RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb.

The conversation enters reset state if the transaction program received CONFIRM_DEALLOCATE on the preceding RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb.

Note:

The transaction program can issue this verb only as a reply to a confirmation request.

DEALLOCATE

Deallocates the specified conversation. DEALLOCATE can perform the function of the FLUSH or the CONFIRM verb before it deallocates the conversation. APPC/PC discards the CONV_ID after APPC/PC deallocates the associated conversation.

DEALLOCATE	Supplied Parameters: TP_ID (variable) CONV_ID (variable) TYPE (SYNC_LEVEL) (FLUSH) (ABEND_PROG) (ABEND_SVC) (ABEND_TIMER) LOG_DATA_LENGTH (variable) LOG_DATA (variable)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation that the transaction program is deallocating.

TYPE specifies the type of deallocation APPC/PC is to perform.

- **SYNC_LEVEL** directs APPC/PC to deallocate the conversation based on one of the following synchronization levels allocated to this conversation:

- If the SYNC_LEVEL is NONE, APPC/PC performs the function of the FLUSH verb and then deallocates the conversation normally.

- If the SYNC_LEVEL is CONFIRM, APPC/PC performs the function of the CONFIRM verb and, if it is successful (as indicated by a return code of OK on this DEALLOCATE verb), APPC/PC deallocates the conversation normally. If it is unsuccessful, the return code determines the state of the conversation.

- **FLUSH** directs APPC/PC to perform the function of the FLUSH verb, and then to deallocate the conversation normally.

- **ABEND_PROG**, **ABEND_SVC**, or **ABEND_TIMER** directs APPC/PC to perform the function of the FLUSH verb when the conversation is in send state, and then to deallocate the conversation abnormally. APPC/PC can perform logical record truncation if the program deallocates a conversation in send state. If the program deallocates a conversation in receive state, APPC/PC can perform data purging.

For types SYNC_LEVEL and FLUSH, APPC/PC informs the remote transaction program of the deallocation with a DEALLOCATE_NORMAL return code. For type ABEND_PROG, ABEND_SVC, or ABEND_TIMER, APPC/PC informs the remote transaction program of the deallocation with a DEALLOCATE_ABEND_PROG, DEALLOCATE_ABEND_SVC, or DEALLOCATE_ABEND_TIMER return code (respectively), except in the following case: if the remote transaction program has issued a SEND_ERROR, it may receive a DEALLOCATE_NORMAL return code instead of one of the DEALLOCATE_ABEND return codes.

If APPC/PC performs the FLUSH or the CONFIRM function as part of the DEALLOCATE verb, APPC/PC also flushes the send buffer of the local LU.

LOG_DATA_LENGTH specifies the number of bytes of log data to be sent. Set this parameter to 0 if there is no log data. The parameter must be 0 for any TYPE value except ABEND_PROG, ABEND_SVC or ABEND_TIMER. Values between 0 and 65535 are valid for this parameter, but the sum of this value and the offset of the LOG_DATA address must not exceed 65535. This limit prevents the log data from crossing a segment boundary.

LOG_DATA specifies the variable (consisting of a single LL) containing transaction program error information that APPC/PC is to send to the partner LU. The partner LU either discards the data or saves it in a system error log. (For a description of how APPC/PC passes the LOG_DATA it receives to the application subsystem, see "Managing Logged Errors (SYSLOG Exit)" on page 2-10.) The transaction program must format this error information as an error log GDS variable (for the correct format, see "GDS Variables" in the *SNA Reference Summary*).

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The TYPE parameter determines which of the following DEALLOCATE return codes APPC/PC can return to the program.

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.
  - DATA_AREA_ACROSS_SEGMENT: The log data area crosses a segment boundary.
  - DEALLOCATE_BAD_TYPE: APPC/PC does not recognize the specified TYPE.

- LOG_LL_WRONG: The program specified an incorrect length for the log data.

- STATE_CHECK

  - DEALLOC_FLUSH_BAD_STATE: The program specified the TYPE(SYNC_LEVEL) parameter for a conversation specified with SYNC_LEVEL(NONE) and the conversation is not in send state. Alternatively, the program may have specified TYPE(FLUSH) when the conversation was not in send state.

  - DEALLOC_CONFIRM_BAD_STATE: The program specified TYPE(SYNC_LEVEL) for a conversation specified with SYNC_LEVEL(CONFIRM) when the conversation was not in send state.

For detailed information on the following return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- OK.

If the program specifies TYPE(SYNC_LEVEL) and the synchronization level allocated to this conversation is CONFIRM, APPC/PC can report one of the following return codes: (For detailed information on these return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes.")

- ALLOCATION_ERROR
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DEALLOCATE_ABEND_PROG
- DEALLOCATE_ABEND_SVC

- DEALLOCATE_ABEND_TIMER
- PROG_ERROR_PURGING
- SVC_ERROR_PURGING.

State Changes:

The conversation enters reset state when the RETURN_CODE indicates OK. For information on state changes when the RETURN_CODE indicates other than OK, see "Understanding Basic Conversation Return Codes" on page 7-5.

Notes:

1. A transaction program can use the TYPE(SYNC_LEVEL) parameter to deallocate the conversation based on the synchronization level allocated to the conversation.

   - If the synchronization level is NONE, APPC/PC deallocates the conversation unconditionally.

   - If the synchronization level is CONFIRM, APPC/PC deallocates the conversation if the remote transaction program issues CONFIRMED to respond to the confirmation request. The conversation remains allocated if the remote transaction program issues SEND_ERROR to respond to the confirmation request.

2. A transaction program can use the TYPE(FLUSH) parameter to deallocate the conversation unconditionally, regardless of its synchronization level.

3. A transaction program can use the TYPE(ABEND_PROG), TYPE(ABEND_SVC), and TYPE(ABEND_TIMER) parameters to deallocate the conversation unconditionally, regardless of its synchronization level and its current state. Specifically:

   - The TYPE(ABEND_PROG) parameter is for use by a transaction program when it detects an error

condition that prevents further useful communication; that is, communications that would lead to successful completion of the transaction. The specific use and meaning of ABEND_PROG are program-defined.

- The TYPE(ABEND_SVC) parameter is for use by an LU services component, such as one that processes mapped conversation verbs, when it detects an error condition caused by its peer LU services component in the remote LU. For example, the LU services component issues the DEALLOCATE verb with this parameter when it detects a format error in control information sent by the peer LU services component.

- The TYPE(ABEND_TIMER) parameter is for use by an LU services component, such as one that processes mapped conversation verbs, when it detects or is informed of a condition that requires the conversation to be deallocated without further communication. For example, the LU services component issues the DEALLOCATE verb with this parameter if too much time elapses without receiving any information, or an operator prematurely ends program execution.

4. DEALLOCATE with TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER) resets or cancels posting. If posting is active and APPC/PC has posted the conversation, it resets the posting. If posting is active and APPC/PC has not posted the conversation, APPC/PC cancels posting so that no more posting occurs. For more details about posting, see the description of the POST_ON_RECEIPT verb.

FLUSH

Flushes the send buffer of the local LU by sending all buffered information to the remote LU. Information buffered by the LU can come from ALLOCATE, DEALLOCATE, SEND_DATA, PREPARE_TO_RECEIVE, and SEND_ERROR. Refer to the descriptions of these verbs for more details on this buffered information, including when APPC/PC places information in the buffer.

FLUSH	Supplied Parameters: TP_ID (variable) CONV_ID (variable)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation for which the local LU is to flush its send buffer.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The FLUSH return codes are as follows:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.

- BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

- STATE_CHECK

  - FLUSH_NOT_SEND_STATE: The conversation is not in send state.

For detailed information on the following return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- OK.

State Changes:

None

Notes:

1. A transaction program can use this verb to optimize the processing between the local and remote transaction programs. The LU normally buffers the data from consecutive SEND_DATA verbs until it has a sufficient amount for transmission. At that time it transmits the buffered data. However, the local transaction program can issue FLUSH in order to force the LU to send the buffered data, minimizing the delay in the remote transaction programs processing of the data.

2. The LU flushes its send buffer only when it has information to transmit; otherwise, the FLUSH verb has no effect.

GET_ATTRIBUTES

Returns the attributes of the specified conversation.

GET_ATTRIBUTES	Supplied Parameters: TP_ID (variable) CONV_ID (variable)
	Returned Parameters: RETURN_CODE (variable) LU_ID (variable) OWN_NET_NAME (variable) OWN_LU_NAME (variable) PARTNER_LU_NAME (variable) PARTNER_FULLY_QUALIFIED_LU_NAME (variable) MODE_NAME (variable) SYNC_LEVEL (variable) USER_ID (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation for which the attributes are desired.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The GET_ATTRIBUTES return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

For detailed information on the following return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- OK.

LU_ID indicates the identifier of the LU at which the local transaction program is located. APPC/PC passes this value to the application subsystem when the application subsystem defines the LU by issuing the ATTACH_LU verb. (For more information on the LU_ID parameter, see "ATTACH_LU" on page 5-7.)

OWN_NET_NAME indicates the name (in EBCDIC) of the network containing the LU at which the local transaction program is located.

OWN_LU_NAME indicates the name (in EBCDIC) of the LU at which the local transaction program is located.

PARTNER_LU_NAME indicates the name (in EBCDIC) of the LU at which the remote transaction program is located. The local LU uses this name to identify the remote LU when allocating a conversation. (For more information on the PARTNER_LU_NAME, see the description of the LU_NAME parameter of the ALLOCATE verb.)

PARTNER_FULLY_QUALIFIED_LU_NAME indicates the fully-qualified name (in EBCDIC) of the LU at which the remote transaction program is located. If APPC/PC does not know the partner's fully-qualified LU name, it returns a null (0-length) value.

MODE_NAME indicates the mode name (in EBCDIC) for the session on which APPC/PC allocated the conversation.

SYNC_LEVEL indicates the level of synchronization processing specified for the conversation. For information on a conversation's level of synchronization, see the description of the SYNC_LEVEL parameter of the ALLOCATE verb. The synchronization levels are:

- NONE
- CONFIRM.

USER_ID indicates the user ID (in EBCDIC) if the program specified conversation-level security. An incoming ALLOCATE specifies this USER_ID when it requests the application subsystem to start a transaction program. If the program did not specify conversation-level security, or if the transaction program was initiated locally, APPC/PC returns a null (0-length) value for the USER_ID parameter.

State Changes:

None

Note:

The following parameters are returned in EBCDIC:

- OWN_NET_ID
- OWN_LU_NAME
- PARTNER_LU_NAME
- PARTNER_LU_FULLY_QUALIFIED_NAME
- MODE_NAME
- USER_ID.

You can use the CONVERT verb to convert these parameter values from ASCII to EBCDIC.

GET_TYPE

Returns the conversation type (basic or mapped) of the specified conversation. A transaction program can issue this verb for both basic and mapped conversations.

GET_TYPE	Supplied Parameters: TP_ID (variable) CONV_ID (variable)
	Returned Parameters: RETURN_CODE (variable) TYPE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation for which the conversation type is desired.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The GET_TYPE return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

For detailed information on the following return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- OK.

TYPE indicates the conversation type. The conversation types are:

- BASIC_CONVERSATION indicates that a basic conversation was initiated by one of the following methods:

  - The conversation was initiated with an ALLOCATE(CONV_TYPE=BASIC_CONVERSATION) verb.

  - An incoming ALLOCATE specified a basic conversation.

- MAPPED_CONVERSATION indicates that a mapped conversation was initiated by one of the following methods:

  - The conversation was initiated with an MC_ALLOCATE verb.

  - The conversation was initiated with an ALLOCATE(CONV_TYPE=MAPPED_CONVERSATION) verb.

  - An incoming ALLOCATE specified a mapped conversation.

State Changes:

None

Note:

A program that APPC/PC can process at either the basic conversation API or the mapped conversation API uses this verb to determine which category of verbs, basic conversation or mapped conversation, to issue.

POST_ON_RECEIPT

Causes the LU to post the specified conversation when information is available for the transaction program to receive. The information can be data, conversation status, or a request for confirmation. If the transaction program must wait for posting to occur, it should issue the WAIT verb after POST_ON_RECEIPT. Alternatively, the transaction program can determine whether posting has occurred by issuing the TEST verb after POST_ON_RECEIPT.

POST_ON_RECEIPT	Supplied Parameters: TP_ID (variable) CONV_ID (variable) FILL (BUFFER) (LL) MAX_LENGTH (variable)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation the LU is to post when it receives information for the local transaction program.

FILL specifies when the LU should post the conversation.

- **BUFFER** directs the LU to post the conversation when the amount of available data is at least equal to that specified by the MAX_LENGTH parameter, or when the end of the data is available, whichever

occurs first. The amount of data required for posting is independent of the logical record format of the data.

- **LL** directs the LU to post the conversation when it receives a complete or truncated logical record, or when it receives a part of a logical record that is at least equal in length to that specified by the MAX_LENGTH parameter, whichever occurs first.

The FILL(BUFFER) and FILL(LL) parameters for POST_ON_RECEIPT apply only at the time the transaction program issues the verb. They do not affect later uses of the POST_ON_RECEIPT verb or the other verbs that use these parameters (RECEIVE_AND_WAIT and RECEIVE_IMMEDIATE).

Posting also occurs independent of the FILL specification when the LU receives information other than data, such as conversation status (SEND, PROG_ERROR_TRUNC, or DEALLOCATE_NORMAL indications, for example), or a confirmation request.

MAX_LENGTH specifies the maximum length of received data that causes posting to occur. The transaction program should set this value equal to the maximum amount of data that it is prepared to receive. The LU uses the MAX_LENGTH and FILL parameters to determine when to post the conversation for the receipt of data. Values between 0 and 65535 bytes are valid for this parameter.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The POST_ON_RECEIPT return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

- INVALID_LENGTH: The program specified an illegal value for the MAX_LENGTH parameter.

- P_ON_R_BAD_FILL: The program specified an illegal value for the FILL parameter.

- STATE_CHECK

  - P_ON_R_NOT_RCV_STATE: The conversation is not in receive state.

For detailed information on the following return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- OK.

State Changes:

None

Notes:

1. A transaction program can use this verb with the TEST and WAIT verbs. The program can use POST_ON_RECEIPT followed by WAIT to synchronize the receipt of data from multiple conversations. To prepare to receive data from multiple conversations, the program issues POST_ON_RECEIPT for each of the conversations and then issues WAIT for all of the conversations. The program then waits for the LU to post one of the conversations as soon as the LU receives the specified amount of data from its respective partner LUs.

   The transaction program can periodically use POST_ON_RECEIPT followed by TEST to continue processing until the test indicates that the local LU data is available.

2. Posting occurs when the LU has the specified amount of information that the transaction program can receive. This information can include conversation status, or a request for confirmation, as well as data from the partner transaction program. For a description of the different types of information a program can receive, see the description of the RECEIVE_AND_WAIT verb.

3. Posting is active for a conversation after the transaction program issues POST_ON_RECEIPT for the conversation and before it issues a verb that resets or cancels posting for that conversation.

   A transaction program can reset posting by issuing any of the following verbs *after* the LU posts the conversation:

   - DEALLOCATE with TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER)
   - RECEIVE_AND_WAIT
   - RECEIVE_IMMEDIATE
   - SEND_ERROR
   - TEST
   - WAIT.

   A transaction program can cancel posting by issuing any of the following verbs *before* the LU posts the conversation:

   - DEALLOCATE with TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER)
   - RECEIVE_IMMEDIATE
   - SEND_ERROR.

   The transaction program can reactivate posting by issuing another POST_ON_RECEIPT verb.

4. A transaction program can issue any number of POST_ON_RECEIPTs for a given conversation before it resets or cancels posting. The last POST_ON_RECEIPT that a program issues for a conversation determines when the LU posts the conversation.

   For example, a program can specify posting parameters by issuing POST_ON_RECEIPT with FILL(BUFFER) and MAX_LENGTH(1000) in preparation to receive 1000 bytes of data. If the program then issues the verb again with MAX_LENGTH(500), the LU posts the conversation when 500 bytes of data are available. Alternatively, if the program issues POST_ON_RECEIPT again with FILL(LL), the LU posts the conversation in terms of logical records.

5. POST_ON_RECEIPT with MAX_LENGTH(0) or MAX_LENGTH(1) directs the LU to post a conversation upon receipt of any number of bytes of data.

6. The FILL(BUFFER) parameter directs the LU to post data according to the number of data bytes it receives, regardless of the logical record format. The LU posts the conversation when the number of data bytes in its receive buffer is equal to, or less than, the number specified with the MAX_LENGTH parameter. The LU posts the conversation for less than the MAX_LENGTH amount of data only when it receives the end of the data. The state of the conversation changes to send, confirm, or reset when the local LU receives the end of the data.

   For more information on the change in state, see the description of the RECEIVE_AND_WAIT verb.

7. APPC/PC stores incoming data in its buffers during a POST operation. If the POST_ON_RECEIPT verb specifies a large MAX_LENGTH value, the received data could possibly exhaust the storage reserved for these buffers before the program accepted the data from the LU. If receive pacing is not specified, APPC/PC does not control the sending of data from

the partner transaction program and APPC/PC abnormally terminates if its workspace buffers become filled. If you use receive pacing and issue a WAIT or TEST for more than R(P-1) bytes of data (where P = pacing window, R = negotiated maximum RU size), APPC/PC may report a buffer full condition with a return code of DATA_POSTING_BLOCKED. When the buffers are full, you can choose to receive the data. If you want the buffers to accumulate more than this amount of data before you receive the data, reconfigure APPC/PC to increase the pacing window size or the MAX_RU_SIZE. Alternatively, you can choose not to use pacing.

PREPARE_TO_RECEIVE

Changes a basic conversation from send to receive state in preparation to receive data. Also performs the function of the FLUSH or CONFIRM verbs.

PREPARE_TO_RECEIVE	Supplied Parameters: TP_ID (variable) CONV_ID (variable) TYPE (SYNC_LEVEL) (FLUSH) LOCKS (SHORT) (LONG)
	Returned Parameters: RETURN_CODE (variable)

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which the local transaction program is preparing to receive data.

TYPE specifies the type of prepare-to-receive to be performed for this conversation.

- **SYNC_LEVEL** directs APPC/PC to perform the prepare-to-receive based on the synchronization level allocated to this conversation:
  - If the SYNC_LEVEL is NONE, PREPARE_TO_RECEIVE performs the function of the FLUSH verb and then places the conversation in receive state.

- If the SYNC_LEVEL is CONFIRM, PREPARE_TO_RECEIVE performs the function of the CONFIRM verb (which includes the function of the FLUSH verb) and then, if the verb is successful, places the conversation in receive state. If the action of the CONFIRM verb is not successful (as indicated by a return code other than OK on the PREPARE_TO_RECEIVE verb), the return code determines the state of the conversation.

- **FLUSH** directs APPC/PC to perform the function of the FLUSH verb and then to place the conversation in receive state.

LOCKS specifies when APPC/PC is to return control to the local transaction program after it performs the CONFIRM function of this verb. APPC/PC ignores this parameter unless TYPE(SYNC_LEVEL) is also specified and the synchronization level for this conversation is CONFIRM.

- **SHORT** directs APPC/PC to return control as soon as the LU receives a CONFIRMED reply from the partner transaction program. The SHORT option returns control to the transaction program more quickly than the LONG option but it causes more information to be sent to the partner.

- **LONG** directs APPC/PC to return control after the LU receives information, such as data, from the remote transaction program following a CONFIRMED reply. The LONG option causes less information to be sent to the partner than the SHORT option but it requires more time.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The TYPE parameter determines which of the following return codes can be returned to the program:

- PARAMETER_CHECK

- BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.

- BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

- P_TO_R_INVALID_TYPE: APPC/PC does not recognize the specified TYPE.

- STATE_CHECK

  - UNFINISHED_LL: The conversation is in send state, and the program started, but did not finish, sending a logical record.

  - P_TO_R_NOT_SEND_STATE: The conversation is not in send state.

For detailed information on the following four return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- OK.

If the program specifies TYPE(SYNC_LEVEL) and the synchronization level allocated to this conversation is CONFIRM, APPC/PC can report one of the following return codes. For detailed information on these return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- ALLOCATION_ERROR
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DEALLOCATE_ABEND_PROG
- DEALLOCATE_ABEND_SVC

- DEALLOCATE_ABEND_TIMER
- PROG_ERROR_PURGING
- SVC_ERROR_PURGING.

State Changes:

The conversation enters receive state when the RETURN_CODE indicates OK. For information on state changes when the RETURN_CODE indicates other than OK, see "Understanding Basic Conversation Return Codes" on page 7-5.

Notes:

1. The conversation for the remote transaction program enters send state when the remote transaction program issues a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb and receives the SEND indication from the local transaction program on the WHAT_RECEIVED parameter. The remote transaction program can then send data to the local transaction program.

2. If the local transaction program issues PREPARE_TO_RECEIVE with a SYNC_LEVEL of CONFIRM, the remote transaction program enters send state after issuing CONFIRMED.

RECEIVE_AND_WAIT

Waits for information to arrive on the specified conversation and then receives the information. If information is already available, the transaction program receives it without waiting. The information can be data, conversation status, or a request for confirmation. APPC/PC returns control to the transaction program and indicates the type of information.

The transaction program can issue this verb when the conversation is in either send or receive state. If the conversation is in send state, the LU flushes its send buffer, sending all buffered information and the SEND indication to the remote transaction program. The SEND indication places the conversation in receive state. The LU then waits for information to arrive from the remote transaction program. The remote transaction program sends data to the local transaction program after it receives the SEND indication.

RECEIVE_AND_WAIT	Supplied Parameters: TP_ID (variable) CONV_ID (variable) FILL (BUFFER) (LL) DATA_PTR (variable) MAX_LENGTH (variable)
	Returned Parameters: RETURN_CODE (variable) DATA_LENGTH (variable) DATA (see DATA_PTR) WHAT_RECEIVED (variable) REQUEST_TO_SEND_RECEIVED (YES) (NO)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which the local transaction program is waiting to receive information.

FILL specifies whether the transaction program should receive data according to the logical record format of the data or according to the number of bytes received.

- **BUFFER** directs the LU to release the data in its receive buffer when the amount of available data is at least equal to that specified by the MAX_LENGTH parameter, or when the end of data is available, whichever occurs first. The amount of data required for receipt by the transaction program is independent of its logical record format. The amount of data received is equal to, or less than, the number of data bytes specified on the MAX_LENGTH parameter. The amount can be less than the specified length only at the end of the data.

- **LL** directs the LU to release the data in its receive buffer when it receives a complete or truncated logical record, or a part of a logical record that is at least equal in length to that specified by the MAX_LENGTH parameter, whichever occurs first.

The FILL(BUFFER) and FILL(LL) parameters for RECEIVE_AND_WAIT apply only at the time the transaction program issues the verb. They do not affect later uses of the RECEIVE_AND_WAIT verb or the other verbs that use these parameters (POST_ON_RECEIPT and RECEIVE_IMMEDIATE).

DATA_PTR specifies the address of the buffer which is to contain the received data.

MAX_LENGTH specifies the maximum amount of data (in bytes) that the transaction program is to receive. Values between 0 and 65535 are valid for this parameter, but the sum of this value and the offset portion of DATA_PTR must not exceed 65535. This limit keeps the incoming data from crossing a segment boundary.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The RECEIVE_AND_WAIT return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.
  - DATA_AREA_ACROSS_SEGMENT: The receive data area crosses a segment boundary.
  - RCV_AND_WAIT_BAD_FILL: The transaction program specified an illegal value for the FILL parameter.
- STATE_CHECK
  - RCV_AND_WAIT_BAD_STATE: The conversation is not in send or receive state.
  - RCV_AND_WAIT_NOT_LL_BDY: The conversation is in send state, and the program started, but did not finish, sending a logical record.

For detailed information on the following return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- ALLOCATION_ERROR

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DEALLOCATE_ABEND_PROG
- DEALLOCATE_ABEND_SVC
- DEALLOCATE_ABEND_TIMER
- DEALLOCATE_NORMAL
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- OK
- PROG_ERROR_NO_TRUNC
- PROG_ERROR_PURGING
- PROG_ERROR_TRUNC
- SVC_ERROR_NO_TRUNC
- SVC_ERROR_PURGING
- SVC_ERROR_TRUNC.

DATA_LENGTH indicates the actual amount of data the transaction program received up to the maximum (MAX_LENGTH). If the program receives information other than data the value of this variable is 0.

DATA from the partner transaction program is received in the buffer specified by the address in the DATA_PTR parameter. APPC/PC does not place any information in this buffer when the program receives information other than data, as indicated by the WHAT_RECEIVED parameter.

WHAT_RECEIVED indicates what the transaction program received. The program should examine this variable only when RETURN_CODE indicates OK. APPC/PC does not place any information in this variable when RETURN_CODE indicates other than OK.

- DATA indicates that the transaction program received a MAX_LENGTH amount of data or the end of data (independent of its logical record format) after specifying FILL(BUFFER).

- DATA_COMPLETE indicates that the transaction program received a complete logical record or the last portion of a record after specifying FILL(LL).

- DATA_INCOMPLETE indicates that the transaction program received less than a complete logical record after specifying FILL(LL). The program can use another RECEIVE_AND_WAIT (or possibly more than one) to receive the rest of the data for the incomplete logical record.

- CONFIRM indicates that the remote transaction program has issued CONFIRM, requesting the local transaction program to respond by issuing CONFIRMED. The program can respond, instead, by issuing SEND_ERROR.

- CONFIRM_DEALLOCATE indicates that the remote transaction program has issued DEALLOCATE with TYPE(SYNC_LEVEL), and the synchronization level is CONFIRM. The local transaction program can respond by issuing CONFIRMED or SEND_ERROR.

- CONFIRM_SEND indicates that the remote transaction program has issued PREPARE_TO_RECEIVE with TYPE(SYNC_LEVEL), and the synchronization level is CONFIRM. The local transaction program can respond by issuing CONFIRMED, or by issuing another verb such as SEND_ERROR.

- SEND indicates that the remote transaction program has entered receive state, placing the local transaction program in send state. This indication signals that the local transaction program can issue the SEND_DATA verb to send data to the remote transaction program.

REQUEST_TO_SEND_RECEIVED indicates whether the local LU has received a REQUEST_TO_SEND notification. The indication is either YES or NO.

- YES indicates that the local LU has received a REQUEST_TO_SEND notification from the remote transaction program. The remote transaction program

has issued REQUEST_TO_SEND, requesting the local transaction program to enter receive state and place the remote transaction program in send state.

- NO indicates that the local LU has not received a REQUEST_TO_SEND notification from the remote transaction program.

State Changes:

- A change in state occurs only if RETURN_CODE indicates OK. For information on state changes when RETURN_CODE indicates other than OK, see "Understanding Basic Conversation Return Codes" on page 7-5.

- The conversation enters receive state when the transaction program issues RECEIVE_AND_WAIT in send state and WHAT_RECEIVED indicates DATA, DATA_COMPLETE, or DATA_INCOMPLETE.

- The conversation enters send state when WHAT_RECEIVED indicates SEND.

- The conversation enters confirm state when WHAT_RECEIVED indicates CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE.

- No state change occurs when transaction program issues RECEIVE_AND_WAIT in receive state and WHAT_RECEIVED indicates DATA, DATA_COMPLETE, or DATA_INCOMPLETE.

Notes:

1. When the transaction program issues RECEIVE_AND_WAIT in send state, the LU implicitly executes a PREPARE_TO_RECEIVE with TYPE(FLUSH) before it executes the RECEIVE_AND_WAIT. See the description of the PREPARE_TO_RECEIVE verb for details of its function.

2. When the transaction program specifies FILL(LL) to receive data in terms of logical records, the WHAT_RECEIVED parameter can return either the DATA_COMPLETE or the DATA_INCOMPLETE indication depending on the data received. The two sequences of events leading to these indications are:

   - The transaction program receives a complete logical record or the last portion of a record. The length of the record or portion of the record is equal to or less than the length specified on the MAX_LENGTH parameter. The WHAT_RECEIVED parameter indicates DATA_COMPLETE.

   - The transaction program receives an incomplete logical record. The logical record is incomplete because:

     - The logical record contains more data than the program specified in the MAX_LENGTH parameter, so the program receives a portion of the logical record equal to the specified length.

     - Only a portion of the logical record is available because the remote transaction program truncated it. The length of the truncated portion is less than, or equal to, the length the program specifies in the MAX_LENGTH parameter.

     In these two cases, the WHAT_RECEIVED parameter indicates DATA_INCOMPLETE. The transaction program issues another RECEIVE_AND_WAIT (or more than one) to receive the remainder of the logical record.

   For a definition of complete and incomplete logical records, see the description of the SEND_DATA verb.

3. The transaction program specifies FILL(BUFFER) to receive data independent of its logical record format. This specification indicates that the program is to receive an amount of data less than, or equal to, the

length it specifies in the MAX_LENGTH parameter. The local transaction program can receive less than the MAX_LENGTH amount of data only at the end of the data sent from the remote transaction program. The end of data occurs when the conversation changes to send, confirm, or reset state after the program receives the data. When the transaction program specifies FILL(BUFFER), it must perform its own tracking of the data's logical record format.

4. A transaction program can use RECEIVE_AND_WAIT with MAX_LENGTH(0) to determine the type of information available without actually receiving any information. The RETURN_CODE and WHAT_RECEIVED parameters indicate the type of information available as usual. If data is available and the program specified FILL(LL), the WHAT_RECEIVED parameter indicates DATA_INCOMPLETE. If data is available and the program specified FILL(BUFFER), the WHAT_RECEIVED parameter indicates DATA. In either case, however, the program receives no data.

5. The transaction program receives only one kind of information at a time. For example, it may receive data or a CONFIRM request, but it cannot receive both at the same time. Also, if the remote transaction program truncates a logical record, the local transaction program receives an indication of the truncation from the RECEIVE_AND_WAIT verb it issues after receiving the truncated record.

   The RETURN_CODE and WHAT_RECEIVED parameters indicate the kind of information the program receives.

6. RECEIVE_AND_WAIT performs the same posting as the POST_ON_RECEIPT verb. If posting is already active when the local transaction program issues the RECEIVE_AND_WAIT verb, the parameter values of this verb supersede those specified with the previous POST_ON_RECEIPT verb. APPC/PC resets posting when it finishes executing RECEIVE_AND_WAIT.

For more information on posting, see the description of the POST_ON_RECEIPT verb.

7. The local transaction program usually receives a REQUEST_TO_SEND notification when it is in send state. APPC/PC reports the REQUEST_TO_SEND notification to the program with a SEND_DATA verb or with a SEND_ERROR verb it issues in send state. However, the program can receive the REQUEST_TO_SEND notification when its conversation is in receive state under the following conditions:

   - When the local transaction program has just entered receive state and the remote transaction program issues REQUEST_TO_SEND before it enters send state.

   - When the remote transaction program has just entered receive state by issuing the PREPARE_TO_RECEIVE verb (not RECEIVE_AND_WAIT), and then issues REQUEST_TO_SEND before the local transaction program responds by entering send state. This ambiguity can occur because REQUEST_TO_SEND is an expedited request and this expedited request can arrive ahead of the request carrying the SEND indication.

     The local transaction program might not be able to distinguish this condition from the preceding case. The remote transaction program can avoid this ambiguity by waiting until it receives information from the local transaction program before it issues the REQUEST_TO_SEND verb.

   - When the remote transaction program issues the REQUEST_TO_SEND in send state.

RECEIVE_IMMEDIATE

Receives information that is available from the specified conversation, but does not wait for information to arrive. The information (if any) can be data, conversation status, or a request for confirmation. APPC/PC returns control to the transaction program with an indication of whether information was received and, if so, the type of information.

RECEIVE_IMMEDIATE	Supplied Parameters: TP_ID (variable) CONV_ID (variable) FILL (BUFFER) (LL) DATA_PTR (variable) MAX_LENGTH (variable)
	Returned Parameters: RETURN_CODE (variable) DATA_LENGTH (variable) DATA (see DATA_PTR) WHAT_RECEIVED (variable) REQUEST_TO_SEND_RECEIVED (YES) (NO)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which the local transaction program is to receive information.

FILL specifies whether the program should receive data according to the logical record format of the data or according to the number of bytes received.

- **BUFFER** directs the LU to release the data in its receive buffer when the amount of available data is at least equal to that specified by the MAX_LENGTH parameter, or when the end of data is available, whichever occurs first. The amount of data required for receipt by the transaction program is independent of its logical record format. The amount of data received is equal to, or less than, the number of data bytes specified on the MAX_LENGTH parameter. The amount can be less than the specified length only at the end of the data.

- **LL** directs the LU to release the data from its receive buffer when it receives a complete or truncated logical record, or part of a logical record that is at least as long as the logical record specified by the MAX_LENGTH parameter, whichever occurs first.

The FILL(BUFFER) and FILL(LL) parameters for RECEIVE_IMMEDIATE apply only at the time the transaction program issues the verb. They do not affect later uses of the RECEIVE_IMMEDIATE verb or the other verbs that use these parameters (POST_ON_RECEIPT and RECEIVE_AND_WAIT).

DATA_PTR specifies the address of the buffer that is to contain the received data.

MAX_LENGTH specifies the maximum amount of data (in bytes) that the transaction program is to receive. Values between 0 and 65535 are valid for this parameter, but the sum of this value and the offset portion of DATA_PTR must not exceed 65535. This limit prevents the incoming data from crossing a segment boundary.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The RECEIVE_AND_WAIT return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.
  - DATA_AREA_ACROSS_SEGMENT: The receive data area crosses a segment boundary.
  - RCV_IMMD_BAD_FILL: The program specified an illegal value for the FILL parameter.
- STATE_CHECK
  - RCV_IMMD_NOT_RCV_STATE: The conversation is not in receive state.
- UNSUCCESSFUL: The local LU has nothing for the program to receive.

For detailed information on the following return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- ALLOCATION_ERROR
- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DEALLOCATE_ABEND_PROG
- DEALLOCATE_ABEND_SVC
- DEALLOCATE_ABEND_TIMER
- DEALLOCATE_NORMAL
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- OK
- PROG_ERROR_NO_TRUNC
- PROG_ERROR_PURGING
- PROG_ERROR_TRUNC
- SVC_ERROR_NO_TRUNC

- SVC_ERROR_PURGING
- SVC_ERROR_TRUNC.

DATA_LENGTH indicates the actual amount of data the local transaction program received up to the maximum (MAX_LENGTH). If the program receives information other than data, or no information at all, the value of this variable is 0.

DATA from the partner transaction program is received in the buffer specified by the address in the DATA_PTR parameter. APPC/PC does not place any information in this buffer when the program receives information other than data, as indicated by the WHAT_RECEIVED parameter.

WHAT_RECEIVED indicates what the transaction program received. The program should examine this variable only when RETURN_CODE indicates OK. APPC/PC does not place any information in this variable when RETURN_CODE indicates other than OK.

- DATA indicates that the transaction program received a MAX_LENGTH amount of data or the end of data (independent of its logical record format) after specifying FILL(BUFFER).

- DATA_COMPLETE indicates that the transaction program received a complete logical record or the last portion of a record after specifying FILL(LL).

- DATA_INCOMPLETE indicates that the transaction program received less than a complete logical record after specifying FILL(LL). The program can use one or more RECEIVE_IMMEDIATE or RECEIVE_AND_WAIT verbs to receive the rest of the data for the incomplete logical record.

- CONFIRM indicates that the remote transaction program has issued CONFIRM, requesting the local transaction program to respond by issuing CONFIRMED. The program can respond, instead, by issuing SEND_ERROR.

- CONFIRM_SEND indicates that the remote transaction program has issued PREPARE_TO_RECEIVE with TYPE(SYNC_LEVEL), and the synchronization level is CONFIRM. The local transaction program can respond by issuing CONFIRMED, or SEND_ERROR.

- CONFIRM_DEALLOCATE indicates that the remote transaction program has issued DEALLOCATE with TYPE(SYNC_LEVEL), and the synchronization level is CONFIRM. The local transaction program can respond by issuing either CONFIRMED or SEND_ERROR.

- SEND indicates that the remote transaction program has entered receive state, placing the local transaction program in send state. This indication signals that the local transaction program can issue the SEND_DATA verb to send data to the remote transaction program.

REQUEST_TO_SEND_RECEIVED indicates whether the local LU has received a REQUEST_TO_SEND notification. The indication is either YES or NO.

- YES indicates that the local LU received a REQUEST_TO_SEND notification from the remote transaction program. The remote transaction program has issued REQUEST_TO_SEND, requesting the local transaction program to enter receive state and place the remote transaction program in send state.

- NO indicates that the local LU has not received a REQUEST_TO_SEND notification from the remote transaction program.

State Changes:

The following state changes occur when the RETURN_CODE indicates OK. For information on state changes when RETURN_CODE indicates other than OK, see "Understanding Basic Conversation States" on page 7-2.

- The conversation enters send state when WHAT_RECEIVED indicates SEND.

- The conversation enters confirm state when WHAT_RECEIVED indicates CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE.

- No state change occurs when WHAT_RECEIVED indicates DATA, DATA_COMPLETE, or DATA_INCOMPLETE.

Notes:

1. When the transaction program specifies FILL(LL) to receive data in terms of logical records, the WHAT_RECEIVED parameter can return either the DATA_COMPLETE or the DATA_INCOMPLETE indication depending on the data received. The two sequences of events leading to these indications are:

   - The transaction program receives a complete logical record or the last portion of a record. The length of the record or portion of the record is equal to or less than the length specified on the MAX_LENGTH parameter. The WHAT_RECEIVED parameter indicates DATA_COMPLETE.

   - The transaction program receives an incomplete logical record. The logical record is incomplete because:

     – The logical record contains more data than the program specifies in the MAX_LENGTH parameter so the program receives a portion of the logical record equal to the specified length.

     – Only a portion of the logical record is available because the remote transaction program truncated it. The length of the truncated portion is less than, or equal to, the length the program specifies in the MAX_LENGTH parameter.

In these two cases, the WHAT_RECEIVED parameter indicates DATA_INCOMPLETE. The transaction program issues another RECEIVE_IMMEDIATE or RECEIVE_AND_WAIT (or more than one) to receive the remainder of the logical record.

For a definition of complete and incomplete logical records, see the description of the SEND_DATA verb.

2. The transaction program specifies FILL(BUFFER) to receive data independent of its logical record format. This specification directs the program to receive whatever data is available up to the MAX_LENGTH amount. When the program specifies FILL(BUFFER), it must perform its own tracking of the data's logical record format.

3. A program can use RECEIVE_IMMEDIATE with MAX_LENGTH(0) to determine the type of information available without actually receiving any information. The RETURN_CODE and WHAT_RECEIVED parameters indicate the type of information available as usual. If data is available and the program specified FILL(LL), the WHAT_RECEIVED parameter indicates DATA_INCOMPLETE. If data is available and the program specified FILL(BUFFER), the WHAT_RECEIVED parameter indicates DATA. In either case, however, the transaction program receives no data.

4. The transaction program receives only one kind of information at a time. For example, it may receive data or a CONFIRM request, but it cannot receive both at the same time. Also, if the remote transaction program truncates a logical record, the local transaction program receives the indication of the truncation from the RECEIVE_IMMEDIATE verb it issues after receiving the truncated record.

   The RETURN_CODE and WHAT_RECEIVED parameters indicate the kind of information the transaction program receives.

5. RECEIVE_IMMEDIATE resets or cancels posting. RECEIVE_IMMEDIATE resets posting if posting is active and the LU has posted the conversation. The verb cancels further posting if posting is active and the LU has not posted the conversation. For information about posting, see the description of the POST_ON_RECEIPT verb.

6. The local transaction program usually receives a REQUEST_TO_SEND notification when it is in send state. APPC/PC reports the REQUEST_TO_SEND notification to the program with a SEND_DATA verb or with a SEND_ERROR verb it issues in send state. However, the program can receive the REQUEST_TO_SEND notification when its conversation is in receive state under the following conditions:

   - When the local transaction program has just entered receive state and the remote transaction program issues REQUEST_TO_SEND before it enters send state.

   - When the remote transaction program has just entered receive state by issuing the PREPARE_TO_RECEIVE verb (not RECEIVE_IMMEDIATE), and then issuing REQUEST_TO_SEND before the local transaction program responds by entering send state. This ambiguity can occur because REQUEST_TO_SEND is an expedited request and this expedited request can arrive ahead of the request carrying the SEND indication.

     The local transaction program might not be able to distinguish this condition from the preceding case. The remote transaction program can avoid this ambiguity by waiting until it receives information from the local transaction program before it issues the REQUEST_TO_SEND.

   - When the remote transaction program issues the REQUEST_TO_SEND in send state.

REQUEST_TO_SEND

Notifies the remote transaction program that the local transaction program is requesting to enter send state. APPC/PC places the conversation in send state when the local transaction program subsequently receives a SEND indication from the remote transaction program.

REQUEST_TO_SEND	Supplied Parameters: TP_ID (variable) CONV_ID (variable)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the instance of the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation for which the local transaction program is requesting to enter send state.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The REQUEST_TO_SEND return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

- STATE_CHECK
  - R_T_S_NOT_RCV_STATE: The conversation is not in receive or confirm state.

For detailed information on the following return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- OK.

State Changes:

None

Notes:

1. A transaction program enters receive state by issuing the PREPARE_TO_RECEIVE verb or the RECEIVE_AND_WAIT verb. The remote transaction program enters send state after the local transaction program issues RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE, and after it receives the SEND indication on the WHAT_RECEIVED parameter.

2. APPC/PC normally returns the REQUEST_TO_SEND_RECEIVED indication of YES to the remote transaction program when the conversation is in send state; that is, on a SEND_DATA or SEND_ERROR verb issued in send state. However, APPC/PC can also return the REQUEST_TO_SEND indication of YES on a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb. See the description of these verbs for more information about the REQUEST_TO_SEND indication.

3. When the remote LU receives the REQUEST_TO_SEND notification, it retains the

notification until the remote transaction program issues a verb on which LU can indicate the notification with any verb that includes the REQUEST_TO_SEND_RECEIVED parameter.

The remote LU retains only one REQUEST_TO_SEND notification at a time (for each conversation). The remote LU discards additional notifications until the LU can indicate the retained notification to the remote transaction program. Therefore, the local transaction program can issue the REQUEST_TO_SEND verb to the remote transaction program more times than the remote LU indicates.

SEND_DATA

Sends one data record to the remote transaction program. The data format consists of logical records but the amount of data that this verb sends is independent of the data format.

SEND_DATA	Supplied Parameters: TP_ID (variable) CONV_ID (variable) DATA_LENGTH (variable) DATA (variable)
	Returned Parameters: RETURN_CODE (variable) REQUEST_TO_SEND_RECEIVED (YES) (NO)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which the local transaction program is sending the data.

DATA_LENGTH specifies the length of the data the local transaction program is sending. This data length has no relation to the length of a logical record. APPC/PC uses this information only to determine the length of the data at the location specified by the DATA parameter.

Values between 0 and 65535 are valid for this parameter, but the sum of this value and the offset portion of the

DATA address must not exceed 65535. This limit prevents the data from crossing a segment boundary.

If the transaction program specifies a data length value of 0, the local LU does not send any data. In this case, APPC/PC ignores the DATA parameter.

DATA specifies the variable containing the data record that the transaction program is sending. The data may include multiple records or part of a record. Each logical record consists of a 2-byte length field (denoted as LL) followed by a data field; the length of the data field can range from 0 to 32765 bytes. The 2-byte length field contains the 15-bit binary length of the record, and a high-order bit. The LU ignores this high-order bit when performing basic conversations (the LU's mapped conversation component uses this bit to support the mapped conversation verbs).

The length of the record includes the 2-byte length field; that is, a record is 2 bytes longer than the length of the data. For this reason, record length values of X'0000', X'0001', X'8000', and X'8001' are invalid.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The SEND_DATA return codes are:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.
  - DATA_AREA_ACROSS_SEGMENT: The data to be sent crosses a segment boundary.
  - BAD_LL: The DATA parameter contains an invalid logical record length (LL) value of X'0000', X'0001', X'8000', or X'8001'.

- STATE_CHECK
  - SEND_DATA_NOT_SEND_STATE: The conversation is not in send state.

For detailed information on the following return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- ALLOCATION_ERROR
- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DEALLOCATE_ABEND_PROG
- DEALLOCATE_ABEND_SVC
- DEALLOCATE_ABEND_TIMER
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- OK
- PROG_ERROR_PURGING
- SVC_ERROR_PURGING.

REQUEST_TO_SEND_RECEIVED indicates whether the local LU has received a REQUEST_TO_SEND notification. The indication is either YES or NO.

- YES indicates that the local LU has received a REQUEST_TO_SEND notification from the remote transaction program. The remote transaction program has issued REQUEST_TO_SEND, requesting the local transaction program to enter receive state and place the remote transaction program in send state.

- NO indicates that the local LU has not received a REQUEST_TO_SEND notification from the remote transaction program.

State Changes:

The state of the conversation does not change when RETURN_CODE indicates OK. For information on state

changes when RETURN_CODE indicates other than OK, see "Understanding Basic Conversation Return Codes" on page 7-5.

Notes:

1. The data sent by the transaction program consists of logical records. The DATA_LENGTH parameter of the SEND_DATA verb determines the amount of data a program sends when it issues this verb. This amount of data is independent of the length of the logical records. That is, the data may consist of one or more complete records, the beginning of a record, the middle of a record, or the end of a record. The following combinations of complete and partial records are also possible:

   - One or more complete records, followed by the beginning of a record
   - The end of a record, followed by one or more complete records
   - The end of a record, followed by one or more complete records, followed by the beginning of a record
   - The end of a record, followed by the beginning of a record.

2. A complete logical record contains the 2-byte LL field followed by the number of data bytes the program specifies in the LL field. (A value of 2 in the LL field specifies that the complete logical record contains only the 2-byte length field.)

   An incomplete logical record contains any amount of data less than a complete record. It can consist of only the first byte of the 2-byte LL field, the LL field plus all of the data field except the last byte, or any amount in between. A logical record remains incomplete until the program sends the last byte of the data field.

If the value of the LL field is 2 (indicating a data field of 0-length), the logical record is complete after the transaction program sends the second byte of the LL field.

3. The transaction program must finish sending a logical record before it can issue any of the following verbs:

   - CONFIRM
   - DEALLOCATE with TYPE(FLUSH)
   - DEALLOCATE with TYPE(SYNC_LEVEL)
   - PREPARE_TO_RECEIVE
   - RECEIVE_AND_WAIT.

   A transaction program finishes sending a logical record when it sends a complete record or when it truncates an incomplete record.

4. A program can truncate an incomplete logical record by issuing the SEND_ERROR verb. When the LU receives the SEND_ERROR verb, it flushes its buffer and sends whatever portion of a logical record it has accumulated before the program issues the SEND_ERROR verb. The LU then treats the first 2 bytes of data specified in the next SEND_DATA as the LL field for the next record.

   The program can also truncate an incomplete logical record by issuing DEALLOCATE with TYPE(ABEND_PROG), TYPE(ABEND_SVC), or TYPE(ABEND_TIMER).

5. The LU retains the data from the program until it accumulates (from one or more SEND_DATA verbs) a sufficient amount for transmission, or until the program issues a verb that forces the LU to flush its send buffer. The amount of data that is sufficient for transmission depends on the characteristics of the session allocated for the conversation. This amount varies from one session to another.

6. When the REQUEST_TO_SEND_RECEIVED parameter indicates YES, the remote transaction program is requesting that the local transaction

program enter receive state and place the remote transaction program in send state. A transaction program enters receive state by issuing the PREPARE_TO_RECEIVE verb or the RECEIVE_AND_WAIT verb. The partner transaction program enters send state after it issues the RECEIVE_AND_WAIT verb or the RECEIVE_IMMEDIATE verb, and receives the SEND indication from its partner transaction program on the WHAT_RECEIVED parameter.

SEND_ERROR

Informs the remote transaction program that the local transaction program detected an error. If the conversation is in send state, the LU flushes its send buffer.

After the successful completion of this verb, the local transaction program is in send state and the remote transaction program is in receive state. The transaction program must take the appropriate actions to correct the problem.

SEND_ERROR	Supplied Parameters: TP_ID (variable) CONV_ID (variable) TYPE (PROG) (SVC) LOG_DATA_LENGTH (variable) LOG_DATA (variable)
	Returned Parameters: RETURN_CODE (variable) REQUEST_TO_SEND_RECEIVED (YES) (NO)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which the local transaction program is notifying the remote transaction program of an error.

TYPE specifies the type of error being reported. A normal transaction program should use PROG. Programs that provide services to transaction programs should use SVC.

LOG_DATA_LENGTH specifies the number of bytes of log data to be sent. Set this parameter to 0 if there is no log data. The parameter must be 0 for any TYPE value except ABEND_PROG, ABEND_SVC or ABEND_TIMER. Values between 0 and 65535 are valid for this parameter, but the sum of this value and the offset of the LOG_DATA address must not exceed 65535. This limit prevents the log data from crossing a segment boundary.

LOG_DATA specifies the variable (consisting of a single LL) containing transaction program error information that APPC/PC is to send to the partner LU. The partner LU either discards the data or saves it in a system error log. (For a description of how APPC/PC passes LOG_DATA it receives to the application subsystem, see "Managing Logged Errors (SYSLOG Exit)" on page 2-10. The transaction program must format this error information as an error log GDS variable (for the correct format, see "GDS Variables" in the *SNA Reference Summary*).

Returned Parameters:

RETURN_CODE indicates the result of verb execution. APPC/PC can return the following return codes in any conversation state:

- PARAMETER_CHECK
  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.
  - BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.
  - DATA_AREA_ACROSS_SEGMENT: The log data area crosses a segment boundary.

- LOG_LL_WRONG: The LL field of the log data does not match the specified LOG_DATA_LENGTH.

- BAD_TYPE: APPC/PC does not recognize the specified TYPE.

For detailed information on the following return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- OK.

The following return codes can occur when the transaction program issues SEND_ERROR and the conversation is in send state. For detailed information on these return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- ALLOCATION_ERROR
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DEALLOCATE_ABEND_PROG
- DEALLOCATE_ABEND_SVC
- DEALLOCATE_ABEND_TIMER
- PROG_ERROR_PURGING
- SVC_ERROR_PURGING.

The following return codes can occur when the transaction program issues SEND_ERROR and the conversation is in receive state. For detailed information on these return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY

- DEALLOCATE_NORMAL.

The following return codes can occur when the transaction program issues SEND_ERROR and the conversation is in confirm state. For detailed information on these return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY.

REQUEST_TO_SEND_RECEIVED indicates whether the local LU has received a REQUEST_TO_SEND notification. The indication is either YES or NO.

- YES indicates that the local LU has received a REQUEST_TO_SEND notification from the remote transaction program. The remote transaction program has issued REQUEST_TO_SEND, requesting the local transaction program to enter receive state and place the remote transaction program in send state.

- NO indicates that the local LU has not received a REQUEST_TO_SEND notification.

State Changes:

The following state changes can occur if RETURN_CODE indicates OK. For information on state changes when RETURN_CODE indicates other than OK, see "Understanding Basic Conversation States" on page 7-2.

The conversation enters send state when the transaction program issues the verb in receive or confirm state. No state change occurs when the program issues SEND_ERROR in send state.

Notes:

1. The LU retains the error notification and log data in its send buffer until it accumulates enough information for transmission, or until the local transaction program issues a verb that causes the LU

to flush its send buffer. The amount of information that is sufficient for transmission depends on the characteristics of the session allocated for the conversation. This amount can vary from one session to another.

2. APPC/PC reports the SEND_ERROR to the remote transaction program as one of the following return codes (based on the TYPE parameter).

   For detailed information on these return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

   - PROG_ERROR_NO_TRUNC
   - PROG_ERROR_PURGING
   - PROG_ERROR_TRUNC
   - SVC_ERROR_NO_TRUNC
   - SVC_ERROR_PURGING
   - SVC_ERROR_TRUNC.

3. When the local transaction program issues SEND_ERROR in receive state, APPC/PC purges incoming information including return code indications. After purging these return code indications, APPC/PC reports others in their place.

   APPC/PC reports DEALLOCATE_NORMAL for the following purged return code indications.

   - ALLOCATION_ERROR
   - DEALLOCATE_ABEND_PROG
   - DEALLOCATE_ABEND_SVC
   - DEALLOCATE_ABEND_TIMER.

   APPC/PC reports OK for the following purged return code indications.

   - PROG_ERROR_PURGING
   - SVC_ERROR_PURGING.

   The other kinds of incoming information that APPC/PC can purge are:

- Data that a program sends by issuing the SEND_DATA verb

- Confirmation requests that a program sends by issuing the CONFIRM verb.

If the transaction program sends a confirmation request in conjunction with the DEALLOCATE verb by specifying the TYPE(SYNC_LEVEL) parameter, APPC/PC also purges the deallocation request.

APPC/PC does not purge an incoming REQUEST_TO_SEND indication. When the transaction program issues a verb that includes the REQUEST_TO_SEND_RECEIVED parameter, it receives the REQUEST_TO_SEND indication on this parameter.

4. The transaction program may use the SEND_ERROR verb for various application-level functions. For example, the program can issue this verb to truncate an incomplete logical record it is sending, to inform the remote program of an error it detected in data it received, or to reject a confirmation request.

5. The SEND_ERROR verb resets or cancels posting. This verb resets posting if posting is active and the LU has posted the conversation. The SEND_ERROR verb cancels posting if posting is active and the LU has not posted the conversation. Further posting will not occur. (For information on posting, see the description of the POST_ON_RECEIPT verb.)

6. LOG_DATA_LENGTH specifies the number of bytes of log data to be sent. Set this parameter to 0 if there is no log data. The parameter must be 0 for any TYPE value except ABEND_PROG, ABEND_SVC or ABEND_TIMER. Values between 0 and 65535 are valid for this parameter, but the sum of this value and the offset of the LOG_DATA address must not exceed 65535. This limit prevents the log data from crossing a segment boundary.

TEST

Tests the specified conversation for a condition. The return code indicates the result of the test.

TEST	Supplied Parameters: TP_ID (variable) CONV_ID (variable) TEST (POSTED) (REQUEST_TO_SEND_RECEIVED)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID specifies the ID of the conversation on which the local transaction program is performing the test.

TEST specifies the condition to be tested.

- **POSTED** tests whether the LU has posted the conversation. Before posting can occur, the program must have issued the POST_ON_RECEIPT verb. The return code indicates whether posting has occurred, as follows:

  - OK indicates that the LU has posted the conversation and that posting is now reset. The transaction program should issue RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE to receive the information waiting in the receive buffer of the local LU.

- UNSUCCESSFUL indicates that the conversation is in receive state but the LU has not posted the conversation.

- POSTING_NOT_ACTIVE indicates that posting is not active for the specified conversation.

- **REQUEST_TO_SEND_RECEIVED** tests whether the local LU has received a REQUEST_TO_SEND notification from the remote transaction program. The return code indicates the results of this test as follows:

  - OK indicates that the LU has received a REQUEST_TO_SEND indication. The remote transaction program has issued REQUEST_TO_SEND, requesting the local transaction program to enter receive state and place the remote transaction program in send state.

  - UNSUCCESSFUL indicates that the LU has not received a REQUEST_TO_SEND indication.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The TEST return codes other than those described above are:

- OK

  - POSTED_DATA: The LU has posted the conversation and data is available in the LU's receive buffer.

  - POSTED_NOT_DATA: The LU has posted the conversation and information other than data is available in the LU's receive buffer.

- PARAMETER_CHECK

  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.

- BAD_CONV_ID: APPC/PC does not recognize the specified CONV_ID.

- TEST_INVALID_TYPE: APPC/PC does not recognize the specified TYPE.

- STATE_CHECK

  - NOT_RCV_STATE: The conversation is not in receive state and the transaction program specified the POSTED option for the TYPE parameter.

- DATA_POSTING_BLOCKED: APPC/PC does not have sufficient storage space to permit posting one of the active conversations. This error does not cause APPC/PC to reset or cancel posting. The transaction program can continue issuing verbs, but to provide sufficient space for further posting it should issue RECEIVE_IMMEDIATE or RECEIVE_AND_WAIT to free some space. Alternatively, the program can issue DEALLOCATE_ABEND to terminate a conversation.

  If you use receive pacing and issue a TEST or WAIT for more than R(P-1) bytes of data (where P = the receive pacing window, and R = the negotiated maximum RU size for the session), APPC/PC may report a buffer full condition with this return code. Even this amount of data may be too large if the partner transaction program issues the FLUSH verb between SEND_DATA verbs, causing messages of less than the maximum RU size to flow on the line. Posting is blocked if the allotted session buffers are filled for the conversation being posted. If you need to test for more data than this amount, reconfigure APPC/PC and increase the pacing window size or do not use pacing. If you increase the pacing window size you must also increase the workspace size accordingly. However, if receive pacing is not used, APPC/PC will have no control of data sent from the partner transaction program. If its workspace becomes full APPC/PC abnormally terminates and rejects subsequent verbs.

For detailed information on the following return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- ALLOCATION_ERROR
- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DEALLOCATE_ABEND_PROG
- DEALLOCATE_ABEND_SVC
- DEALLOCATE_ABEND_TIMER
- DEALLOCATE_NORMAL
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- PROG_ERROR_NO_TRUNC
- PROG_ERROR_PURGING
- PROG_ERROR_TRUNC
- SVC_ERROR_NO_TRUNC
- SVC_ERROR_PURGING
- SVC_ERROR_TRUNC.

State Changes:

None

Notes:

1. The transaction program can use this verb in conjunction with POST_ON_RECEIPT. Using POST_ON_RECEIPT with this verb enables a transaction program to continue its processing while waiting for information to become available. The program can issue POST_ON_RECEIPT for one or more conversations and then periodically issue the TEST verb for each conversation to determine when information is available in the receive buffer of the local LU.

2. Posting is active for a conversation if the transaction program has issued the POST_ON_RECEIPT verb for that conversation and posting has not been reset or

canceled. (For information on posting, see the description of the POST_ON_RECEIPT verb).

3. The remote transaction program sends a REQUEST_TO_SEND notification to request the local transaction program to enter receive state and place the remote transaction program in send state. To enter receive state, a program issues either RECEIVE_AND_WAIT or PREPARE_TO_RECEIVE. The partner transaction program enters send state when the local transaction program issues either the RECEIVE_AND_WAIT verb or the RECEIVE_IMMEDIATE verb and receives the SEND indication on the WHAT_RECEIVED parameter.

WAIT

Waits for posting to occur on any basic conversation included in a list of conversations specified with this verb. Posting a conversation occurs when posting is active for the conversation and the LU has any information that the transaction program can receive, such as data, conversation status, or a request for confirmation.

WAIT	Supplied Parameters: TP_ID (variable) CONV_ID_LIST (variable1, variable2, ...)
	Returned Parameters: RETURN_CODE (variable) CONV_POSTED (variable)
	;

Supplied Parameters:

TP_ID specifies the identifier for the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the transaction program. For more information on the TP_ID parameter, see "CREATE_TP" on page 5-66 or "TP_STARTED" on page 5-58.

CONV_ID_LIST specifies the ID of each of the conversations that the LU can post.

- **variable1, variable2, ...** are the individual conversation IDs. The transaction program can specify one or more CONV_IDs in this list.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The WAIT return codes are:

- OK: The LU has posted one of the conversations in the CONV_ID_LIST.

- POSTED_DATA: The LU has posted a conversation and data is available in the LU's receive buffer.

- POSTED_NOT_DATA: The LU has posted a conversation and information other than data is available in the LU's receive buffer.

- PARAMETER_CHECK

  - BAD_TP_ID: APPC/PC does not recognize the specified TP_ID.

  - BAD_CONV_ID: APPC/PC does not recognize one of the CONV_ID values specified by CONV_ID_LIST.

- STATE_CHECK

  - NOT_RCV_STATE: One of the conversations in the CONV_LIST is not in receive state.

- POSTING_NOT_ACTIVE: Posting is not active for any of the conversations in the CONV_ID_LIST.

- DATA_POSTING_BLOCKED: One of the active conversations cannot be posted because the APPC/PC internal storage is temporarily being used. The local program cannot send a pacing response. This error does not cause APPC/PC to reset or cancel posting. The transaction program can continue issuing verbs, but to provide sufficient space for further posting, it should issue RECEIVE_IMMEDIATE or RECEIVE_AND_WAIT to free some space. Alternatively, the program can issue DEALLOCATE(TYPE=ABEND_PROG) to terminate a conversation.

  If you use receive pacing and issue a TEST or WAIT for more than R(P-1) bytes of data (where P = the receive pacing window, and R = the negotiated maximum RU size for the session), APPC/PC may report a buffer full condition with this return code. Even this amount of data may be too large if the

partner transaction program issues the FLUSH verb between SEND_DATA verbs causing messages of less than the maximum RU size to flow on the line. Posting is blocked if the allotted session buffers are filled for the conversation being posted. If you need to test for more data than this amount, reconfigure APPC/PC and increase the pacing window size or do not use pacing. If you increase the pacing window size, you must also increase the workspace size accordingly. However, if receive pacing is not used, APPC/PC will have no control of data sent from the partner transaction program. If its workspace becomes full APPC/PC abnormally terminates and rejects subsequent verbs.

For detailed information on the following return codes, see "Understanding Basic Conversation Return Codes" on page 7-5 and Appendix C, "Verb Return Codes."

- ALLOCATION_ERROR
- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DEALLOCATE_ABEND_PROG
- DEALLOCATE_ABEND_SVC
- DEALLOCATE_ABEND_TIMER
- DEALLOCATE_NORMAL
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- PROG_ERROR_NO_TRUNC
- PROG_ERROR_PURGING
- PROG_ERROR_TRUNC
- SVC_ERROR_NO_TRUNC
- SVC_ERROR_PURGING
- SVC_ERROR_TRUNC.

CONV_POSTED: indicates the CONV_ID of the posted (or blocked) conversation.

UNSUCCESSFUL indicates that APPC/PC could not successfully execute the function of the verb, or that the requested data or notification was not yet available. APPC/PC reports this return code to the local program on the verb issued. The state of the conversation remains unchanged.

State Changes:

None

Notes:

1. A transaction program can use this verb with the POST_ON_RECEIPT verb to receive information from multiple conversations in a synchronous fashion. The program must issue POST_ON_RECEIPT for each of the conversations and then issue WAIT for all of these conversations to wait until the LU has information for the program to receive from one or more of the conversations.

2. Posting may not be active for all of the conversations included in the CONV_LIST. This WAIT verb waits for posting to occur only on the conversations for which posting is active. If posting is not active for any of the conversations included in the CONV_LIST, APPC/PC reports the POSTING_NOT_ACTIVE return code to the program.

3. Posting is active for a conversation if the transaction program has issued the POST_ON_RECEIPT verb for that conversation and posting has not been reset or canceled. (For information on posting, see the description of the POST_ON_RECEIPT verb).

4. The return code OK indicates that the LU has posted one of the conversations included in the CONV_LIST for which posting is active. This return code also indicates that posting for that conversation is now reset. APPC/PC returns the CONV_ID of the posted conversation with the CONV_POSTED parameter. The transaction program should issue

RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE to receive the information from a posted conversation.

Chapter 8. Using the Network Management Verb

A fundamental capability of SNA is to manage a node or a network of nodes. SNA also provides facilities that an operator (programmed or human) can use to manage an SNA node or network of nodes.

A transaction program uses the TRANSFER_MS_DATA verb to provide management services information to a network management services function (that is, an SSCP).

The SSCP can use this information to manage the network to which this node is connected.

Understanding the Network Management Verb

The TRANSFER_MS_DATA verb builds and sends an unsolicited Network Management Vector Transport (NMVT) RU to a network management services function. A transaction program can also use the TRANSFER_MS_DATA verb to log user-defined data. Transaction programs can provide the following NMVT vector information to APPC/PC:

- Alerts
- Problem determination statistics (PDSTATS)
- Other network management data.

For information on the specific format of the NMVT RU, see the *SNA Reference Summary*.

Alerts

Alerts notify the control point management services that a system conversation is unavailable (or will soon become unavailable) to end users. An alert is the primary method that a transaction program can use to communicate problem determination information to the network operator.

Problem Determination Statistics

PDSTATS provide the SNA control point management services with error-rate data for communication links. This information assists the control point management services in determining and diagnosing problems associated with communication links.

Verb Description

TRANSFER_MS_DATA

TRANSFER_MS_DATA	Supplied Parameters: DATA_TYPE (USER_DEFINED) (NMVT) (ALERT_SUBVECTORS) (PDSTATS_SUBVECTORS) CORRELATOR_SUBVECTOR (ADD) (NO_ADD) PRODUCT_SET_ID_SUBVECTOR (ADD) (NO_ADD) SYSLOG (LOG) (NO_LOG) SSCP_PU_SESSION (SEND) (NO_SEND) DATA_LENGTH (variable) DATA (variable)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

DATA_TYPE specifies the variable that the transaction program uses to define the type of data it is providing. The valid data types are:

- **USER_DEFINED** specifies that the data is user defined. APPC/PC logs this data and does not send it on the SSCP-PU session.

- **NMVT** specifies that the data contains a complete NMVT. The data must include the Network Services (NS) header, the major Network Management Vector, and the appropriate subvectors.

- **ALERT_SUBVECTORS** specifies that the data contains management services information for an alert, in the format appropriate for an alert. (For specific details on alerts, see the *SNA Reference Summary*.) The data consists of the RU without the NMVT or the NS header which APPC/PC adds automatically.

- **PDSTATS_SUBVECTORS** specifies that the data contains management services information for problem determination statistics (PD Stats) in the format appropriate for this subvector. (For details on the formats for the PD Stats, see the *SNA Reference Summary*.) The data consists of the RU without the NMVT or the NS header which APPC/PC adds automatically.

CORRELATOR_SUBVECTOR specifies whether APPC/PC is to add the CORRELATOR subvector (X'42') to the NMVT. This parameter applies to all of the options identified by the DATA_TYPE parameter.

The two options for this parameter are:

- **ADD** directs APPC/PC to add the CORRELATOR subvector to the NMVT.

- **NO_ADD** directs APPC/PC not to add the CORRELATOR subvector to the NMVT.

PRODUCT_SET_ID_SUBVECTOR specifies whether APPC/PC adds the Product Set ID Subvector (X'10') to the data. Network management services use this information to identify the sender of the alert and the alerted conversation. This parameter applies to all of the entities identified by DATA_TYPE, *except* USER_DEFINED. APPC/PC ignores this parameter if the program specifies DATA_TYPE(USER_DEFINED). The two options for this parameter are:

- **ADD** directs APPC/PC to add the Product Set ID Subvector.

- **NO_ADD** directs APPC/PC not to add the Product Set ID Subvector.

SYSLOG specifies whether APPC/PC is to transfer the data to the PU SYSLOG exit. (For more information, see "ATTACH_PU" on page 5-17.) The two options for this parameter are:

- **LOG** directs APPC/PC to transfer the log data to the PU syslog exit.

- **NO_LOG** directs APPC/PC not to transfer the log data to the PU syslog exit.

SSCP_PU_SESSION specifies whether or not APPC/PC is to send the data on the SSCP-PU session. APPC/PC logs the data if the program specifies not to send the data. APPC/PC also logs the data if the program specifies to send the data, but the SSCP-PU session is not active. APPC/PC ignores this parameter if the program specifies DATA_TYPE(USER_DEFINED). Therefore, the program cannot send user-defined data on the SSCP-PU session. The two options for this parameter are:

- **SEND** directs APPC/PC to send the data on the SSCP-PU session.

- **NO_SEND** directs APPC/PC not to send the data on the SSCP-PU session.

DATA_LENGTH specifies the number of data bytes the program is sending (including the LL field). All data, including fields that APPC/PC adds, must fit within one RU (as specified for the adapter on the APPC/PC configuration menu).

DATA specifies the variable containing the management services data or the user-defined data that the program is sending. When the program specifies the TYPE(NMVT) this data includes only LL fields; APPC/PC adds the headers.

Returned Parameters:

RETURN_CODE indicates the result of verb execution. The TRANSFER_MS_DATA return codes are:

- APPC_ABENDED indicates that APPC/PC has been abnormally terminated.

- APPC_BUSY indicates that APPC/PC is executing another verb and cannot execute this verb. This error can occur if a verb is issued after APPC/PC execution is interrupted (for example, by a Ctrl-Break or timer interrupt).

- APPC_DISABLED indicates that APPC/PC is disabled by the DISABLE/ENABLE_APPC verb.

- DATA_EXCEEDS_RU_SIZE indicates that the data length exceeded the allowable RU size.

- OK indicates that APPC/PC executed the verb successfully.

- SSCP_PU_SESSION_NOT_ACTIVE indicates that APPC/PC could not send the NMVT because the SSCP-PU session was not active.

Chapter 9. Other APPC/PC Services

APPC/PC provides additional services for the convenience of the programmer. These functions include:

- A passthrough capability which enables an application subsystem to define its own verbs using the same interrupt vector as APPC/PC.
- An ASCII/EBCDIC conversion service.
- The ability to trace API invocations and sent and received messages.
- The ability to disable and re-enable the operation of APPC/PC.

Verb Descriptions

The descriptions of the verbs that provide these services are as follows:

SET_PASSTHROUGH

The SET_PASSTHROUGH verb provides the address of an exit to which APPC/PC branches when the application subsystem issues the PASSTHROUGH verb. This exit enables a transaction program to issue verbs to the application subsystem, using the same interrupt X'68' mechanism used by APPC/PC verbs. The exit procedure may itself issue further verbs.

Unlike most verbs, the program must define the information for the SET_PASSTHROUGH verb entirely within registers; there is no verb structure.

SET_PASSTHROUGH	Supplied Parameters: PASSTHROUGH_ADDRESS (variable) ;

Supplied Parameters:

PASSTHROUGH_ADDRESS specifies the exit address to which APPC/PC branches when the program issues the PASSTHROUGH verb.

PASSTHROUGH

Permits an application subsystem to define its own verbs using the interrupt vector X'68'. APPC/PC ignores these user-defined verbs as it passes them through to the application subsystem.

When the program issues the PASSTHROUGH verb, APPC/PC performs a far branch (not a call) to the specified exit procedure. APPC/PC leaves hardware interrupts inhibited and registers untouched. The application subsystem uses the SET_PASSTHROUGH verb to specify the address of the exit procedure.

The exit procedure may itself issue APPC/PC verbs.

The PASSTHROUGH verb enables a program to access application subsystem services without using another software interrupt. The application subsystem must define its own verbs for transaction programs to access through this verb.

CONVERT

CONVERT provides a utility service for ASCII/EBCDIC conversion. This verb operates on a specified character string to produce a converted character string.

APPC/PC assumes that all names that a program passes over the interrupt X'68' interface use EBCDIC coding. Therefore, programs that use ASCII coding for APPC/PC names (LU names, TP names, passwords, and so on) must convert these names to EBCDIC. Conversely, the application subsystem may need to convert incoming names from EBCDIC to ASCII.

A program may also need to perform data conversion if it is communicating with a node that expects EBCDIC data.

Appendix H, "ASCII/EBCDIC Translation Tables" provides the tables APPC/PC uses to convert data from EBCDIC to ASCII and ASCII to EBCDIC. You can specify your own (type G) conversion table on the APPC/PC configuration menus.

CONVERT	Supplied Parameters: DIRECTION (ASCII_TO_EBCDIC) (EBCDIC_TO_ASCII) LENGTH (variable) SOURCE (variable) TARGET (variable) CHARACTER_SET (AE) (A) (G)
	Returned Parameters: RETURN_CODE (variable)
	;

Supplied Parameters:

DIRECTION specifies whether conversion is ASCII to EBCDIC or EBCDIC to ASCII. The two options for this parameter are:

- ASCII_TO_EBCDIC: Directs APPC/PC to convert ASCII characters to EBCDIC.

- EBCDIC_TO_ASCII: Directs APPC/PC to convert EBCDIC characters to ASCII.

LENGTH specifies the number of characters in the string APPC/PC is to convert.

SOURCE specifies the variable containing the string APPC/PC is to convert.

Warning: The sum of the SOURCE offset and the LENGTH must be less than 65535 to prevent the area for the SOURCE string from overlapping a segment boundary.

TARGET specifies the variable that is to receive the converted character string.

Warning: The sum of the TARGET offset and the LENGTH must be less than 65535 to prevent the area for the TARGET string from overlapping a segment boundary.

If the program does not need to preserve the source string, it can specify the same variable for the SOURCE and TARGET parameters.

CHARACTER_SET enables the program to specify the set of characters permitted in the source string. The options for this parameter are:

- Type AE: Permits the source string to include lowercase a-z; uppercase A-Z; numerics 0-9; and special characters $, #, @, and the period (.). This option does not place any restrictions on the first character in the source string; it also permits the string to include trailing blanks.

- Type A: Permits the source string to include uppercase A-Z; numerics 0-9; and special characters $, #, and @. In the ASCII to EBCDIC direction, the type A option accepts lowercase characters and converts them to uppercase EBCDIC characters. This option requires the first character of the string to be an uppercase letter or one of the three special characters. The type A option also permits the source string to include trailing blanks.

- Type G: This option permits the source string to include any character. APPC/PC converts the source string according to the conversion code table that you specify on the APPC/PC configuration menus. If no such table exists, the program should not specify type G conversion.

Returned Parameters:

RETURN_CODE indicates whether the conversion was successful.

- APPC_ABENDED indicates that APPC/PC has been abnormally terminated.

- CONVERSION_ERROR indicates that the program specified the CHARACTER_SET(A) or CHARACTER_SET(AE) parameter and APPC/PC encountered one or more characters in the source string that are not defined in the selected conversion table. APPC/PC converts undefined characters to X'00'. This error cannot occur if the program specifies the CHARACTER_SET(G) parameter.

- INVALID_DIRECTION indicates that APPC/PC does not recognize the specified DIRECTION.

- INVALID_FIRST_CHARACTER indicates that the program specified the CHARACTER_SET(A) parameter and the first character in the source string does not satisfy the requirements of this character set specification.

- INVALID_TYPE indicates that APPC/PC does not recognize the specified CHARACTER_SET type.

- OK indicates that the code conversion was successful.

- SEGMENT_OVERLAP indicates that the area specified for either the source or target string area overlaps a segment boundary.

- TABLE_ERROR indicates that APPC/PC could not locate the type G conversion table.

TRACE

TRACE enables or disables the APPC/PC tracing function. After a program enables tracing, APPC/PC can trace verbs through the API, and trace sent and received messages above the adapter layer. APPC/PC passes trace parameters in registers instead of using a control block. The program can direct APPC/PC to display, print, file, or save trace messages in a storage buffer. APPC/PC uses ASCII coding for trace messages.

TRACE	Supplied Parameters: TRACE_MESSAGES (ON (TRUNC(variable))) (OFF) TRACE_API (ON) (OFF) TRACE_DESTINATION (DISPLAY) (PRINTER) (STORAGE TRACE_STATS(variable)) (FILE TRACE_STATS(variable))
	;

Supplied Parameters:

TRACE_MESSAGES directs APPC/PC to enable or disable tracing. The two options for this parameter are ON and OFF. The ON specification includes a TRUNC specification that determines how many characters of a message the trace can return.

- ON directs APPC/PC to begin tracing sent and received messages.

  In addition to the messages, a trace indicates when a link becomes active by returning LINK CONNECTED. When the link becomes inactive because of a link error, the trace indicates LINK INOPERATIVE.

  TRUNC specifies the maximum number of characters in the RU portion of the trace that APPC/PC is to return. APPC/PC truncates any characters exceeding

this maximum number. A value of 0 specifies that APPC/PC should not truncate the RU portion of the trace.

- OFF directs APPC/PC to stop tracing messages.

TRACE_API enables and disables the tracing of verbs passing through the API. The two options for this parameter are ON and OFF.

- ON: Directs APPC/PC to trace verbs passed through the API.

  This option returns the first 64 bytes of each traced verb in hexadecimal format.

- OFF: Stops the tracing of verbs passed through the API.

TRACE_DESTINATION determines whether APPC/PC directs the tracing output to the IBM PC screen, printer, storage, or a file. A program can specify more than one destination for the tracing output. The options for this parameter are:

- DISPLAY: Directs APPC/PC to display the trace output on the IBM PC screen. The operator can stop and restart the display of the trace output by pressing the Scroll Lock key.

- PRINTER: Directs APPC/PC to send the trace output to the IBM PC printer (LPT1:). The operator can stop and restart the display of the trace output by pressing the Scroll Lock key.

- STORAGE: Directs APPC/PC to place the trace output in internal storage. This option requires the application subsystem to provide a routine for printing the trace output stored in this buffer. This option stores 80 character records, the last two characters of which are carriage return and line feed (X'0D0A').

  When the program directs APPC/PC to place the trace output in an internal storage buffer, it must also

include certain information in the TRACE_STATS parameter area of the TRACE verb. The TRACE_STATS parameter area must provide the address of the buffer that is to receive the trace output, the maximum number of records to place in the buffer, and an initialized space for trace statistics. For a description of the contents of the TRACE_STATS buffer, see the description of the TRACE verb in Appendix A, "Verb Operation Codes and Formats."

After the trace output fills the buffer with the specified maximum number of records, it wraps around to begin placing new records at the front of the buffer again.

- FILE: Directs APPC/PC to place the trace output in a file named OUTPUT.PC (in the same directory as APPC/PC).

  When the program directs APPC/PC to place the trace output in a file, it must also include certain information with the TRACE_STATS parameter area of the TRACE verb. For the FILE option, the TRACE_STATS parameter area must provide the maximum number of records to place in the file, and an initialized space for trace statistics. After the trace output fills the file with the specified maximum number of records, it wraps around to begin placing new records at the front of the file again.

The following table illustrates the tracing formats for API Entry Format, API Return Format, Sent Message Format, and Received Message Format:

Tracing Message Formats

```
API Entry Format
API req          5D00:20C6
     2E007E3D 2E007E3D C620005D 01000000        <..==..==F .)....>
     00000000 00000000 00000000 7E3D0A00        <............==..>
     00000000 00000001 00000000 00000000        <................>
     0000004C 55534944 4532204D 4F444531        <...LUSIDE2 MODE1>
     20202008 54505349 44453241 20202020        <   .TPSIDE2A    >
     20202020 20202020 20202020 20202020        <                >
```

```
<=S==    #:6000 TH:2D000101000E RH:6B8000
 RU: 31001307 B0B0D0B1 01008585 80010602 00000000 00000000 20000008 4C555349
     44453120 28000902 4D4F4445 31202020 09030000 00000F00 00001204 41505043
     4E455420 4B4C5553 49444531 2000084C 55534944 453220

==R=>    #:4F00 TH:2D000101000E RH:EB8000
 RU: 31001307 B0B0D0B1 01018585 81010602 00000000 00000000 20000000 28000902
     4D4F4445 31202020 09030000 00000F00 00001205 41505043 4E455420 4B4C5553
     49444532 2000

API ret          5D00:20C6
     67007E3D 67007E3D C620005D 01000000        <g.==g.==F .)....>
     00000000 00000000 00000000 7E3D0A00        <............==..>
     00000000 CB3D0001 00000000 00000000        <.....=.........>
     0000004C 55534944 4532204D 4F444531        <...LUSIDE2 MODE1>
     20202008 54505349 44453241 20202020        <   .TPSIDE2A    >
     20202020 20202020 20202020 20202020        <                >

API req indicates an issued verb at the given address
API ret indicates a completed verb at the given address
<=S==   indicates a message being sent
==R=>   indicates a message being received
#       indicates the size in hexidecimal of the RU
TH      indicates the Transmission Header
RH      indicates the Request or Response Header
          The second line and succeeding lines describe the RU in
          groups of 8 hexidecimal digits (4 bytes).
...     indicates APPC/PC has performed truncation
          according to the TRUNC specification.
```

DISABLE/ENABLE_APPC

This verb provides a program with the ability to enable and disable the operation of APPC/PC. A program can use this verb to avoid recursion problems in exit routines after it intercepts a call to DOS or BIOS (if those exit routines include calls to DOS or BIOS).

APPC/PC does not end completely when the application subsystem specifies the DISABLE option. Instead, the adapters continue operating to queue incoming messages for APPC/PC. APPC/PC processes these queued messages after the program specifies the ENABLE option.

In addition to the DISABLE/ENABLE_APPC verb, APPC/PC can execute the CONVERT, TRACE, SET_PASSTHROUGH, and PASSTHROUGH verbs when it is operating in the disabled state. In normal operation, the program should not issue any other APPC/PC verbs. If the program issues one of the other APPC/PC verbs, it receives an APPC_DISABLED return code.

DISABLE/ENABLE_APPC	Supplied Parameters: DISABLE_OR_ENABLE (DISABLE) (ENABLE)
	;

Supplied Parameters:

DISABLE_OR_ENABLE specifies whether to disable or enable APPC/PC operation. The two options for this parameter are:

- DISABLE: Suspends APPC/PC operation.
- ENABLE: Resumes APPC/PC operation.

Chapter 10. Resolving Error Conditions

This chapter describes different kinds of error conditions and indicates possible solutions.

The types of error conditions that can occur during APPC/PC operation are:

- Errors indicated with a return code
- Errors indicated in a log
- System deadlocks.

Note that APPC/PC can detect return code and log errors but cannot detect system deadlocks.

Return Code Error Indications

When an application subsystem or transaction program issues a verb, APPC/PC uses the RETURN_CODE parameter to indicate the success or failure of the request. The program requesting APPC/PC services by issuing the verb must check the value of the return code to determine its next action. Appendix C, "Verb Return Codes" contains the complete list of return codes that APPC/PC can report after it executes (or attempts to execute) each verb. This appendix also suggests the appropriate actions to correct the problem causing the error.

Logged Errors

During processing, APPC/PC can detect error conditions that are not directly associated with verb requests. In addition to APPC/PC, a transaction program at either end of a conversation can also detect these errors.

When APPC/PC detects this type of error, it builds a SYSLOG control block, and invokes the error exit routine that the application subsystem specifies. Use of a log exit routine is recommended because APPC/PC itself does not display error messages.

System Deadlocks

Using Multiple Active Transaction Programs

System deadlocks can occur as a result of one verb blocking the completion of another verb, or due to other unforeseen conditions. This section describes methods of diagnosing and correcting system deadlocks.

Under a single-tasking operating system such as PC DOS on the IBM PC, certain deadlock conditions are possible which can be prevented by careful design of the application subsystem.

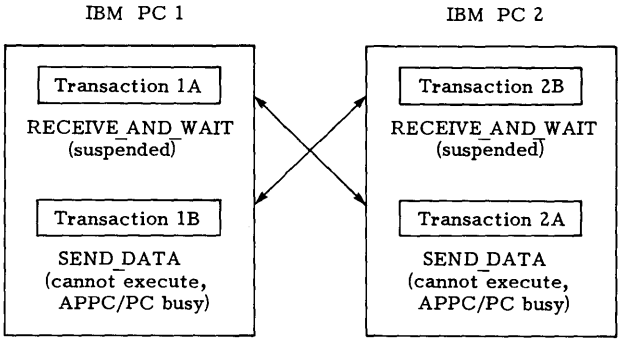
These deadlocks involve "blocking" verbs that do not return control to the transaction program until some form of data or indicator has been received from the remote partner. The remote partner must issue a verb on the conversation before control at the local node can be returned. Deadlock occurs when the partner cannot issue that verb because it too is issuing a blocking verb on a different simultaneous conversation.

The potential blocking verbs include most conversation verbs. (Any verb that lists INCOMPLETE as a possible return code is a potential blocking verb.)

Such deadlocks are possible only when all nodes involved in the simultaneous conversations operate under a single-tasking system (which is the case if all nodes are IBM PCs), and only if the graph of such nodes forms a loop. For example, using parallel sessions between two IBM PCs forms a loop between them.

Such deadlocks are not possible if you support only one active transaction program (one TP_ID) at every single-tasking node. In this case no such loop is possible because you can allocate a conversation only to a new remote transaction program, not to an existing one.

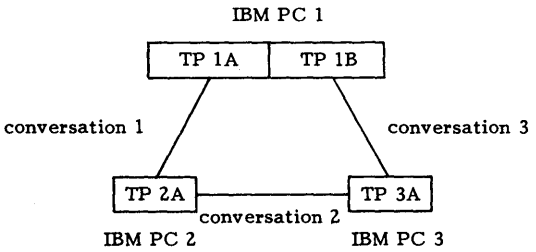
An example of a situation where two parallel conversations can experience a blocking deadlock condition is when each of the partner programs waits for data after issuing a RECEIVE_AND_WAIT verb. While these transaction programs wait, no other transaction program at either node can issue a SEND_DATA to satisfy its partner. The figure below illustrates this deadlock situation.



If Transaction 1A (in IBM PC 1) is conversing with Transaction 2A (in IBM PC 2), and issues a

RECEIVE_AND_WAIT on that conversation, IBM PC 1 waits until data is received to satisfy the RECEIVE_AND_WAIT. If Transaction 2B (in IBM PC 2) is conversing with Transaction 1B (in IBM PC 1) concurrently, and issues a RECEIVE_AND_WAIT on that conversation, then IBM PC 2 also waits to receive data.

Deadlocks are also possible between more than two nodes even when only one session and one conversation exists between any two nodes. For example, in the figure below, if TP 1A in IBM PC 1 ALLOCATEs conversation 1 to TP 2A in IBM PC 2, TP 2A ALLOCATEs conversation 2 to TP 3A in IBM PC 3, and TP 3A in turn ALLOCATEs conversation 3 to TP 1B in IBM PC 1, then TP 1A may issue a blocking verb on conversation 1, TP 2A may issue one on conversation 2, and TP 3A may issue one on conversation 3, each waiting for a verb at the partner which will not be issued. The figure below illustrates this deadlock situation.



Use of pacing can also cause a deadlock. A local transaction program can be prevented from sending data because its remote partner is blocked from issuing a receive verb which would clear the session buffers of the remote partner. If the session buffers are not cleared, the partner cannot transmit the pacing response required before the local send can occur.

If the system appears to be in a deadlock state and if the configuration of APPC/PC included the DOS Ctrl-Break

option, the operator may be able to return to the DOS prompt by pressing Ctrl-Break. For a description of Ctrl-Break operation, see "Canceling a Transaction" on page 2-13 in Chapter 2, "Developing an Application Subsystem."

Alternately, you can bring down the connection by turning off the modem, if there is one. Any existing conversations on this line will fail.

Designing to Avoid System Deadlocks

There are two methods of avoiding deadlocks caused by issuing blocking verbs:

Method 1 (Recommended): Using the INCOMPLETE option

Specify RETURN_CONTROL=INCOMPLETE in the ATTACH_PU verb, and expect return codes of INCOMPLETE on conversation verbs, as well as on TP_VALID and TP_ENDED. APPC/PC will return control to the application, rather than suspend operation even if a verb is unfinished, so that verbs that were blocked can now be issued.

You must issue verbs on all other transaction programs, then re-issue the incomplete verb (or verbs, if several verbs from different transaction programs are incomplete) without alteration. You can, however, use the first 12 bytes to keep the incomplete verbs in a list. You may have to issue a verb several times before it completes.

If you are using synchronous management of incoming ALLOCATEs (queueing them in the LU), and have not stopped queueing by issuing a CHANGE_LU verb, you must also periodically issue GET_ALLOCATE, because a queued ALLOCATE can also cause blocking of the partner.

In order to manage multiple simultaneous transaction programs, the application subsystem can implement some form of scheduler to issue verbs from the various transaction programs. With Method 1, the scheduler must get INCOMPLETE return codes, save the verb (or a pointer to it) and periodically re-issue the verb until it completes. You should, therefore, use a different verb storage buffer, as well as a different data buffer, for each transaction program.

TP_VALID can return INCOMPLETE only if the incoming ALLOCATE was rejected with a non-0 sense code (in which case the equivalent of SEND_ERROR is performed). You do not need to test for INCOMPLETE if you accept the incoming ALLOCATE.

TP_ENDED can return INCOMPLETE only if it was issued before ending any existing conversations (in which case the equivalent of DEALLOCATE TYPE = ABEND is performed). You do not need to test for INCOMPLETE if all conversations have been terminated before you issue TP_ENDED.

Method 2: Restrictions on your protocol design

Impose the following restrictions on your transaction programs and application subsystem:

1. Do not use pacing.

2. Send small amounts of data (using one or more consecutive SEND_DATA verbs) at a time. Wait for data from the partner that indicates it has received what you have sent before sending more. The amount that you can send at one time depends on the storage that is made available to your node and the partner. (For more information, see "APPC/PC Workspace Requirements" in the *APPC/PC Installation and Configuration Guide.*)

3. Do not use conversations with a synchronization level of CONFIRM.

4. Do not use the following blocking verbs:

   - ALLOCATE or MC_ALLOCATE (TYPE = WHEN_SESSION_ALLOCATED)
   - SEND_ERROR or MC_SEND_ERROR
   - RECEIVE_AND_WAIT or MC_RECEIVE_AND_WAIT
   - DEALLOCATE(TYPE = ABEND_PROG, ABEND_SVC, ABEND_TIMER)
   - MC_DEALLOCATE(TYPE = ABEND)
   - WAIT.

5. If you use synchronous management of incoming ALLOCATEs, stop queueing (by issuing the CHANGE_LU verb) when any transaction program is initiated. You can start multiple transaction programs by using the asynchronous exit.

6. Do not issue TP_VALID with a non-0 sense code in the CREATE_TP structure. Use the asynchronous exit if you need to test, and possibly reject, incoming ALLOCATEs.

7. Issue TP_ENDED only after all conversations of the transaction program have been deallocated.

Investigating a Deadlock Situation

The TRACE verb can provide valuable information for debugging many problems, including deadlock situations. For a description of the TRACE verb, see "TRACE" on page 9-8 in Chapter 9, "Other APPC/PC Services." You can determine the point at which a deadlock occurs by enabling tracing. Tracing indicates both the start and finish of verb execution. This information can indicate which verb was the last one to start before the deadlock occurred. You can also use tracing to check the contents of the parameter lists for each verb as it executes.

Appendix A. Verb Operation Codes and Formats

This appendix defines the operation codes for APPC/PC verbs and the internal formats for the parameter lists passed between a transaction program and APPC/PC.

APPC/PC to Transaction Program Verbs

APPC/PC performs a FAR CALL to execute verbs that send information from APPC/PC to the application subsystem (SYSLOG, CREATE_TP, ACCESS_LU_LU_PW). The SS : SP register pair points to an 8-byte field on the stack. The 8 bytes consist of:

- A return address (4 bytes in IBM PC byte order)
- A verb pointer (4 bytes in IBM PC byte order).

In many programming languages, this procedure is the standard calling convention used when a program passes a single parameter (such as a verb pointer) to a subroutine. When the routine begins, at least 64 bytes of stack space are available on the stack for use by the routine.

When the routine is complete and ready to return control to APPC/PC, it must remove the verb pointer parameter from the stack. In IBM PC Macro Assembler language, the return statement would appear as follows:

```
create_tp proc    far
        .
        .
        .
        ret     4   ;Return to APPC after
                    ;removing verb pointer
create_tp endp
```

Transaction Program to APPC/PC Verbs

All verb requests come to APPC/PC through interrupt vector X'68'. The transaction program can verify that APPC/PC is loaded by checking for the ASCII character string 'APPC/PC' starting 9 bytes prior to the address pointed to by interrupt vector X'68' (storage location 416).

Before issuing the interrupt, the transaction program sets the AH register as indicated by the record format for each verb and points the DS:DX register pair to the location of the verb parameter list.

If APPC/PC receives an invalid AH value from the transaction program, it rejects the verb and places a value of X'FF' in the AL register. APPC/PC indicates a valid AH value by setting AL to 0. The AH setting for each verb appears immediately after the header line for the verb record. Following the AH setting for each verb is the parameter list. The first parameter you supply on this list is the operation code that identifies the verb.

Verb Operation Codes

The following is a list of the verb operation codes and their corresponding hexadecimal values.

Op Code (hex)	AH Value	Verb Name
0100	2	ALLOCATE or MC_ALLOCATE
0200	2	(reserved)
0300	2	CONFIRM or MC_CONFIRM
0400	2	CONFIRMED or MC_CONFIRMED
0500	2	DEALLOCATE or MC_DEALLOCATE
0600	2	FLUSH or MC_FLUSH
0700	2	GET_ATTRIBUTES or MC_GET_ATTRIBUTES
0800	2	GET_TYPE
0900	2	POST_ON_RECEIPT
0A00	2	PREPARE_TO_RECEIVE or MC_PREPARE_TO_RECEIVE
0B00	2	RECEIVE_AND_WAIT or MC_RECEIVE_AND_WAIT
0C00	2	RECEIVE_IMMEDIATE or MC_RECEIVE_IMMEDIATE
0D00		(reserved)
0E00	2	REQUEST_TO_SEND or MC_REQUEST_TO_SEND
0F00	2	SEND_DATA or MC_SEND_DATA
1000	2	SEND_ERROR or MC_SEND_ERROR
1100	-	(reserved)
1200	2	TEST or MC_TEST
1300	2	WAIT
1400	-	(reserved)
1500	6	CNOS
1600	-	(reserved)
1700	-	(reserved)
1800	-	(reserved)
1900	-	ACCESS_LU_LU_PW
1A00	251	CONVERT
1B00	1	DISPLAY

Op Code (hex)	AH Value	Verb Name
1C00	5	TRANSFER_MS_DATA
1D00	-	(reserved)
1E00	-	(reserved)
1F00	-	(reserved)
2000	1	ATTACH_PU
2100	1	ATTACH_LU
2200	1	DETACH_LU
2300	-	CREATE_TP
2400	3	TP_STARTED
2500	4	TP_ENDED
2600	-	SYSLOG
2700	1	DETACH_PU
2800	3	GET_ALLOCATE
2900	4	TP_VALID
2A00	3	CHANGE_LU
2B00	1	ACTIVATE_DLC

Verb Record Formats

The rest of this appendix contains the record formats for the APPC/PC verbs. The records appear in alphabetic order according to the verb names.

Each record begins with a header line that specifies the AH value. Following the header line, each record includes four columns:

- **Disp** indicates the displacement of the parameter from the beginning of the record.
- **Lng** indicates the length of the parameter field in bytes.
- **Value** indicates any constant values defined for the parameter.
- **Field Name** indicates the name of the parameter field and the names of constant parameter values (indented below the name of the constant parameter).

The program using APPC/PC must set all "Reserved" parameters to 0.

Most data fields of the verb are in normal order with the high order byte in the lower address. However, for values designated by an asterisk (*) in the "Length" column, the value is in the reversed-order style of the IBM PC. That is, for 2-byte values (such as a length field), the IBM PC stores the most significant byte in the right-most byte of the field. The decimal value appears within parentheses for such a field if it is a non-0 constant.

Similarly, for 4-byte values (such as an address field), the most significant byte is stored in byte 3 (right-most) of the field, the next-most significant byte is stored in byte 2 of the field, the next in byte 1 of the field, and the least-significant byte is stored in byte 0 of the field.

The IBM PC uses the following ordering conventions for bit fields: bit 0 is the left-most bit of a field; bit 7 is the right-most bit of a byte; and bit 15 is the right-most bit of a half-word.

All data areas passed to APPC/PC, either for sending or receiving data, must fit within a single segment. For example, when issuing the RECEIVE_AND_WAIT verb, a program provides a 32-bit pointer and a 16-bit maximum length value specifying the area into which APPC/PC places received data. All bytes in the range delimited by these values must have the same segment value.

For example, a transaction program violates this restriction if the data area pointer was segment X'1234', the offset was X'FFF0' and the length was X'0030'. In this case, the sum of the offset and length exceeds X'FFFF', the maximum offset value within a segment. APPC/PC provides individual return codes to indicate violations of this restriction.

A double asterisk (**) in the "Value" column indicates that APPC/PC uses the field internally.

APPC/PC operation requires at least 256 bytes of stack space. At least 320 bytes must be available if the program

specifies tracing. This allocation provides 64 bytes of stack space for interrupt processing while using APPC/PC.

Note:

All numbers in the following formats are decimal unless preceded by an X (meaning hexadecimal) or a B (meaning bit). In this appendix, the abbreviation CS:IP refers to the Code Segment: Instruction Pointer (offset) values.

ACCESS_LU_LU_PW

record format

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'1900' (25)	Verb operation code
14	8		LU_ID
22	8		LU_NAME
30	8		PARTNER_LU_NAME
38	17		PARTNER_FULLY_QUALIFIED_LU_NAME
55	1		PASSWORD_AVAILABLE
		X'00'	NO
		X'01'	YES
56	8		PASSWORD

ACTIVATE_DLC

record format (AH = 1)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'2B00' (43)	Verb operation code
14	6	6X'00'	Reserved
20	4		RETURN_CODE
		X'00000000'	OK
		X'00000008'	NO_PU_ATTACHED
		X'00000283'	DLC_FAILURE
		X'00000284'	UNRECOGNIZED_DLC
		X'00000286'	DUPLICATE_DLC

Disp	Lng	Value	Field Name
		X'F0010000'	APPC_DISABLED
		X'F0020000'	APPC_BUSY
		X'F0030000'	APPC_ABENDED
24	8		DLC_NAME
			ITRN
			SDLC
32	1		ADAPTER_NUMBER

ALLOCATE and MC_ALLOCATE

record format (AH = 2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'0100' (1)	Verb operation code
14	1		Verb extension code
		X'00'	ALLOCATE
		X'01'	MC_ALLOCATE
15	5	5X'00'	Reserved
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0014'	UNSUCCESSFUL
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
		X'F004'	INCOMPLETE
22	4		Secondary RETURN_CODE
		X'00000001'	BAD_TP_ID
		X'00000004'	ALLOCATION_FAILURE_NO_RETRY
		X'00000005'	ALLOCATION_FAILURE_RETRY
		X'00000006'	DATA_AREA_ACROSS_SEGMENT
		X'00000010'	BAD_TPN_LEN
		X'00000011'	BAD_CONV_TYPE
		X'00000012'	BAD_SYNC_LEVEL
		X'00000013'	BAD_SECURITY_SELEC
		X'00000014'	BAD_RETURN_CONTROL
		X'00000015'	TOO_BIG_SEC_TOKENS
		X'00000016'	PIP_LEN_INCORRECT
		X'00000017'	NO_USE_OF_SNASVCMG
		X'00000018'	UNKNOWN_PARTNER_MODE
26	8		TP_ID
34	4		CONV_ID
38	1		CONVERSATION_TYPE (note 1)
		X'00'	BASIC_CONVERSATION
		X'01'	MAPPED_CONVERSATION (user supported)
39	1		SYNC_LEVEL
		X'00'	NONE
		X'01'	CONFIRM

Disp	Lng	Value	Field Name
40	2	2X'00'	Reserved
42	1		RETURN_CONTROL
		X'00'	WHEN_SESSION_ALLOCATED
		X'01'	IMMEDIATE
		X'02'	WHEN_SESSION_FREE
43	8	8X'00'	Reserved
51	8		PARTNER_LU_NAME
59	8		MODE_NAME
67	1		TP name length
68	64		TP name
132	1		SECURITY
		X'00'	NONE
		X'01'	SAME
		X'02'	PGM
133	11	11X'00'	Reserved
144	1		Password length
145	10		PASSWORD
155	1		User ID length
156	10		USER_ID
166	2*		PIP_DATA_LENGTH (0 if no PIP data)
168	4*		PIP_DATA

Note 1: Reserved if MC_ALLOCATE.

Note 2: Not applicable if MC_ALLOCATE.

ATTACH_LU

record format (AH = 1)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'2100' (33)	Verb operation code
14	6	6X'00'	Reserved
20	4		RETURN_CODE
		X'00000000'	OK
		X'00000008'	NO_PU_ATTACHED
		X'00000211'	ALREADY_ACTIVE_LU
		X'00000212'	BAD_PART_SESS
		X'00000213'	BAD_RU_SIZES
		X'00000214'	BAD_MODE_SESS
		X'00000216'	BAD_PACING_CNT
		X'00000219'	EXTREME_RUS
		X'0000021A'	SNASVCMG_1
		X'00000284'	UNRECOGNIZED_DLC
		X'F0010000'	APPC_DISABLED
		X'F0020000'	APPC_BUSY

Disp	Lng	Value	Field Name
		X'F0030000'	APPC_ABENDED
24	2*	X'4600' (70)	Offset to partner LU record length field
26	8		LU_NAME
34	8		LU_ID
42	1		LU_LOCAL_ADDRESS
43	1		LU_SESSION_LIMIT
44	4*		CREATE_TP_EXIT (in CS :IP format)
		X'FFFFFFFF'	Reject incoming ALLOCATES
		X'00000000'	QUEUE_ALLOCATES (YES)
48	4*	4X'00'	Reserved
52	4*		SYSTEM_LOG_EXIT (in CS :IP format)
		X'FFFFFFFF'	Do not log errors
56	4*	4X'00'	Reserved
60	1		MAX_TPS
61	1		QUEUE_DEPTH
62	4*		LU_LU_PASSWORD_EXIT (in CS :IP format)
		X'FFFFFFFF'	No password exit
66	4*	4X'00'	Reserved
70	2*		Total length of partner LU records
		(Repeat for each partner LU)	
72+0	2*		Length of this partner LU record (including parameters for each mode name defined)
2	2*	X'2A00' (42)	Offset to start of mode records
4	8		PARTNER_LU_NAME
12	1		PARTNER_LU_SECURITY_CAPABILITIES
bits 0-4		B'00000'	Reserved
bit 5			Session level security
		B'0'	Not supported
		B'1'	Supported
bit 6			Conversation level security
		B'0'	Not supported
		B'1'	Supported
bit 7			Already verified
		B'0'	Not supported
		B'1'	Supported
13	1		PARTNER_LU_SESSION_LIMIT
14	2*		PARTNER_LU_MAX_MC_SEND_LL
16	8		PARTNER_LU_DLC_NAME
24	1		PARTNER_LU_ADAPTER_NUMBER
25	1		Length of partner LU adapter address
26	16		PARTNER_LU_ADAPTER_ADDRESS
42	2*		Total length of all mode name records
44+0	2*	X'1000' (16)	Length of this mode name record

Disp	Lng	Value	Field Name
		(Repeat for each mode)	
2	8		MODE_NAME
10	2*		RU_SIZE high bound
12	2*		RU_SIZE low bound
14	1		MODE_MAX_NEGOTIABLE_ SESSION_LIMIT
15	1		PACING_SIZE (for receive)

ATTACH_PU

record format (AH = 1)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'2000' (32)	Verb operation code
14	6	6X'00'	Reserved
20	4		RETURN_CODE
		X'00000000'	OK
		X'00000201'	ALREADY_ACTIVE_PU
		X'F0010000'	APPC_DISABLED
		X'F0020000'	APPC_BUSY
		X'F0030000'	APPC_ABENDED
24	2	X'0000'	Reserved
26	1		VERSION
27	1		RELEASE
28	8		NET_NAME
36	8		PU_NAME
44	8	8X'00'	Reserved
52	4*		SYSTEM_LOG_EXIT (in CS :IP format) (X'FFFFFFFF' = > none)
56	4	4X'00'	Reserved
60	1		RETURN_CONTROL
		X'00'	COMPLETE
		X'01'	INCOMPLETE

CHANGE_LU

record format (AH = 3)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'2A00' (42)	Verb operation code
14	6	6X'00'	Reserved
20	4		RETURN_CODE
		X'00000000'	OK
		X'00000003'	BAD_LU_ID
		X'00000230'	INVALID_CHANGE
		X'F0010000'	APPC_DISABLED
		X'F0020000'	APPC_BUSY
		X'F0030000'	APPC_ABENDED
24	2	2X'00'	Reserved
26	8		LU_ID
34	4*		CREATE_TP_EXIT
			(in CS :IP format)
		X'FFFFFFFF'	Reject incoming ALLOCATES
		X'00000000'	QUEUE_ALLOCATES (YES)
38	4*	4X'00'	Reserved
42	4*		SYSTEM_LOG_EXIT
			(in CS :IP format)
		X'FFFFFFFF'	Do not log errors
46	4*	4X'00'	Reserved
50	1		MAX_TPS
51	1		QUEUE_ALLOCATES
		X'00'	STOP
		X'01'	RESUME
52	4*		LU_LU_PASSWORD_EXIT
		X'FFFFFFFF'	No password exit
56	4*	4X'00'	Reserved

CNOS

record format (AH = 6)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'1500' (21)	Verb operation code
14	6	6X'00'	Reserved
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK

Disp	Lng	Value	Field Name
		X'0003'	ALLOCATION_ERROR
		X'0018'	CNOS_PARTNER_LU_REJECT
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
22	4		Secondary RETURN_CODE
		X'00000000'	CNOS_ACCEPTED
		X'00000001'	CNOS_NEGOTIATED
		X'00000003'	BAD_LU_ID
		X'00000004'	ALLOCATION_FAILURE_NO_RETRY
		X'00000005'	ALLOCATION_FAILURE_RETRY
		X'00000151'	CANT_RAISE_LIMITS
		X'00000153'	ALL_MODE_MUST_RESET
		X'00000154'	BAD_SNASVCMG_LIMITS
		X'00000155'	MINS_GT_TOTAL
		X'00000156'	MODE_CLOSED (primary RETURN_CODE - X'0001')
		X'00000156'	CNOS_MODE_CLOSED (primary RETURN_CODE - X'0018')
		X'00000157'	BAD_MODENAME (primary RETURN_CODE - X'0001')
		X'00000157'	CNOS_BAD_MODENAME (primary RETURN_CODE - X'0018')
		X'00000159'	RESET_SNA_DRAINS
		X'0000015A'	SINGLE_NOT_SRC_RESP
		X'0000015B'	BAD_PARTNER_LU
		X'0000015C'	EXCEEDS_MAX_ALLOWED
		X'0000015D'	CHANGE_SRC_DRAINS
		X'0000015E'	LU_DETACHED
		X'0000015F'	CNOS_COMMAND_RACE_REJECT
26	8		LU_ID
34	8	8X'20'	Reserved
42	8		PARTNER_LU_NAME
50	8		MODE_NAME (if MODE_NAME_SELECT=ONE)
58	1		
bits 0-5		B'000000'	Reserved
bit 6			SET_NEGOTIABLE
		B'0'	Do not set negotiable values
		B'1'	Set negotiable values
bit 7			MODE_NAME_SELECT
		B'0'	ONE
		B'1'	ALL
59	1		PARTNER_LU_MODE_SESSION_LIMIT
60	1		MIN_CONWINNERS_SOURCE
61	1		MIN_CONWINNERS_TARGET
62	1		AUTO_ACTIVATE
63	1	X'00'	Reserved
64	1		Termination settings
bits 0-4		B'00000'	Reserved
bit 5			RESPONSIBLE
		B'0'	SOURCE (local)
		B'1'	TARGET (partner)
bit 6			DRAIN_SOURCE
		B'0'	NO
		B'1'	YES

Disp	Lng	Value	Field Name
bit 7			DRAIN_TARGET
		B'0'	NO
		B'1'	YES

CONFIRM or MC_CONFIRM

record format (AH = 2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'0300' (3)	Verb operation code
14	1		Verb extension code
		X'00'	CONFIRM
		X'01'	MC_CONFIRM
15	5	5X'00'	Reserved
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0005'	DEALLOCATE_ABEND (note 2)
		X'0006'	DEALLOCATE_ABEND_PROG (note 1)
		X'0007'	DEALLOCATE_ABEND_SVC (note 1)
		X'0008'	DEALLOCATE_ABEND_TIMER (note 1)
		X'000E'	PROG_ERROR_PURGING
		X'000F'	CONV_FAILURE_RETRY
		X'0010'	CONV_FAILURE_NO_RETRY
		X'0013'	SVC_ERROR_PURGING (note 1)
		X'0019'	CONVERSATION_TYPE_MIXED
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
		X'F004'	INCOMPLETE
22	4		Secondary RETURN_CODE
		X'00000001'	BAD_TP_ID
		X'00000002'	BAD_CONV_ID
		X'00000031'	CONFIRM_ON_SYNC_NONE
		X'00000032'	CONFIRM_BAD_STATE
		X'00000033'	CONFIRM_NOT_LL_BDY
26	8		TP_ID
34	4		CONV_ID
38	1		REQUEST_TO_SEND_RECEIVED
		X'00'	NO
		X'01'	YES

Note 1: Not applicable if MC_CONFIRM.

Note 2: Not applicable if CONFIRM.

For descriptions of the ALLOCATION_ERROR secondary return codes, see Chapter 6, "Using Transaction Mapped Conversation

Verbs" or Chapter 7, "Using Transaction Basic Conversation Verbs."

CONFIRMED or MC_CONFIRMED

record format (AH = 2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'0400' (4)	Verb operation code
14	1		Verb extension code
		X'00'	CONFIRMED
		X'01'	MC_CONFIRMED
15	5	5X'00'	Reserved
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0019'	CONVERSATION_TYPE_MIXED
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
22	4		Secondary RETURN_CODE
		X'00000001'	BAD_TP_ID
		X'00000002'	BAD_CONV_ID
		X'00000041'	CONFIRMED_BAD_STATE
26	8		TP_ID
34	4		CONV_ID

CONVERT

record format (AH = 251)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'1A00'(26)	Verb operation code
14	6	6X'00'	Reserved
20	4		RETURN_CODE
		X'00000000'	OK
		X'00000401'	INVALID_DIRECTION
		X'00000402'	INVALID_TYPE
		X'00000403'	SEGMENT_OVERLAP
		X'00000404'	INVALID_FIRST_CHARACTER
		X'00000405'	TABLE_ERROR

Disp	Lng	Value	Field Name
		X'00000406'	CONVERSION_ERROR
		X'F0030000'	APPC_ABENDED
24	1		DIRECTION
		X'00'	ASCII_TO_EBCDIC
		X'01'	EBCDIC_TO_ASCII
25	1		CHARACTER_SET
		X'00'	AE
		X'01'	A
		X'02'	G
26	2*		LENGTH
28	4*		SOURCE
32	4*		TARGET

CREATE_TP

record format

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'2300' (35)	Verb operation code
14	6	6X'00'	Reserved
20	4		SENSE_CODE
		X'00000000'	OK
		X'080F6051'	SECURITY_NOT_VALID
		X'084B6031'	TP_NOT_AVAIL_RETRY
		X'084C0000'	TP_NOT_AVAIL_NO_RETRY
		X'10086021'	TP_NAME_NOT_RECOGNIZED
		X'10086034'	CONVERSATION_TYPE_MISMATCH
		X'10086041'	SYNC_LEVEL_NOT_SUPPORTED
24	8		TP_ID
32	8		LU_ID
40	4		CONV_ID
44	1		TYPE
		X'00'	BASIC_CONVERSATION
		X'01'	MAPPED_CONVERSATION
45	1		SYNC_LEVEL
		X'00'	NONE
		X'01'	CONFIRM
46	1		Reserved
47	1		Transaction program name length
48	64		TPN
112	6	6X'00'	Reserved
118	2*		Length of ERROR_LOG_DATA to return
120	4*		Pointer to ERROR_LOG_DATA to return
124	8		PARTNER_LU_NAME
132	1		Length of fully qualified partner LU name
133	17		PARTNER_FULLY_QUALIFIED_LU_NAME
150	8		MODE_NAME
158	12	12X'00'	Reserved
170	1		Length of password

Disp	Lng	Value	Field Name
171	10		PASSWORD
181	1		Length of user ID
182	10		USER_ID
192	1		ALREADY_VERIFIED
		X'00'	NO - Verification should be performed
		X'01'	YES

DEALLOCATE or MC_DEALLOCATE

record format (AH = 2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'0500' (5)	Verb operation code
14	1		Verb extension code
		X'00'	DEALLOCATE
		X'01'	MC_DEALLOCATE
15	5	5X'00'	Reserved
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0005'	DEALLOCATE_ABEND (note 2)
		X'0006'	DEALLOCATE_ABEND_PROG (note 1)
		X'0007'	DEALLOCATE_ABEND_SVC (note 1)
		X'0008'	DEALLOCATE_ABEND_TIMER (note 1)
		X'000E'	PROG_ERROR_PURGING
		X'000F'	CONV_FAILURE_RETRY
		X'0010'	CONV_FAILURE_NO_RETRY
		X'0013'	SVC_ERROR_PURGING (note 1)
		X'0019'	CONVERSATION_TYPE_MIXED
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
		X'F004'	INCOMPLETE
22	4		Secondary RETURN_CODE
		X'00000001'	BAD_TP_ID
		X'00000002'	BAD_CONV_ID
		X'00000006'	DATA_AREA_ACROSS_SEGMENT (note 1)
		X'00000051'	DEALLOC_BAD_TYPE
		X'00000052'	DEALLOC_FLUSH_BAD_STATE
		X'00000053'	DEALLOC_CONFIRM_BAD_STATE
		X'00000055'	DEALLOC_NOT_LL_BDY (note 1)
		X'00000057'	LOG_LL_WRONG (note 1)
26	8		TP_ID
34	4		CONV_ID

Disp	Lng	Value	Field Name
38	1	X'00'	Reserved
39	1		TYPE
		X'00'	SYNC_LEVEL
		X'01'	FLUSH
		X'02'	ABEND_PROG (note 1)
		X'03'	ABEND_SVC (note 1)
		X'04'	ABEND_TIMER (note 1)
		X'05'	ABEND (note 4)
40	2*		Length of error log data (note 3)
42	4*		Address of error log data (note 3)

Note 1: Not applicable if MC_DEALLOCATE.

Note 2: Not applicable if DEALLOCATE.

Note 3: Reserved if MC_DEALLOCATE.

Note 4: Reserved if DEALLOCATE.

For descriptions of the ALLOCATION_ERROR secondary return codes, see Chapter 6, "Using Transaction Mapped Conversation Verbs" or Chapter 7, "Using Transaction Basic Conversation Verbs."

DETACH_LU

record format (AH = 1)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'2200' (34)	Verb operation code
14	6	6X'00'	Reserved
20	4		RETURN_CODE
		X'00000000'	OK
		X'00000003'	BAD_LU_ID
		X'00000223'	SSCP_CONNECTED_LU
		X'F0010000'	APPC_DISABLED
		X'F0020000'	APPC_BUSY
		X'F0030000'	APPC_ABENDED
24	8		LU_ID
32	1	X'00'	Reserved

DETACH_PU

record format (AH = 1)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'2700' (39)	Verb operation code
14	6	6X'00'	Reserved
20	4		RETURN_CODE
		X'00000000'	OK
		X'00000008'	NO_PU_ATTACHED
		X'00000272'	ADAPTER_CLOSE_FAILURE
		X'F0010000'	APPC_DISABLED
24	1		TYPE
		X'00'	HARD
		X'01'	SOFT

DISABLE/ENABLE_APPC

record format (AH = 250)

This verb passes all parameters in registers; no control structure is used.

- Disable APPC/PC: AL = 1 (note 1)
- Enable APPC/PC: AL = 0 (note 1)

Note 1: APPC/PC examines only the last bit of the AL register.

DISPLAY

record format (AH = 1)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'1B00' (27)	Verb operation code
14	6	6X'00'	Reserved
20	4		RETURN_CODE

Disp	Lng	Value	Field Name
		X'00000000'	OK
		X'00000003'	BAD_LU_ID
		X'000001B1'	BAD_PART_LUNAME
		X'000001B2'	BAD_MODENAME
		X'F0010000'	APPC_DISABLED
		X'F0020000'	APPC_BUSY
		X'F0030000'	APPC_ABENDED
24	2	X'0000'	Reserved
26	8		LU_ID
34	8		PARTNER_LU_NAME
42	8		MODE_NAME
50	1		LU_SESSION_LIMIT
51	1		PARTNER_LU_SESSION_LIMIT
52	1		MODE_MAX_NEGOTIABLE_SESSION_LIMIT
53	1		CURRENT_SESSION_LIMIT
54	1		MIN_NEGOTIATED_WINNER_LIMIT
55	1		MIN_NEGOTIATED_LOSER_LIMIT
56	1		ACTIVE_SESSION_COUNT
57	1		ACTIVE_CONWINNER_SESSION_COUNT
58	1		ACTIVE_CONLOSER_SESSION_COUNT
59	1		SESSION_TERMINATION_COUNT
60	1		Termination settings
bits 0-5		B'000000'	Reserved
bit 6			SESSION_TERMINATION_ SOURCE_DRAIN
		B'0'	NO
		B'1'	YES
bit 7			SESSION_TERMINATION_ TARGET_DRAIN
		B'0'	NO
		B'1'	YES

FLUSH or MC_FLUSH

record format (AH = 2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'0600' (6)	Verb operation code
14	1		Verb extension code
		X'00'	FLUSH
		X'01'	MC_FLUSH
15	5	5X'00'	Reserved
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0019'	CONVERSATION_TYPE_MIXED
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY

Disp	Lng	Value	Field Name
		X'F003'	APPC_ABENDED
22	4		Secondary RETURN_CODE
		X'00000001'	BAD_TP_ID
		X'00000002'	BAD_CONV_ID
		X'00000061'	FLUSH_NOT_SEND_STATE
26	8		TP_ID
34	4		CONV_ID

GET_ALLOCATE

record format (AH = 3)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'2800' (40)	Verb operation code
14	6	6X'00'	Reserved
20	4		RETURN_CODE
		X'00000000'	OK
		X'00000003'	BAD_LU_ID
		X'00000281'	GET_ALLOC_BAD_TYPE
		X'00000282'	UNSUCCESSFUL
		X'F0010000'	APPC_DISABLED
		X'F0020000'	APPC_BUSY
		X'F0030000'	APPC_ABENDED
24	2	2X'00'	Reserved
26	8		LU_ID
34	1		TYPE
		X'00'	DEQUEUE
		X'01'	TEST
35	4*		Pointer to CREATE_TP record

GET_ATTRIBUTES or MC_GET_ATTRIBUTES

record format (AH = 2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'0700' (7)	Verb operation code
14	1		Verb extension code
		X'00'	GET_ATTRIBUTES
		X'01'	MC_GET_ATTRIBUTES
15	5	5X'00'	Reserved

Disp	Lng	Value	Field Name
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0019'	CONVERSATION_TYPE_MIXED
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
22	4		Secondary RETURN_CODE
		X'00000001'	BAD_TP_ID
		X'00000002'	BAD_CONV_ID
26	8		TP_ID
34	4		CONV_ID
38	8		LU_ID
46	1	X'00'	Reserved
47	1		SYNC_LEVEL
		X'00'	NONE
		X'01'	CONFIRM
48	8		MODE_NAME
56	8		OWN_NET_NAME
64	8		OWN_LU_NAME
72	8		PARTNER_LU_NAME
80	1		Length of PARTNER_FULLY_ QUALIFIED_LU_NAME
81	17		PARTNER_FULLY_QUALIFIED_ LU_NAME
98	1	X'00'	Reserved
99	1		Length of USER_ID
100	10		USER_ID

GET_TYPE

record format (AH = 2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'0800' (8)	Verb operation code
14	6	6X'00'	Reserved
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
22	4		Secondary RETURN_CODE
		X'00000001'	BAD_TP_ID
		X'00000002'	BAD_CONV_ID
26	8		TP_ID
34	4		CONV_ID
38	1		TYPE
		X'00'	BASIC_CONVERSATION

Disp	Lng	Value	Field Name
		X'01'	MAPPED_CONVERSATION

PASSTHROUGH

record format (AH = 7)

This verb must be formatted according to the specifications of the application subsystem.

POST_ON_RECEIPT

record format (AH = 2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'0900' (9)	Verb operation code
14	6	6X'00'	Reserved
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0019'	CONVERSATION_TYPE_MIXED
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
22	4		Secondary RETURN_CODE
		X'00000001'	BAD_TP_ID
		X'00000002'	BAD_CONV_ID
		X'00000091'	INVALID_LENGTH
		X'00000092'	P_ON_R_NOT_RCV_STATE
		X'00000093'	P_ON_R_BAD_FILL
26	8		TP_ID
34	4		CONV_ID
38	2*		MAX_LENGTH
40	1		FILL
		X'00'	BUFFER
		X'01'	LL

PREPARE_TO_RECEIVE and MC_PREPARE_TO_RECEIVE

record format (AH=2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'0A00' (10)	Verb operation code
14	1		Verb extension code
		X'00'	PREPARE_TO_RECEIVE
		X'01'	MC_PREPARE_TO_RECEIVE
15	5	5X'00'	Reserved
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0005'	DEALLOCATE_ABEND (note 2)
		X'0006'	DEALLOCATE_ABEND_PROG (note 1)
		X'0007'	DEALLOCATE_ABEND_SVC (note 1)
		X'0008'	DEALLOCATE_ABEND_TIMER (note 1)
		X'000E'	PROG_ERROR_PURGING
		X'000F'	CONV_FAILURE_RETRY
		X'0010'	CONV_FAILURE_NO_RETRY
		X'0013'	SVC_ERROR_PURGING (note 1)
		X'0019'	CONVERSATION_TYPE_MIXED
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
		X'F004'	INCOMPLETE
22	4		Secondary RETURN_CODE
		X'00000001'	BAD_TP_ID
		X'00000002'	BAD_CONV_ID
		X'000000A1'	P_TO_R_INVALID_TYPE
		X'000000A2'	UNFINISHED_LL (note 1)
		X'000000A3'	P_TO_R_NOT_SEND_STATE
26	8		TP_ID
34	4		CONV_ID
38	1		TYPE
		X'00'	SYNC_LEVEL
		X'01'	FLUSH
39	1		LOCKS
		X'00'	SHORT
		X'01'	LONG

Note 1: Not applicable if MC_PREPARE_TO_RECEIVE.

Note 2: Not applicable if PREPARE_TO_RECEIVE.

For descriptions of the ALLOCATION_ERROR secondary return codes, see Chapter 6, "Using Transaction Mapped Conversation

Verbs" or Chapter 7, "Using Transaction Basic Conversation Verbs."

RECEIVE_AND_WAIT and MC_RECEIVE_AND_WAIT

record format (AH = 2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'0B00' (11)	Verb operation code
14	1		Verb extension code
		X'00'	RECEIVE_AND_WAIT
		X'01'	MC_RECEIVE_AND_WAIT
15	5	5X'00'	Reserved
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0005'	DEALLOCATE_ABEND (note 3)
		X'0006'	DEALLOCATE_ABEND_PROG (note 1)
		X'0007'	DEALLOCATE_ABEND_SVC (note 1)
		X'0008'	DEALLOCATE_ABEND_TIMER (note 1)
		X'0009'	DEALLOCATE_NORMAL
		X'000C'	PROG_ERROR_NO_TRUNC
		X'000D'	PROG_ERROR_TRUNC (note 1)
		X'000E'	PROG_ERROR_PURGING
		X'000F'	CONV_FAILURE_RETRY
		X'0010'	CONV_FAILURE_NO_RETRY
		X'0011'	SVC_ERROR_NO_TRUNC (note 1)
		X'0012'	SVC_ERROR_TRUNC (note 1)
		X'0013'	SVC_ERROR_PURGING (note 1)
		X'0019'	CONVERSATION_TYPE_MIXED
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
		X'F004'	INCOMPLETE
22	4		Secondary RETURN_CODE
		X'00000001'	BAD_TP_ID
		X'00000002'	BAD_CONV_ID
		X'00000006'	DATA_AREA_ACROSS_SEGMENT
		X'000000B1'	RCV_AND_WAIT_BAD_STATE
		X'000000B2'	RCV_AND_WAIT_NOT_LL_BDY (note 1)
		X'000000B5'	RCV_AND_WAIT_BAD_FILL (note 1)
26	8		TP_ID
34	4		CONV_ID
38	1		WHAT_RECEIVED
		X'00'	DATA (note 1)

Disp	Lng	Value	Field Name
		X'01'	DATA_COMPLETE
		X'02'	DATA_INCOMPLETE
		X'03'	CONFIRM
		X'04'	CONFIRM_SEND
		X'05'	CONFIRM_DEALLOCATE
		X'06'	SEND
39	1		FILL (note 2)
		X'00'	BUFFER
		X'01'	LL
40	1		REQUEST_TO_SEND_RECEIVED
		X'00'	NO
		X'01'	YES
41	2*		MAX_LENGTH
43	2*		DATA_LENGTH
45	4*		DATA_PTR

Note 1: Not applicable if MC_RECEIVE_AND_WAIT.

Note 2: Reserved if MC_RECEIVE_AND_WAIT.

Note 3: Not applicable if RECEIVE_AND_WAIT.

For descriptions of the ALLOCATION_ERROR secondary return codes, see Chapter 6, "Using Transaction Mapped Conversation Verbs" or Chapter 7, "Using Transaction Basic Conversation Verbs."

RECEIVE_IMMEDIATE and MC_RECEIVE_IMMEDIATE

record format (AH = 2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'0C00' (12)	Verb operation code
14	1		Verb extension code
		X'00'	RECEIVE_IMMEDIATE
		X'01'	MC_RECEIVE_IMMEDIATE
15	5	5X'00'	Reserved
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0005'	DEALLOCATE_ABEND (note 3)

Disp	Lng	Value	Field Name
		X'0006'	DEALLOCATE_ABEND_PROG (note 1)
		X'0007'	DEALLOCATE_ABEND_SVC (note 1)
		X'0008'	DEALLOCATE_ABEND_TIMER (note 1)
		X'0009'	DEALLOCATE_NORMAL
		X'000C'	PROG_ERROR_NO_TRUNC
		X'000D'	PROG_ERROR_TRUNC (note 1)
		X'000E'	PROG_ERROR_PURGING
		X'000F'	CONV_FAILURE_RETRY
		X'0010'	CONV_FAILURE_NO_RETRY
		X'0011'	SVC_ERROR_NO_TRUNC (note 1)
		X'0012'	SVC_ERROR_TRUNC (note 1)
		X'0013'	SVC_ERROR_PURGING (note 1)
		X'0014'	UNSUCCESSFUL
		X'0019'	CONVERSATION_TYPE_MIXED
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
		X'F004'	INCOMPLETE
22	4		Secondary RETURN_CODE
		X'00000001'	BAD_TP_ID
		X'00000002'	BAD_CONV_ID
		X'00000006'	DATA_AREA_ACROSS_SEGMENT
		X'000000C1'	RCV_IMMD_NOT_RCV_STATE
		X'000000C4'	RCV_IMMD_BAD_FILL (note 1)
26	8		TP_ID
34	4		CONV_ID
38	1		WHAT_RECEIVED
		X'00'	DATA (note 1)
		X'01'	DATA_COMPLETE
		X'02'	DATA_INCOMPLETE
		X'03'	CONFIRM
		X'04'	CONFIRM_SEND
		X'05'	CONFIRM_DEALLOCATE
		X'06'	SEND
39	1		FILL (note 2)
		X'00'	BUFFER
		X'01'	LL
40	1		REQUEST_TO_SEND_RECEIVED
		X'00'	NO
		X'01'	YES
41	2*		MAX_LENGTH
43	2*		DATA_LENGTH
45	4*		DATA_PTR

Note 1: Not applicable if MC_RECEIVE_IMMEDIATE.

Note 2: Reserved if MC_RECEIVE_IMMEDIATE.

Note 3: Not applicable if RECEIVE_IMMEDIATE.

For descriptions of the ALLOCATION_ERROR secondary return codes, see Chapter 6, "Using Transaction Mapped Conversation Verbs" or Chapter 7, "Using Transaction Basic Conversation Verbs."

REQUEST_TO_SEND and MC_REQUEST_TO_SEND

record format (AH=2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2'	X'0E00' (14)	Verb operation code
14	1		Verb extension code
		X'00'	REQUEST_TO_SEND
		X'01'	MC_REQUEST_TO_SEND
15	5	5X'00'	Reserved
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0019'	CONVERSATION_TYPE_MIXED
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
		X'F004'	INCOMPLETE
22	4		Secondary RETURN_CODE
		X'00000001'	BAD_TP_ID
		X'00000002'	BAD_CONV_ID
		X'000000E1'	R_T_S_NOT_RCV_STATE
26	8		TP_ID
34	4		CONV_ID

SEND_DATA or MC_SEND_DATA

record format (AH=2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'0F00' (15)	Verb operation code
14	1		Verb extension code
		X'00'	SEND_DATA
		X'01'	MC_SEND_DATA
15	5	5X'00'	Reserved
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0005'	DEALLOCATE_ABEND (note 2)

Disp	Lng	Value	Field Name
		X'0006'	DEALLOCATE_ABEND_PROG (note 1)
		X'0007'	DEALLOCATE_ABEND_SVC (note 1)
		X'0008'	DEALLOCATE_ABEND_TIMER (note 1)
		X'000E'	PROG_ERROR_PURGING
		X'000F'	CONV_FAILURE_RETRY
		X'0010'	CONV_FAILURE_NO_RETRY
		X'0013'	SVC_ERROR_PURGING (note 1)
		X'0019'	CONVERSATION_TYPE_MIXED
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
		X'F004'	INCOMPLETE
22	4		Secondary RETURN_CODE
		X'00000001'	BAD_TP_ID
		X'00000002'	BAD_CONV_ID
		X'00000006'	DATA_AREA_ACROSS_SEGMENT
		X'000000F1'	BAD_LL (note 1)
		X'000000F2'	SEND_DATA_NOT_SEND_STATE
26	8		TP_ID
34	4		CONV_ID
38	1		REQUEST_TO_SEND_RECEIVED
		X'00'	NO
		X'01'	YES
39	1	X'00'	Reserved
40	2*		DATA_LENGTH
42	4*		DATA_PTR

Note 1: Not applicable if MC_SEND_DATA.

Note 2: Not applicable if SEND_DATA.

SEND_ERROR or MC_SEND_ERROR

record format (AH = 2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'1000' (16)	Verb operation code
14	1		Verb extension code
		X'00'	SEND_ERROR
		X'01'	MC_SEND_ERROR
15	5	5X'00'	Reserved
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0003'	ALLOCATION_ERROR

Disp	Lng	Value	Field Name
		X'0005'	DEALLOCATE_ABEND (note 3)
		X'0006'	DEALLOCATE_ABEND_PROG (note 2)
		X'0007'	DEALLOCATE_ABEND_SVC (note 2)
		X'0008'	DEALLOCATE_ABEND_TIMER (note 2)
		X'0009'	DEALLOCATE_NORMAL
		X'000E'	PROG_ERROR_PURGING
		X'000F'	CONV_FAILURE_RETRY
		X'0010'	CONV_FAILURE_NO_RETRY
		X'0013'	SVC_ERROR_PURGING (note 2)
		X'0019'	CONVERSATION_TYPE_MIXED
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
		X'F004'	INCOMPLETE
22	4		Secondary RETURN_CODE
		X'00000001'	BAD_TP_ID
		X'00000002'	BAD_CONV_ID
		X'00000006'	DATA_AREA_ACROSS_SEGMENT (note 2)
		X'00000102'	LOG_LL_WRONG (note 2)
		X'00000103'	BAD_TYPE (note 2)
26	8		TP_ID
34	4		CONV_ID
38	1		REQUEST_TO_SEND_RECEIVED
		X'00'	NO
		X'01'	YES
39	1		TYPE (note 1)
		X'00'	PROG
		X'01'	SVC
40	4	4X'00'	Reserved
44	2*		LOG_DATA_LENGTH (note 1)
46	4*		LOG_DATA (note 1)

Note 1: Reserved if MC_SEND_ERROR.

Note 2: Not applicable if MC_SEND_ERROR.

Note 3: Not applicable if SEND_ERROR.

For descriptions of the ALLOCATION_ERROR secondary return codes, see Chapter 6, "Using Transaction Mapped Conversation Verbs" or Chapter 7, "Using Transaction Basic Conversation Verbs."

SET_PASSTHROUGH

format (AH = 255)

This verb passes all parameters in registers; APPC/PC does not require a control structure block.

DS:DX contains the passthrough exit address.

SYSLOG

record format

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'2600' (38)	Verb operation code
14	10	10X'00'	Reserved
24	2		TYPE See Appendix D, "SYSLOG Type Codes"
26	4		SUBTYPE See Appendix D, "SYSLOG Type Codes"
30	4*		Pointer to ADDITIONAL_INFO
34	4		CONV_ID (0 if unknown)
38	8		TP_ID (0 if unknown)
46	8		PU_OR_LU_NAME (0 if unknown)
54	2*		DATA_LENGTH
56	4*		DATA_ADDRESS
60	1	X'00'	Reserved

TEST or MC_TEST

record format (AH = 2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'1200' (18)	Verb operation code
14	1		Verb extension code
		X'00'	TEST
		X'01'	MC_TEST
15	5	5X'00'	Reserved

Disp	Lng	Value	Field Name
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK (note 1)
		X'0003'	ALLOCATION_ERROR (note 1)
		X'0006'	DEALLOCATE_ABEND_PROG (note 1)
		X'0007'	DEALLOCATE_ABEND_SVC (note 1)
		X'0008'	DEALLOCATE_ABEND_TIMER (note 1)
		X'0009'	DEALLOCATE_NORMAL (note 1)
		X'000A'	DATA_POSTING_BLOCKED (note 1)
		X'000B'	POSTING_NOT_ACTIVE
		X'000C'	PROG_ERROR_NO_TRUNC (note 1)
		X'000D'	PROG_ERROR_TRUNC (note 1)
		X'000E'	PROG_ERROR_PURGING (note 1)
		X'000F'	CONV_FAILURE_RETRY (note 1)
		X'0010'	CONV_FAILURE_NO_RETRY (note 1)
		X'0011'	SVC_ERROR_NO_TRUNC (note 1)
		X'0012'	SVC_ERROR_TRUNC (note 1)
		X'0013'	SVC_ERROR_PURGING (note 1)
		X'0014'	UNSUCCESSFUL
		X'0019'	CONVERSATION_TYPE_MIXED
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
		X'F004'	INCOMPLETE
22	4		Secondary RETURN_CODE
		X'00000000'	POSTED_DATA (note 1)
		X'00000001'	POSTED_NOT_DATA (primary RETURN_CODE - X'0000')
		X'00000001'	BAD_TP_ID (primary RETURN_CODE - X'0001')
		X'00000002'	BAD_CONV_ID
		X'00000121'	TEST_INVALID_TYPE
		X'00000122'	NOT_RCV_STATE (note 1)
26	8		TP_ID
34	4		CONV_ID
38	1		TEST (note 2)
		X'00'	POSTED
		X'01'	REQUEST_TO_SEND_RECEIVED

Note 1: Not applicable if MC_TEST.

Note 2: Reserved if MC_TEST.

For descriptions of the ALLOCATION_ERROR secondary return codes, see Chapter 6, "Using Transaction Mapped Conversation Verbs" or Chapter 7, "Using Transaction Basic Conversation Verbs."

TP_ENDED

record format (AH = 4)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'2500' (37)	Verb operation code
14	6	6X'00'	Reserved
20	4		RETURN_CODE
		X'00000000'	OK
		X'00000001'	BAD_TP_ID
		X'F0010000'	APPC_DISABLED
		X'F0020000'	APPC_BUSY
		X'F0030000'	APPC_ABENDED
		X'F0040000'	INCOMPLETE
24	2	2X'00'	Reserved
26	8		TP_ID

TP_STARTED

record format (AH = 3)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'2400' (36)	Verb operation code
14	6	6X'00'	Reserved
20	4		RETURN_CODE
		X'00000000'	OK
		X'00000003'	BAD_LU_ID
		X'00000243'	TOO_MANY_TPS
		X'F0010000'	APPC_DISABLED
		X'F0020000'	APPC_BUSY
		X'F0030000'	APPC_ABENDED
24	2	2X'00'	Reserved
26	8		LU_ID
34	8		TP_ID

TP_VALID

record format (AH = 4)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'2900' (41)	Verb operation code
14	6	6X'00'	Reserved
20	4		RETURN_CODE
		X'00000000'	OK
		X'00000001'	BAD_TP_ID
		X'00000002'	BAD_CONV_ID
		X'00000110'	BAD_STATE
		X'F0010000'	APPC_DISABLED
		X'F0020000'	APPC_BUSY
		X'F0030000'	APPC_ABENDED
		X'F0040000'	INCOMPLETE
24	2	2X'00'	Reserved
26	8		TP_ID
34	4*		CREATE_TP_PTR

TRACE

formats

This verb passes all parameters in registers; APPC/PC does not require a control structure block.

- TRACE_MESSAGES (AH = 252)
  - AL = 0: Trace OFF (disable message tracing)
  - AL = 1: Trace ON (enable message tracing)

    DX = n: Number of bytes at which to truncate RU
    DX = 0: Indicates no truncation
- TRACE_API (AH = 253)
  - AL = 1: Trace ON (enable API verb trace)
  - AL = 0: Trace OFF (disable API verb trace).
- TRACE_DESTINATION (AH = 254)

Combinations of the following values cause APPC/PC to direct the trace statistics to multiple destinations. For example, AL=6 directs the trace output to the display of the IBM PC and to a file in the current directory.

- AL=1: STORAGE

  DS:DX = user-provided address of TRACE_STATS

- AL=2: DISPLAY

- AL=4: FILE

  OUTPUT.PC = file containing the trace
  DS:DX = user-provided address of TRACE_STATS

- AL=8: PRINTER

TRACE_STATS formats

Record format for TRACE_STATS, pointed to by DS :DX.

Disp	Lng	Description
0	4*	Start address of storage buffer (needed for storage trace only)
4	2*	Maximum number of records to retain (file or buffer will wrap after this many 80-byte records)
6	2	Current record number (must be initialized to 0)
8	4	Number of records written (must be initialized to 0)
12	4	Reserved

Note: The TRACE_STATS record must remain at the specified address while tracing is active. When tracing to STORAGE or FILE, APPC/PC updates the record number (displacement 6) and number of records written (displacement 8) each time it makes a trace entry.

TRANSFER_MS_DATA

record format (AH=5)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'1C00' (28)	Verb operation code
14	1		DATA_TYPE
		X'00'	USER_DEFINED
		X'01'	NMVT
		X'02'	ALERT_SUBVECTORS
		X'03'	PDSTATS_SUBVECTORS
15	5	5X'00'	Reserved
20	4		RETURN_CODE
		X'00000000'	OK
		X'00000301'	SSCP_PU_SESSION_NOT_ACTIVE
		X'00000302'	DATA_EXCEEDS_RU_SIZE
		X'F0010000'	APPC_DISABLED
		X'F0020000'	APPC_BUSY
		X'F0030000'	APPC_ABENDED
24	12	12X'00'	Reserved
36	1		Verb options
bit 0			CORRELATOR_SUBVECTOR
		B'0'	ADD
		B'1'	NO_ADD
bit 1			PRODUCT_SET_ID_SUBVECTOR
		B'0'	ADD
		B'1'	NO_ADD
bit 2			SYSLOG
		B'0'	LOG
		B'1'	NO_LOG
bit 3			SSCP_PU_SESSION
		B'0'	SEND
		B'1'	NO_SEND
bits 4-7		B'0000'	Reserved
37	1	X'00'	Reserved
38	2*		DATA_LENGTH
40	n		DATA (note 1)

Note 1: For information on management services data, refer to the *SNA Format and Protocol Reference Manual* under Management Services and under User-Defined Data.

WAIT

record format (AH = 2)

Disp	Lng	Value	Field Name
0	12	**	Required APPC/PC area
12	2*	X'1300' (19)	Verb operation code
14	6	6X'00'	Reserved
20	2		Primary RETURN_CODE
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0006'	DEALLOCATE_ABEND_PROG
		X'0007'	DEALLOCATE_ABEND_SVC
		X'0008'	DEALLOCATE_ABEND_TIMER
		X'0009'	DEALLOCATE_NORMAL
		X'000A'	DATA_POSTING_BLOCKED
		X'000B'	POSTING_NOT_ACTIVE
		X'000C'	PROG_ERROR_NO_TRUNC
		X'000D'	PROG_ERROR_TRUNC
		X'000E'	PROG_ERROR_PURGING
		X'000F'	CONV_FAILURE_RETRY
		X'0010'	CONV_FAILURE_NO_RETRY
		X'0011'	SVC_ERROR_NO_TRUNC
		X'0012'	SVC_ERROR_TRUNC
		X'0013'	SVC_ERROR_PURGING
		X'0019'	CONVERSATION_TYPE_MIXED
		X'F001'	APPC_DISABLED
		X'F002'	APPC_BUSY
		X'F003'	APPC_ABENDED
		X'F004'	INCOMPLETE
22	4		Secondary RETURN_CODE
		X'00000000'	POSTED_DATA
		X'00000001'	POSTED_NOT_DATA (primary RETURN_CODE - X'0000')
		X'00000001'	BAD_TP_ID (primary RETURN_CODE - X'0001')
		X'00000002'	BAD_CONV_ID
		X'00000122'	NOT_RCV_STATE
26	8		TP_ID
34	4		CONV_POSTED
38	1		Number of conversations to wait on
39	4*n		CONV_ID_LIST

For descriptions of the ALLOCATION_ERROR secondary return codes, see Chapter 6, "Using Transaction Mapped Conversation Verbs," or Chapter 7, "Using Transaction Basic Conversation Verbs."

Appendix B. Conversation State Matrices

This appendix shows the conversation state transitions that can occur when a program issues a conversation verb.

The three causes of state transitions are:

- Verbs the local program issues
- Verbs the remote program issues
- Network errors.

APPC/PC indicates a state change caused by the remote program or a network error by reporting a return code other than OK to the local program. For the RECEIVE_AND_WAIT and RECEIVE_IMMEDIATE verbs, APPC/PC can also indicate a state change on the WHAT_RECEIVED parameter.

The following figures illustrate the possible conversation state transitions in a matrix format. The columns of the matrix show the states, and the rows show the verbs. A verb appears more than once if there is more than one way for that verb to cause a state change.

The verbs listed in the matrix apply to the basic conversation API as well as the mapped conversation API, unless noted otherwise by the "not MC" indication next to the verb name.

APPC/PC defines the following conversation states:

Reset is the state in which the program can allocate the conversation.

Send is the state in which the program can send data, request confirmation, or deallocate a conversation normally.

Receive is the state in which the program can receive information from the remote program.

Confirm, Confirm Send, and Confirm Deallocate are the states in which the program can reply to a confirmation request. (Chapters 1 through 10 refer to this group of states as "confirm state.")

These matrices use abbreviations for the parameters, return codes, and WHAT_RECEIVED indications. The definitions for the abbreviations and symbols used in the state-transition matrix appear at the bottom of the following figure.

Several return codes (for example, PARAMETER_CHECK, STATE_CHECK, and CONVERSATION_TYPE_MIXED) are valid return codes for almost all verbs but they do not cause a state change. Therefore, the matrices do not list these errors separately for each verb.

Verb	Verb Conversation States					
	Reset	Send	Re-ceive	Con-firm	Con-firm Send	Con-firm De-allo-cate
	1	2	3	4	5	6
Conv Initiated by Partner	3	/	/	/	/	/
ALLOCATE[ok]	2	/	/	/	/	/
ALLOCATE[er]	-	/	/	/	/	/
CONFIRM[ok]	/	-	/	/	/	/
CONFIRM[er]	/	1	/	/	/	/
CONFIRM[ep]	/	3	/	/	/	/
CONFIRMED	/	/	/	3	2	1
DEALLOCATE(F)[ok]	/	1	/	/	/	/
DEALLOCATE(C)[ok]	/	1	/	/	/	/
DEALLOCATE(A)[ok]	/	1	1	1	1	1
DEALLOCATE(C)[er]	/	1	/	/	/	/
DEALLOCATE(C)[ep]	/	3	/	/	/	/
FLUSH	/	-	/	/	/	/
GET_ATTRIBUTES	/	-	-	-	-	-
GET_TYPE	/	-	-	-	-	-
POST_ON_RECEIPT (not MC)	/	/	-	/	/	/
PREPARE_TO_RECEIVE(F)[ok]	/	3	/	/	/	/
PREPARE_TO_RECEIVE(C)[ok]	/	3	/	/	/	/
PREPARE_TO_RECEIVE(C)[er]	/	1	/	/	/	/
PREPARE_TO_RECEIVE(C)[ep]	/	3	/	/	/	/
RECEIVE_AND_WAIT[ok]{da}	/	3	-	/	/	/
RECEIVE_AND_WAIT[ok]{se}	/	-	2	/	/	/
RECEIVE_AND_WAIT[ok]{co}	/	4	4	/	/	/
RECEIVE_AND_WAIT[ok]{cs}	/	5	5	/	/	/
RECEIVE_AND_WAIT[ok]{cd}	/	6	6	/	/	/
RECEIVE_AND_WAIT[dn]	/	/	1	/	/	/
RECEIVE_AND_WAIT[da]	/	1	1	/	/	/
RECEIVE_AND_WAIT[er]	/	1	1	/	/	/
RECEIVE_AND_WAIT[ep]	/	3	-	/	/	/
RECEIVE_AND_WAIT[et]	/	/	-	/	/	/
RECEIVE_IMMEDIATE[ok]{da}	/	/	-	/	/	/
RECEIVE_IMMEDIATE[ok]{se}	/	/	2	/	/	/
RECEIVE_IMMEDIATE[ok]{co}	/	/	4	/	/	/
RECEIVE_IMMEDIATE[ok]{cs}	/	/	5	/	/	/
RECEIVE_IMMEDIATE[ok]{cd}	/	/	6	/	/	/
RECEIVE_IMMEDIATE[dn]	/	/	1	/	/	/
RECEIVE_IMMEDIATE[da]	/	/	1	/	/	/
RECEIVE_IMMEDIATE[er]	/	/	1	/	/	/
RECEIVE_IMMEDIATE[ep]	/	/	-	/	/	/
RECEIVE_IMMEDIATE[et]	/	/	-	/	/	/

Verb	Verb Conversation States					
	Reset	Send	Receive	Confirm	Confirm Send	Confirm Deallocate
	1	2	3	4	5	6
RECEIVE_IMMEDIATE[un]	/	/	-	/	/	/
REQUEST_TO_SEND	/	/	-	-	-	-
SEND_DATA[ok]	/	-	/	/	/	/
SEND_DATA[er]	/	1	/	/	/	/
SEND_DATA[ep]	/	3	/	/	/	/
SEND_ERROR[ok]	/	-	2	2	2	2
SEND_ERROR[er]	/	1	1	1	1	1
SEND_ERROR[dn]	/	/	1	/	/	/
SEND_ERROR[ep]	/	3	/	/	/	/
TEST(P)[ok]	/	/	-	/	/	/
TEST(P)[un]	/	/	-	/	/	/
TEST(P)[er]	/	/	1	/	/	/
TEST(P)[ep]	/	/	-	/	/	/
TEST(P)[et]	/	/	-	/	/	/
TEST(P)[dn]	/	/	1	/	/	/
TEST(P)[da]	/	/	1	/	/	/
TEST(Q)[ok]	/	-	-	/	/	/
TEST(Q)[un]	/	-	-	/	/	/
WAIT[ok] (Not MC)	/	/	-	/	/	/
WAIT[er]	/	/	1	/	/	/
WAIT[ep]	/	/	-	/	/	/
WAIT[et]	/	/	-	/	/	/
WAIT[dn]	/	/	1	/	/	/
WAIT[da]	/	/	1	/	/	/

Conversation State Transition Matrix Abbreviations

Parameter Abbreviations (...)

A	TYPE(ABEND_PROG), TYPE(ABEND_SVC), TYPE(ABEND_TIMER) if basic conversation, or TYPE(ABEND) if mapped conversation
C	TYPE(SYNC_LEVEL) with synchronization level CONFIRM
F	TYPE(FLUSH)
P	TEST(POSTED)
Q	TEST(REQUEST_TO_SEND_RECEIVED)

Return-Code Abbreviations [...]

er	Error on link or when allocating a conversation to a session or resource failure
ep	Partner issued a SEND_ERROR, data purged or no truncation occurred
et	Partner issued a SEND_ERROR, data truncated
dn	DEALLOCATE issued by partner
un	UNSUCCESSFUL
ok	Successful completion of command

Return-Code Abbreviations, continued

da	DEALLOCATE_ABEND issued by partner

What-Received Abbreviations {...}

co	CONFIRM issued by partner
cd	Confirm Deallocate (partner issued DEALLOCATE using Confirm)
cs	Confirm Send (partner issued PREPARE_ TO_RECEIVE using CONFIRM)
da	DATA received
se	SEND (partner issued some form of receive)

Matrix Symbols

/	Verb cannot be issued in this state
-	Remains in current state
number	Number of next state

Appendix C. Verb Return Codes

This appendix describes the return codes that APPC/PC can report to a program through the RETURN_CODE parameter of each verb. The descriptions include the hexadecimal return code value, an explanation of the condition causing the return code, and the corrective actions required if the return code indicates an error condition.

This appendix includes two sections. The first section describes the return codes that are common to most verbs. The second section describes the return codes that are unique to each verb and lists the common return codes described in the first section. These common return codes are:

- ALLOCATION_ERROR
- APPC_ABENDED
- APPC_BUSY
- APPC_DISABLED
- CONVERSATION_TYPE_MIXED
- CONV_FAILURE_NO_RETRY
- CONV_FAILURE_RETRY
- DATA_POSTING_BLOCKED
- DEALLOCATE_ABEND
- DEALLOCATE_ABEND_PROG
- DEALLOCATE_ABEND_SVC
- DEALLOCATE_ABEND_TIMER
- DEALLOCATE_NORMAL
- INCOMPLETE
- INCOMPLETE_ALTERED_VERB
- INVALID_VERB
- OK
- POSTING_NOT_ACTIVE
- PROG_ERROR_NO_TRUNC
- PROG_ERROR_PURGING
- PROG_ERROR_TRUNC
- SVC_ERROR_NO_TRUNC
- SVC_ERROR_PURGING
- SVC_ERROR_TRUNC.

The INVALID_VERB return code indicates a problem in recognizing a verb.

If the problem is due to an incorrectly issued verb, APPC/PC cannot identify which verb was intended. Therefore, this return code is not listed with each verb.

Conversation verbs and the CNOS verb return 2-byte primary return codes and 4-byte secondary return codes. Control verbs (except CNOS) return 4-byte return codes only. In this appendix, bullets indicate primary return codes and dashes indicate secondary return codes.

Common Return Codes

The following list describes the return codes that are common to many verbs.

- X'00000000' or X'0000' OK

  Description: APPC/PC executed the verb as specified.

  Action: No action is required.

- X'0003' ALLOCATION_ERROR

  Description: APPC/PC could not properly allocate a conversation.

  Action: Check the parameters of the ALLOCATE or MC_ALLOCATE verb.

  The ALLOCATION_ERROR secondary return codes are:

  - X'00000004' ALLOCATION_FAILURE_NO_RETRY

    Description: APPC/PC cannot allocate the conversation on a session because of a permanent condition. For example, APPC/PC cannot activate the session to be used for the conversation because the current mode session limit for the specified partner LU is 0; or because of a system definition error or a session-activation protocol error before it could allocate the conversation.

The program should not try the conversation again until the condition is corrected. APPC/PC reports this return code on the ALLOCATE or MC_ALLOCATE requesting the conversation.

Action: For the IBM Token-Ring Network adapter, check that the adapter number setting in the IBM Token-Ring configuration menu matches the switch setting on the adapter card and the partner LU address specified on the ATTACH_LU verb. For SDLC, check that any modems are properly plugged. Check that the station role in the SDLC configuration menu is properly set at primary, secondary, or negotiable, and that this corresponds to the settings at the partner node. If host-connected, the PC must be secondary. Check that the NRZI setting in the SDLC configuration menu matches the NRZI setting of the partner node. Restart APPC/PC by unloading and then reloading it.

– X'00000005' ALLOCATION_FAILURE_RETRY

Description: APPC/PC cannot allocate the conversation on a session because of a temporary condition. For example, APPC/PC cannot activate the session to be used for the conversation because of a temporary lack of resources at the local LU or because APPC/PC deactivated the session in response to a line or modem failure before it could allocate the conversation. The program can try the conversation again. APPC/PC reports this return code on the ALLOCATE or MC_ALLOCATE requesting the conversation.

Action: For the IBM Token-Ring Network adapter, check that the adapter number setting in the IBM Token-Ring configuration menu matches the switch setting on the adapter card and the partner LU address specified on the ATTACH_LU verb. For SDLC,

check that any modems are properly plugged. Check that the station role in the SDLC configuration menu is properly set at primary, secondary, or negotiable, and that this corresponds to the settings at the partner node. If host-connected, an IBM PC must be secondary. Check that the NRZI setting in the SDLC configuration menu matches the NRZI setting of the partner node.

Reissue the ALLOCATE or MC_ALLOCATE verb. However, to avoid congesting the network with attempted allocation requests, the program should pause or wait for a keystroke before retrying the conversation.

- X'10086034' CONVERSATION_TYPE_MISMATCH

Description: The remote LU rejected the allocation request because it or the remote transaction program does not support the specified conversation type. APPC/PC reports this return code on a verb issued after the ALLOCATE or MC_ALLOCATE verb.

Action: Change the transaction program so that it uses the other conversation type (basic or mapped).

- X'10086031' PIP_NOT_ALLOWED

Description: The remote LU rejected the allocation request because the local transaction program specified program initialization parameters and either the remote LU does not support PIP data or the remote transaction program has no PIP variables defined. APPC/PC reports this return code on a verb issued after the ALLOCATE or MC_ALLOCATE verb.

Action: Do not use PIP data.

- X'10086032' PIP_NOT_SPECIFIED_CORRECTLY

  Description: The remote LU rejected the allocation request because the remote transaction program has one or more PIP variables defined and the local transaction program has specified that no PIP variables are to be used. This secondary return code can also indicate that the number of PIP variables defined by the local transaction program is different than the number specified by the remote transaction program. APPC/PC reports this return code on a verb issued after the ALLOCATE or MC_ALLOCATE verb.

  Action: Specify that PIP data is to be used with the PIP(YES) parameter on the MC_ALLOCATE or ALLOCATE verb and make sure that the number of PIP variables agrees with the number defined for the remote transaction program.

- X'080F6051' SECURITY_NOT_VALID

  Description: The remote LU rejected the user ID or password supplied on the ALLOCATE or MC_ALLOCATE verb. APPC/PC reports this return code on a verb issued after the ALLOCATE or MC_ALLOCATE verb.

  Action: Request the operator to re-enter the user ID and password.

- X'10086041' SYNC_LEVEL_NOT_SUPPORTED

  Description: The remote program rejected the allocation request because the local transaction program specified an unrecognized or unacceptable SYNC_LEVEL type. APPC/PC reports this return code on a verb issued after the ALLOCATE or MC_ALLOCATE verb.

  Action: Change the SYNC_LEVEL specification.

- X'10086021' TPN_NOT_RECOGNIZED

  Description: The remote LU rejected the allocation request because the local transaction program specified a remote transaction program name that the remote LU does not recognize. This return code can also indicate that the remote LU recognized the TPN but could not initiate the TPN using the designated partner LU or mode name.

  Action: Check the validity of the program name and the designated partner LU and mode names.

- X'084C0000' TRANS_PGM_NOT_AVAIL_NO_RETRY

  Description: The remote LU rejected the allocation request because it cannot start the specified transaction program. APPC/PC reports this return code on a verb issued after the ALLOCATE or MC_ALLOCATE verb.

  Action: Do not retry the allocation request. Consult the system operator.

- X'084B6031' TRANS_PGM_NOT_AVAIL_RETRY

  Description: The remote LU rejected the allocation request because it cannot start the specified transaction program immediately. APPC/PC reports this return code on a verb issued after the ALLOCATE or MC_ALLOCATE verb.

  Action: Retry the allocation request. However, to avoid congesting the network with attempted allocation requests, the program should pause or wait for a keystroke before retrying the conversation.

- X'F0010000' or X'F001' APPC_DISABLED

  Description: APPC/PC did not execute the verb because APPC/PC was disabled by the DISABLE/ENABLE_APPC verb.

Action: Issue the DISABLE/ENABLE_APPC verb to re-enable APPC/PC.

- X'F0020000' or X'F002' APPC_BUSY

  Description: APPC/PC did not execute the verb because APPC/PC was processing another verb. APPC/PC can process only one verb at a time. This error can occur if a hardware interrupt during APPC/PC processing calls a routine that issues an APPC/PC verb.

  Action: Wait until APPC/PC is not busy and then reissue the verb.

- X'F0030000' or X'F003' APPC_ABENDED

  Description: APPC/PC did not execute the verb because APPC/PC has abnormally terminated.

  Action: Restart APPC/PC by unloading and then reloading it.

- X'0019' CONVERSATION_TYPE_MIXED

  Description: The program issued both basic and mapped conversation verbs on the same conversation.

  Action: Change the transaction program so that it issues only one type of conversation verb on this conversation.

- X'0010' CONV_FAILURE_NO_RETRY

  Description: A permanent failure prematurely terminated the conversation.

  Action: The condition is not temporary; operator intervention is necessary to correct the problem.

- X'000F' CONV_FAILURE_RETRY

  Description: A temporary failure prematurely terminated the conversation.

  Action: Allocate conversation again.

- X'000A' DATA_POSTING_BLOCKED

  Description: APPC/PC cannot post one of the active conversations because the APPC/PC internal workspace storage is in use and the program is unable to send a pacing response. The condition is temporary. APPC/PC can report this return code on the WAIT and TEST verbs.

  Action: The transaction program can still issue any verb but it should issue RECEIVE_IMMEDIATE or RECEIVE_AND_WAIT to free some space so that APPC/PC can resume posting active conversations.

  Alternatively, you can reconfigure APPC/PC to provide a larger MAX_RU_SIZE, a larger receive pacing window size, or no pacing at all. If you use receive pacing and issue a TEST or WAIT for more than R(P-1) bytes of data (where P = pacing window and R = negotiated maximum RU size for the session), APPC/PC may report a buffer full condition with this return code. Even this amount of data may be too large if the partner transaction program issues the FLUSH verb between SEND_DATA verbs causing messages of less than the maximum RU size to flow on the line. If pacing is not used, APPC/PC cannot control the flow of data sent by the partner transaction program. If its workspace storage becomes full APPC/PC abnormally terminates and rejects subsequent verbs.

- X'0005' DEALLOCATE_ABEND

  Description: The source of the error notification is either the remote transaction program or the remote LU. The remote transaction program causes this error notification by issuing an MC_DEALLOCATE verb specifying the TYPE(ABEND) parameter. Alternatively, the remote LU can issue an MC_DEALLOCATE because of a remote transaction program ABEND condition.

Action: Check the transaction program for errors.

- X'0006' DEALLOCATE_ABEND_PROG

Description: The remote transaction program or the remote LU causes this error notification by issuing a DEALLOCATE verb specifying the TYPE(ABEND_PROG) parameter. If the conversation for the remote transaction program is in receive state when the DEALLOCATE occurs, information sent by the local transaction program, and not yet received by the remote transaction program, is purged.

APPC/PC reports this return code on a verb the program issues in send or receive state when the remote transaction program or remote LU deallocates the conversation.

Action: Check the transaction program for errors.

- X'0007' DEALLOCATE_ABEND_SVC

Description: The remote transaction program or the remote LU causes this error notification by issuing a DEALLOCATE verb specifying the TYPE(ABEND_SVC) parameter. If the conversation for the remote transaction program is in receive state when the DEALLOCATE occurs, information sent by the local program, and not yet received by the remote program, is purged.

APPC/PC reports this return code on a verb the program issues in send or receive state when the remote program or remote LU deallocates the conversation.

Action: Check the transaction program for errors.

- X'0008' DEALLOCATE_ABEND_TIMER

Description: The remote transaction program causes this error notification by issuing a DEALLOCATE verb specifying the TYPE(ABEND_TIMER) parameter. If the conversation for the remote transaction program is in receive state when

the DEALLOCATE occurs, information sent by the local transaction program, and not yet received by the remote transaction program, is purged.

APPC/PC reports this return code on a verb the transaction program issues in send or receive state when the remote transaction program or remote LU deallocates the conversation.

Action: Check the transaction program for errors.

- X'0009' DEALLOCATE_NORMAL

  Description: The remote transaction program issued a DEALLOCATE specifying the TYPE(SYNC_LEVEL) or TYPE(FLUSH) parameter. APPC/PC reports this return code on a verb the local transaction program issues in receive state.

  Action: This return code is not an error indication.

- X'F0040000' or X'F004' INCOMPLETE

  Description: The issued verb has suspended without completing its defined function.

  Action: To avoid a deadlock situation (as described under "System Deadlocks" on page 10-2), issue verbs on any other active transaction programs, and then issue one or more GET_ALLOCATE verbs to process incoming ALLOCATEs queued in any LU. Then re-issue the current verb without changing any parameter, (except for the first 12 bytes which you can use to store list pointers to maintain a list of incomplete verbs).

- X'F0050000' or X'F005' INCOMPLETE_ALTERED_VERB

  Description: A verb that has previously returned as INCOMPLETE was altered when it was re-issued. You may have changed the verb, or issued a new verb from the same transaction program before re-issuing the incomplete verb.

Action: Alter the transaction program so that no verb other than the unchanged incomplete verb (except for the first 12 bytes) is issued from that program.

- X'FFFFFFFF' or X'FFFF' INVALID_VERB

  Description: APPC/PC did not recognize the supplied verb operation code and did not execute the verb. An incorrect AH value can also cause this return code.

  Action: Correct the operation code or the AH value.

- X'000B' POSTING_NOT_ACTIVE

  Description: Posting is not active for the specified conversations.

  Action: Issue POST_ON_RECEIPT before testing the conversation.

- X'000C' PROG_ERROR_NO_TRUNC

  Description: The remote transaction program issued a SEND_ERROR specifying the TYPE(PROG) parameter, the conversation for the remote transaction program was in send state, and the SEND_ERROR did not truncate a logical record. No truncation occurs when a program issues SEND_ERROR before sending any logical records or after sending a complete logical record.

  APPC/PC reports this return code on a receive verb and the conversation remains in receive state.

  Action: Check the error.

- X'000E' PROG_ERROR_PURGING

  Description: The remote transaction program issued a SEND_ERROR verb specifying the TYPE(PROG) parameter in receive or confirm state. If the remote transaction program issues a SEND_ERROR verb when it is in receive

state, information sent to, but not yet received by, the remote transaction program is purged.

APPC/PC normally reports this return code on a verb the local transaction program issues after sending information to the remote transaction program. However, APPC/PC can also report this return code on a verb the program issues before sending any information, depending on the verb and when the program issues it. The conversation remains in receive state.

Action: Check the error and prepare to resend data that may have been lost.

- X'000D' PROG_ERROR_TRUNC

Description: The remote transaction program issued a SEND_ERROR specifying the TYPE(PROG) parameter, the conversation for the remote transaction program was in send state, and the SEND_ERROR truncated a logical record. Truncation occurs when a program begins sending a logical record and then issues SEND_ERROR before sending the complete logical record. APPC/PC reports this return code on a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb the local transaction program issues after receiving the truncated logical record.

Action: Check the error.

- X'0011' SVC_ERROR_NO_TRUNC

Description: The remote transaction program or remote LU issued a SEND_ERROR specifying the TYPE(SVC) parameter, the conversation for the remote transaction program was in send state, and the SEND_ERROR did not truncate a logical record. No truncation occurs when a program issues SEND_ERROR before sending any logical records or after sending a complete logical record.

APPC/PC reports this return code on a receive verb and the conversation remains in receive state.

Action: Check the error.

- X'0013' SVC_ERROR_PURGING

Description: The remote transaction program or remote LU issued a SEND_ERROR specifying the TYPE(SVC) parameter in receive or confirm state. If the remote transaction program issues a SEND_ERROR verb when it is in receive state, information sent to, but not yet received by, the remote transaction program is purged.

APPC/PC reports this return code on a verb the local transaction program issues after sending information to the remote transaction program. However, APPC/PC can also report this return code on a verb the program issues before sending any information, depending on the verb and when the program issues it. The conversation remains in receive state.

Action: Check the error and prepare to resend the data that may have been purged.

- X'0012' SVC_ERROR_TRUNC

Description: The remote transaction program or remote LU issued a SEND_ERROR specifying the TYPE(SVC) parameter, the conversation for the remote transaction program was in send state, and the SEND_ERROR truncated a logical record. Truncation occurs when a program begins sending a logical record and then the program or LU issues SEND_ERROR before sending a complete logical record. APPC/PC reports this return code on a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb the local transaction program issues after receiving the truncated logical record.

Action: Check the error.

Verb-Specific Return Codes

The rest of this appendix describes the return codes for all verbs, arranged in alphabetic order by verb name.

ACTIVATE_DLC

- X'00000008' NO_PU_ATTACHED

  Description: APPC/PC has not yet received a valid ATTACH_PU verb.

  Action: Issue an ATTACH_PU verb.

- X'00000283' DLC_FAILURE

  Description: This return code indicates one of many possible error conditions. For example, the adapter number may have been incorrectly specified on the APPC/PC configuration menus or on the ATTACH_LU verb. This return code can also indicate that the adapter address for the IBM Token-Ring Network exists elsewhere on the token ring.

  Action: If the adapter is for the IBM Token-Ring Network, check switch 2 (the primary/secondary adapter switch) and verify that it matches the adapter number parameters specified on the ACTIVATE_DLC and ATTACH_LU verbs, and the adapter number entry on the APPC/PC configuration menus.

  If another program which uses the IBM Token-Ring Network adapter (such as the NETBIOS program) was loaded before APPC/PC, you may have to reload the other program again, specifying an adequate number of service access points and links for APPC/PC operation. For more information of Service Access Point (SAP) operation, see the *IBM Token-Ring Network NETBIOS Program User's Guide*. For information on loading APPC/PC,

refer to the *APPC/PC Installation and Configuration Guide.*

If you unloaded APPC/PC and moved to another IBM PC in the same token ring after your program issued ACTIVATE_DLC, you must use a new soft address on the new IBM PC. Alternatively, you can free the original soft address by turning off the first IBM PC or by re-initializing the first IBM PC with the adapter diagnostic program after unloading APPC/PC. This program is supplied with the IBM Token-Ring Network adapter.

Check the SYSLOG entry for error information on DLC failures.

- X'00000284' UNRECOGNIZED_DLC

  Description: APPC/PC could not find the specified DLC name and adapter number in the configuration file.

  Action: Check the DLC name and adapter number supplied in the configuration file.

- X'00000286' DUPLICATE_DLC

  Description: The specified DLC is already open.

  Action: Check the DLC name and adapter number on the ACTIVATE_DLC verb to see if they are correct.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F0030000' APPC_ABENDED
- X'F0020000' APPC_BUSY
- X'F0010000' APPC_DISABLED
- X'00000000' OK.

- X'0001' PARAMETER_CHECK

  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

    Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

  - X'00000006' DATA_AREA_ACROSS_SEGMENT

    Description: APPC/PC does not permit PIP data to cross a segment boundary.

    Action: Recalculate the segment and offset registers so that offset registers will not wrap back around to 0.

  - X'00000010' BAD_TPN_LEN

    Description: The value that TPN_LENGTH specifies is too short (less than 1) or too long (greater than 64).

    Action: Check the TPN_LENGTH value.

  - X'00000011' BAD_CONV_TYPE

    Description: APPC/PC does not recognize the specified conversation TYPE.

    Action: Check the conversation TYPE field. The only valid values are X'00' for BASIC_CONVERSATION and X'01' for MAPPED_CONVERSATION.

- X'00000012' BAD_SYNC_LEVEL

  Description: APPC/PC does not recognize the specified SYNC_LEVEL type.

  Action: Check the SYNC_LEVEL type field. The only valid values are X'00' for NONE and X'01' for CONFIRM.

- X'00000013' BAD_SECURITY_SELEC

  Description: APPC/PC does not recognize the specified SECURITY type.

  Action: Check the SECURITY type field. The only valid values are X'00' for NO_CONVERSATION_LEVEL_SECURITY, X'01' for SAME, and X'02' for PGM.

- X'00000014' BAD_RETURN_CONTROL

  Description: APPC/PC does not recognize the specified RETURN_CONTROL value.

  Action: Check the RETURN_CONTROL field. The only valid values are X'00' for WHEN_SESSION_ALLOCATED, X'01' for IMMEDIATE, and X'02' for WHEN_SESSION_FREE.

- X'00000015' TOO_BIG_SEC_TOKENS

  Description: APPC/PC does not accept a password or a user ID that is greater than 8 bytes.

  Action: Check length of password or user ID.

- X'00000016' PIP_LEN_INCORRECT

  Description: APPC/PC does not accept PIP data that is longer than 32767 bytes.

  Action: Make sure that the PIP data is not longer than 32767 bytes.

- X'00000017' NO_USE_OF_SNASVCMG

  Description: APPC/PC does not permit the EBCDIC mode name SNASVCMG with the ALLOCATE or MC_ALLOCATE verbs.

  Action: Check the mode name.

- X'00000018' UNKNOWN_PARTNER_MODE

  Description: APPC/PC does not recognize the specified partner LU or mode name.

  Action: Check the partner LU and mode name specified for the ATTACH_LU verb. The partner LU and mode names must be the same as specified for the ATTACH_LU verb.

- X'0014' UNSUCCESSFUL

  Description: The program specified RETURN_CONTROL(IMMEDIATE) on the allocation request and APPC/PC could not allocate the conversation because no contention-winner sessions were available.

  Action: Use the CNOS verb to increase the number of available contention-winner sessions or issue the allocation request with a RETURN_CONTROL (WHEN_SESSION_ACTIVATED) or RETURN_CONTROL (WHEN_SESSION_FREE) parameter. Check the previously issued verbs for error conditions, check all verb parameters for validity, and then resend the verb.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED
- X'0003' ALLOCATION ERROR
- X'F004' INCOMPLETE
- X'F005' INCOMPLETE_ALTERED_VERB

- X'0000' OK.

ATTACH_LU

- X'00000008' NO_PU_ATTACHED

  Description: The PU is not defined because APPC/PC has not yet received a valid ATTACH_PU verb.

  Action: Issue an ATTACH_PU verb.

- X'00000211' ALREADY_ACTIVE_LU

  Description: The LU is already defined.

  Action: Do not attempt to attach two LUs with the same name.

- X'00000212' BAD_PART_SESS

  Description: The session limit for an individual partner LU (PARTNER_LU_SESSION_LIMIT) is greater than the session limit permitted for all partner LUs (LU_SESSION_LIMIT).

  Action: Increase the session limit for all partner LUs (LU_SESSION_LIMIT) to accommodate the specified session limit for the partner LU (that is, the PARTNER_LU_SESSION_LIMIT) or decrease the session limit for the partner LU.

- X'00000213' BAD_RU_SIZES

  Description: The maximum RU size value of the MAX_RU_SIZE parameter is smaller than the minimum RU size value.

  Action: Reverse the values of the RU sizes.

- X'00000214' BAD_MODE_SESS

  Description: The session limit for an individual mode name (MODE_MAX_NEGOTIABLE_SESSION_ LIMIT) is greater than the session limit (PARTNER_LU_SESSION_LIMIT) permitted

for all mode names used for sessions with the specific partner LU.

Action: Increase the PARTNER_LU_SESSION_LIMIT or decrease the session limit of the individual mode name (MODE_MAX_NEGOTIABLE_SESSION_ LIMIT).

- X'00000216' BAD_PACING_CNT

Description: The PACING_SIZE is not in the range of 0 to 63, inclusive.

Action: Check the pacing size. Increase or decrease this size to fit within the specified range.

- X'00000219' EXTREME_RUS

Description: The upper bound for the MAX_RU_SIZE size is too large or the lower bound is too small.

Action: The low value must be at least 16, and the high value must not be more than allowed by the link station that is to carry the session traffic (specified as the MAX_RU_SIZE value in the configuration file). In the most general environment, you should have low value less than or equal to 256 and high value greater than or equal to 256.

- X'0000021A' SNASVCMG_1

Description: APPC/PC does not accept the EBCDIC name "SNASVCMG" as the mode name for a single session connection to communicate data between transaction programs.

Action: Using the mode name "SNASVCMG" implies parallel sessions are being used. Therefore, a single session connection cannot use the "SNASVCMG" mode name.

- X'00000284' UNRECOGNIZED_DLC

  Description: APPC/PC could not find the specified DLC name and adapter number.

  Action: Check the DLC name and adapter number supplied on the APPC/PC configuration menus.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F0030000' APPC_ABENDED
- X'F0020000' APPC_BUSY
- X'F0010000' APPC_DISABLED
- X'00000000' OK.

ATTACH_PU

- X'00000201' ALREADY_ACTIVE_PU

  Description: The PU is already active and cannot be redefined at this time. This error can result if a previous DETACH_PU has not completed.

  Action: Wait for APPC/PC to complete execution of a previous DETACH_PU verb or issue the DETACH_PU immediately before the ATTACH_LU verb.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F0030000' APPC_ABENDED
- X'F0020000' APPC_BUSY
- X'F0010000' APPC_DISABLED
- X'00000000' OK.

CHANGE_LU

- X'00000003' BAD_LU_ID

  Description: APPC/PC does not recognize the specified LU_ID.

Action: Match this LU_ID with the one APPC returns after executing the ATTACH_LU verb.

- X'00000230' INVALID_CHANGE

  Description: The application subsystem has made an invalid change in the management of incoming ALLOCATEs.

  Action: Check the exit pointer. You cannot change the synchronous or asynchronous handling of incoming ALLOCATEs.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F0030000' APPC_ABENDED
- X'F0020000' APPC_BUSY
- X'F0010000' APPC_DISABLED
- X'00000000' OK.

CNOS

- X'0000' OK

  - X'00000000' CNOS_ACCEPTED

    Description: APPC/PC executed the CNOS verb successfully as specified.

    Action: This return code is not an error condition, just a positive response.

  - X'00000001' CNOS_NEGOTIATED

    Description: APPC/PC executed the CNOS verb successfully as negotiated by the partner LU.

    Action: This return code is not an error condition. It indicates that one or more parameter values have been negotiated. The program can obtain the negotiated values by issuing the DISPLAY verb.

- X'0001' PARAMETER_CHECK

  - X'00000003' BAD_LU_ID

    Description: APPC/PC does not recognize the specified LU_ID.

    Action: Match this LU_ID with the one APPC/PC returns after executing the ATTACH_LU verb.

  - X'00000153' ALL_MODE_MUST_RESET

    Description: APPC/PC does not permit the specification of a non-0 session limit when the MODE_NAME_SELECT parameter indicates ALL.

    Action: When setting the MODE_NAME_SELECT parameter to ALL, you must set the session limits to 0. Set limits to 0 or check the MODE_NAME_SELECT parameter.

  - X'00000154' BAD_SNASVCMG_LIMITS

    Description: The application subsystem has specified invalid settings for the PARTNER_LU_MODE_SESSION_ LIMIT, MIN_CONWINNERS_SOURCE, or MIN_CONWINNERS_TARGET parameters when MODE_NAME(SNASVCMG) is indicated.

    Action: The three groups of valid settings for the SNASVCMG mode are as follows:

    - PARTNER_LU_MODE_SESSION_ LIMIT(2)
      MIN_CONWINNERS_SOURCE(1)
      MIN_CONWINNERS_TARGET(1)

    - PARTNER_LU_MODE_SESSION_ LIMIT(1)
      MIN_CONWINNERS_SOURCE(0)
      MIN_CONWINNERS_TARGET(1)

— PARTNER_LU_MODE_SESSION_LIMIT(0)
MIN_CONWINNERS_SOURCE(0)
MIN_CONWINNERS_TARGET(0)

— X'00000155' MINS_GT_TOTAL

Description: The sum of MIN_CONWINNERS_SOURCE and MIN_CONWINNERS_TARGET is greater than the value specified for the PARTNER_LU_MODE_SESSION_LIMIT.

Action: Check values of these three fields.

— X'00000156' MODE_CLOSED

Description: CNOS cannot set a non-0 limit because the local maximum negotiable session limit is currently 0 for the specified mode.

Action: Issue the CNOS with SET_NEGOTIABLE = YES.

— X'00000157' BAD_MODENAME

Description: The specified partner LU does not support the specified MODE_NAME.

Action: Match the MODE_NAME with the MODE_NAME specified with the ATTACH_LU verb.

— X'00000159' RESET_SNA_DRAINS

Description: The SNASVCMG mode does not support the DRAIN settings (CNOS).

Action: Do not set the DRAIN_SOURCE and the DRAIN_TARGET parameters when using CNOS with SNASVCMG mode.

— X'0000015A' SINGLE_NOT_SRC_RESP

Description: For a single session mode (for which PARTNER_LU_SESSION_LIMIT = 1) or the SNASVCMG mode, APPC/PC permits

only the local LU to be responsible for deactivating sessions.

Action: Change the value of the RESPONSIBLE parameter to SOURCE.

– X'0000015B' BAD_PARTNER_LU

Description: APPC/PC does not recognize the specified partner LU name.

Action: The specified PARTNER_LU_NAME must be one of the PARTNER_LU_NAMEs defined by the ATTACH_LU verb. Check the PARTNER_LU_NAME value specified with the CNOS verb or add the PARTNER_LU_NAME in the ATTACH_LU definition.

– X'0000015C' EXCEEDS_MAX_ALLOWED

Description: The local maximum negotiable session limit is less than the session limit specified with the CNOS verb.

Action: Increase the local maximum session limit (MODE_MAX_NEGOTIABLE_SESSION_LIMIT) determined by the ATTACH_LU verb, decrease the session limit (PARTNER_LU_MODE_SESSION_LIMIT) specified with the CNOS verb or use the SET_NEGOTIABLE(YES) option.

– X'0000015D' CHANGE_SRC_DRAINS

Description: APPC/PC does not permit a program to specify MODE_NAME_SELECT(ONE) and DRAIN_SOURCE(YES) when DRAIN_SOURCE(NO) is currently in effect for the specified mode.

Action: A previous CNOS has set session limits to 0 without draining the source. After this action, a program cannot change the drain option.

- X'0002' STATE_CHECK

  - X'00000151' CNOS_SESSLIM_NOT_ZERO

    Description: APPC/PC does not permit a program to change the session limit to a non-0 value unless the limit is already 0.

    Action: Reset the session limit by issuing the CNOS verb with PARTNER_LU_MODE_SESSION_LIMIT=0.

- X'0018' CNOS_PARTNER_REJECT

  Description: The partner LU rejected a CNOS request from the local LU because of a condition specified in one of the secondary return codes.

  Action: Take the action described for the indicated secondary return code.

  - X'00000156' CNOS_MODE_CLOSED

    Description: The local LU cannot negotiate a non-0 session because the local maximum session limit of the partner LU is 0.

    Action: Check the session limit for the specified mode name on the remote LU.

  - X'00000157' CNOS_BAD_MODENAME

    Description: The partner LU does not recognize the specified MODE_NAME.

    Action: Check the partner LU MODE_NAME on the remote system.

  - X'0000015F' CNOS_COMMAND_RACE_REJECT

    Description: APPC/PC is currently processing a CNOS verb issued by the partner LU.

    Action: This is not an error condition. This secondary return code simply indicates that a race condition occurred during CNOS negotiation between two LUs. The program

can issue the DISPLAY verb to obtain the negotiated values. You may reissue the CNOS verb.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED
- X'0000' OK.

CONFIRM and MC_CONFIRM

- X'0001' PARAMETER_CHECK

  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

    Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

  - X'00000002' BAD_CONV_ID

    Description: APPC/PC does not recognize the specified CONV_ID.

    Action: Check the CONV_ID parameter for validity.

  - X'00000031' CONFIRM_ON_SYNC_NONE

    Description: APPC/PC does not permit the program to use this verb if it allocated the conversation with SYNC_LEVEL(NONE).

    Action: The program can issue a CONFIRM or MC_CONFIRM verb only when the SYNC_LEVEL is confirm.

- X'0002' STATE_CHECK

  - X'00000032' CONFIRM_BAD_STATE

    Description: The conversation is not in send state.

    Action: Issue the verb in send state.

  - X'00000033' CONFIRM_NOT_LL_BDY

    Description: The conversation is in send state, and the program started, but did not finish, sending a logical record.

    Action: Finish sending a logical record before issuing CONFIRM.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'0003' ALLOCATION_ERROR
- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED
- X'0019' CONVERSATION_TYPE_MIXED
- X'0010' CONV_FAILURE_NO_RETRY
- X'000F' CONV_FAILURE_RETRY
- X'0005' DEALLOCATE_ABEND
- X'0006' DEALLOCATE_ABEND_PROG
- X'0007' DEALLOCATE_ABEND_SVC
- X'0008' DEALLOCATE_ABEND_TIMER
- X'F004' INCOMPLETE
- X'F005' INCOMPLETE_ALTERED_VERB
- X'0000' OK
- X'000E' PROG_ERROR_PURGING
- X'0013' SVC_ERROR_PURGING.

CONFIRMED and MC_CONFIRMED

- X'0001' PARAMETER_CHECK

  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

- X'00000002' BAD_CONV_ID

  Description: APPC/PC does not recognize the specified CONV_ID.

  Action: Check the CONV_ID parameter for validity.

- X'0002' STATE_CHECK

  - X'00000041' CONFIRMED_BAD_STATE

    Description: The conversation is not in confirm state.

    Action: The local transaction program can issue a CONFIRMED only as a reply to a confirmation request from the remote LU. Check the transaction program for errors.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED
- X'0019' CONVERSATION_TYPE_MIXED
- X'F004' INCOMPLETE
- X'F005' INCOMPLETE_ALTERED_VERB
- X'0000' OK.

CONVERT

- X'00000401' INVALID_DIRECTION

  Description: APPC/PC does not recognize the specified conversion DIRECTION.

  Action: Check the value supplied in the DIRECTION parameter.

- X'00000402' INVALID_TYPE

  Description: APPC/PC does not recognize the CHARACTER_SET type for conversion.

  Action: Check the value supplied in the CHARACTER_SET parameter.

- X'00000403' SEGMENT_OVERLAP

  Description: The area containing characters to be converted or the area that is to contain the converted characters overlaps a segment boundary.

  Action: Make sure the SOURCE or TARGET offset plus the LENGTH value does not exceed 65535. Alternatively, you can reduce the number of bytes to be converted.

- X'00000404' INVALID_FIRST_CHARACTER

  Description: APPC/PC detected an invalid first character for a type "A" conversion.

  Action: Make the first character in the SOURCE buffer an uppercase letter or one of the special characters, $, #, or @.

- X'00000405' TABLE_ERROR

  Description: APPC/PC could not find the type G conversion table.

  Action: Reload APPC/PC and make sure that the file containing the type G conversion table is in the same directory as APPC/PC. Also make sure that the name of the file is the same as that specified on the APPC/PC configuration menus.

- X'00000406' CONVERSION_ERROR

  Description: The CONVERT verb found an unrecognized character and converted it to X'00'.

  Action: Check the characters in the source string.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F0030000' APPC_ABENDED
- X'F0020000' APPC_BUSY
- X'F0010000' APPC_DISABLED
- X'00000000' OK.

DEALLOCATE and MC_DEALLOCATE

- X'0001' PARAMETER_CHECK

  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

    Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

  - X'00000002' BAD_CONV_ID

    Description: APPC/PC does not recognize the specified CONV_ID.

    Action: Check the CONV_ID parameter for validity.

  - X'00000006' DATA_AREA_ACROSS_SEGMENT

    Description: The log data area crosses a segment boundary.

    Action: Make sure the offset of the LOG_DATA address plus the LOG_DATA_LENGTH does not exceed 65535.

  - X'00000051' DEALLOC_BAD_TYPE

    Description: APPC/PC does not recognize the specified TYPE.

Action: Check the value of the TYPE field. The only valid values for DEALLOCATE are X'00', X'01', X'02', X'03', and X'04'. The only valid values for MC_DEALLOCATE are X'00', X'01', and X'05'.

- X'00000057' LOG_LL_WRONG

  Description: The LOG_DATA_LENGTH does not match the value on the LL field of the LOG_DATA.

  Action: Check the length of the log data.

- X'0002' STATE_CHECK

  - X'00000052' DEALLOC_FLUSH_BAD_STATE

    Description: The program specified the TYPE(SYNC_LEVEL) parameter on a conversation specified with SYNC_LEVEL=NONE when the conversation was not in the send state. Alternatively, the program can cause this error by specifying TYPE(FLUSH) when the conversation is not in send state.

    Action: Issue the verb in the correct state.

  - X'00000053' DEALLOC_CONFIRM_BAD_STATE

    Description: The program specified the TYPE(SYNC_LEVEL) parameter on a conversation specified with SYNC_LEVEL=CONFIRM when the conversation was not in send state.

    Action: Issue the verb in the correct state.

  - X'00000055' DEALLOC_NOT_LL_BDY

    Description: The program specified the TYPE(FLUSH) or the TYPE(SYNC_LEVEL) parameter, the conversation is in send state, and the program started but did not finish sending a logical record.

Action: Finish sending the logical record.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'0003' ALLOCATION_ERROR
- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED
- X'0010' CONV_FAILURE_NO_RETRY
- X'000F' CONV_FAILURE_RETRY
- X'0019' CONVERSATION_TYPE_MIXED
- X'0005' DEALLOCATE_ABEND
- X'0006' DEALLOCATE_ABEND_PROG
- X'0007' DEALLOCATE_ABEND_SVC
- X'0008' DEALLOCATE_ABEND_TIMER
- X'F004' INCOMPLETE
- X'F005' INCOMPLETE_ALTERED_VERB
- X'0000' OK
- X'000E' PROG_ERROR_PURGING
- X'0013' SVC_ERROR_PURGING.

DETACH_LU

- X'00000003' BAD_LU_ID

  Description: APPC/PC does not recognize the specified LU_ID.

  Action: Match this LU_ID with the one APPC/PC returns after executing the ATTACH_LU verb.

- X'00000223' SSCP_CONNECTED_LU

  Description: The program can issue DETACH_LU only for independent LUs that do not have active sessions with the SSCP.

  Action: Use the DETACH_PU verb to reset the node.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F0030000' APPC_ABENDED
- X'F0020000' APPC_BUSY
- X'F0010000' APPC_DISABLED

- X'00000000' OK.

DETACH_PU

- X'00000008' NO_PU_ATTACHED

  Description: The application subsystem has not yet issued ATTACH_PU to define the PU being detached.

  Action: Issue an ATTACH_PU verb.

- X'00000272' ADAPTER_CLOSE_FAILURE

  Description: APPC/PC has experienced a failure while trying to close an adapter.

  Action: Check the SYSLOG entry for possible adapter problem indications.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F0030000' APPC_ABENDED
- X'F0020000' APPC_BUSY
- X'F0010000' APPC_DISABLED
- X'00000000' OK.

DISABLE/ENABLE_APPC

The DISABLE/ENABLE_APPC verb does not provide a RETURN_CODE parameter.

DISPLAY

- X'00000003' BAD_LU_ID

  Description: APPC/PC does not recognize the specified LU_ID.

  Action: Match this LU_ID with the one APPC/PC returns after executing the ATTACH_LU verb.

- X'000001B1' BAD_PART_LUNAME

  Description: APPC/PC does not recognize the supplied PARTNER_LU_NAME parameter value.

  Action: Match the PARTNER_LU_NAME with the PARTNER_LU_NAME specified with the ATTACH_LU verb.

- X'000001B2' BAD_MODENAME

  Description: APPC/PC does not recognize the MODE_NAME parameter value.

  Action: Match the MODE_NAME with the MODE_NAME specified with the ATTACH_LU_verb.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F0030000' APPC_ABENDED
- X'F0020000' APPC_BUSY
- X'F0010000' APPC_DISABLED
- X'00000000' OK.

FLUSH and MC_FLUSH

- X'0001' PARAMETER_CHECK

  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

    Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

  - X'00000002' BAD_CONV_ID

    Description: APPC/PC does not recognize the specified CONV_ID.

Action: Check the CONV_ID parameter for validity.

- X'0002' STATE_CHECK

  - X'00000061' FLUSH_NOT_SEND_STATE

    Description: The conversation must be in send state to flush the local LU's send buffer.

    Action: Issue the verb in send state.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED
- X'0019' CONVERSATION_TYPE_MIXED
- X'0000' OK.

GET_ALLOCATE

- X'00000003' BAD_LU_ID

  Description: APPC/PC does not recognize the specified LU_ID.

  Action: Match this LU_ID with the one APPC/PC returns after executing the ATTACH_LU verb.

- X'00000281' GET_ALLOC_BAD_TYPE

  Description: APPC/PC does not recognize the parameter specified in the TYPE field.

  Action: Check the TYPE field. The TYPE should be X'00' for DEQUEUE or X'01' for TEST.

- X'00000282' UNSUCCESSFUL

  Description: The LU is not currently holding any incoming ALLOCATEs in the queue.

  Action: This is not an error indication.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F0030000' APPC_ABENDED
- X'F0020000' APPC_BUSY
- X'F0010000' APPC_DISABLED
- X'00000000' OK.

GET_ATTRIBUTES and MC_GET_ATTRIBUTES

- X'0001' PARAMETER_CHECK

  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

    Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

  - X'00000002' BAD_CONV_ID

    Description: APPC/PC does not recognize the specified CONV_ID.

    Action: Check the CONV_ID parameter for validity.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED
- X'0019' CONVERSATION_TYPE_MIXED
- X'0000' OK.

GET_TYPE

- X'0001' PARAMETER_CHECK

  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

    Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

  - X'00000002' BAD_CONV_ID

    Description: APPC/PC does not recognize the specified CONV_ID.

    Action: Check the CONV_ID parameter for validity.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED
- X'0000' OK.

PASSTHROUGH

The PASSTHROUGH verb does not provide a RETURN_CODE parameter.

POST_ON_RECEIPT

- X'0001' PARAMETER_CHECK
  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

    Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

  - X'00000002' BAD_CONV_ID

    Description: APPC/PC does not recognize the specified CONV_ID.

    Action: Check the CONV_ID parameter for validity.

  - X'00000091' INVALID_LENGTH

    Description: The program specified an illegal value for the MAX_LENGTH parameter.

    Action: Specify a length in the range 0 to 32767 inclusive.

  - X'00000093' P_ON_R_BAD_FILL

    Description: The program specified an illegal value for the FILL parameter.

    Action: Check the FILL field. The only valid values are X'00' for BUFFER and X'01' for LL.

- X'0002' STATE_CHECK
  - X'00000092' P_ON_R_NOT_RCV_STATE

    Description: The conversation is not in receive state.

    Action: Issue the verb in receive state.

- X'0019' CONVERSATION_TYPE_MIXED

  Description: The program issued the verb on a mapped conversation.

  Action: Issue the verb only on a basic conversation.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED
- X'0000' OK.

PREPARE_TO_RECEIVE and MC_PREPARE_TO_RECEIVE

- X'0001' PARAMETER_CHECK

  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

    Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

  - X'00000002' BAD_CONV_ID

    Description: APPC/PC does not recognize the specified CONV_ID.

    Action: Check the CONV_ID parameter for validity.

  - X'000000A1' P_TO_R_INVALID_TYPE

    Description: APPC/PC does not recognize the specified TYPE.

    Action: Check the TYPE field. The only valid values are X'00' for SYNC_LEVEL and X'01' for FLUSH.

- X'0002' STATE_CHECK

  - X'000000A2' UNFINISHED_LL

    Description: The conversation is in send state, and the program started, but did not finish, sending a logical record.

    Action: Finish sending the logical record.

  - X'000000A3' P_TO_R_NOT_SEND_STATE

    Description: The conversation is not in send state.

    Action: Issue the verb in send state.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'0003' ALLOCATION_ERROR
- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED
- X'0019' CONVERSATION_TYPE_MIXED
- X'0010' CONV_FAILURE_NO_RETRY
- X'000F' CONV_FAILURE_RETRY
- X'0005' DEALLOCATE_ABEND
- X'0006' DEALLOCATE_ABEND_PROG
- X'0007' DEALLOCATE_ABEND_SVC
- X'0008' DEALLOCATE_ABEND_TIMER
- X'F004' INCOMPLETE
- X'F005' INCOMPLETE_ALTERED_VERB
- X'0000' OK
- X'000E' PROG_ERROR_PURGING
- X'0013' SVC_ERROR_PURGING.

RECEIVE_AND_WAIT and MC_RECEIVE_AND_WAIT

- X'0001' PARAMETER_CHECK

  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

- X'00000002' BAD_CONV_ID

  Description: APPC/PC does not recognize the specified CONV_ID.

  Action: Check the CONV_ID parameter for validity.

- X'00000006' DATA_AREA_ACROSS_SEGMENT

  Description: The receive data area crosses a segment boundary.

  Action: Make sure the offset of the LOG_DATA address plus the LOG_DATA_LENGTH does not exceed 65535.

- X'000000B5' RCV_AND_WAIT_BAD_FILL

  Description: The program specified an illegal value for the FILL parameter.

  Action: Check the FILL field. The only valid values are X'00' for BUFFER and X'01' for LL.

- X'0002' STATE_CHECK

  - X'000000B1' RCV_AND_WAIT_BAD_STATE

    Description: The conversation is not in send or receive state.

    Action: Issue the verb only in send or receive state.

  - X'000000B2' RCV_AND_WAIT_NOT_LL_BDY

    Description: The conversation is in send state, and the program started, but did not finish, sending a logical record.

    Action: Finish sending the logical record.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'0003' ALLOCATION_ERROR
- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED
- X'0019' CONVERSATION_TYPE_MIXED
- X'0010' CONV_FAILURE_NO_RETRY
- X'000F' CONV_FAILURE_RETRY
- X'0005' DEALLOCATE_ABEND
- X'0006' DEALLOCATE_ABEND_PROG
- X'0007' DEALLOCATE_ABEND_SVC
- X'0008' DEALLOCATE_ABEND_TIMER
- X'0009' DEALLOCATE_NORMAL
- X'F004' INCOMPLETE
- X'F005' INCOMPLETE_ALTERED_VERB
- X'0000' OK
- X'000C' PROG_ERROR_NO_TRUNC
- X'000E' PROG_ERROR_PURGING
- X'000D' PROG_ERROR_TRUNC
- X'0011' SVC_ERROR_NO_TRUNC
- X'0013' SVC_ERROR_PURGING
- X'0012' SVC_ERROR_TRUNC.

RECEIVE_IMMEDIATE and MC_RECEIVE_ IMMEDIATE

- X'0001' PARAMETER_CHECK

  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

    Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

  - X'00000002' BAD_CONV_ID

    Description: APPC/PC does not recognize the specified CONV_ID.

    Action: Check the CONV_ID parameter for validity.

- X'00000006' DATA_AREA_ACROSS_SEGMENT

  Description: The data area crosses a segment boundary.

  Action: Make sure the offset of the LOG_DATA address plus the LOG_DATA_LENGTH does not exceed 65535.

- X'000000C4' RCV_IMMD_BAD_FILL

  Description: The program specified an illegal value for the FILL parameter.

  Action: Check the FILL field. The only valid values are X'00' for BUFFER and X'01' for LL.

- X'0002' STATE_CHECK

  - X'000000C1' RCV_IMMD_NOT_RCV_STATE

    Description: The conversation is not in receive state.

    Action: Issue the verb in receive state.

- X'0014' UNSUCCESSFUL

  Description: There is nothing to receive.

  Action: This return code is not an error indication.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'0003' ALLOCATION_ERROR
- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED
- X'0019' CONVERSATION_TYPE_MIXED
- X'0010' CONV_FAILURE_NO_RETRY
- X'000F' CONV_FAILURE_RETRY
- X'0005' DEALLOCATE_ABEND
- X'0006' DEALLOCATE_ABEND_PROG
- X'0007' DEALLOCATE_ABEND_SVC
- X'0008' DEALLOCATE_ABEND_TIMER
- X'0009' DEALLOCATE_NORMAL
- X'F004' INCOMPLETE
- X'F005' INCOMPLETE_ALTERED_VERB

- X'0000' OK
- X'000C' PROG_ERROR_NO_TRUNC
- X'000E' PROG_ERROR_PURGING
- X'000D' PROG_ERROR_TRUNC
- X'0011' SVC_ERROR_NO_TRUNC
- X'0013' SVC_ERROR_PURGING
- X'0012' SVC_ERROR_TRUNC.

REQUEST_TO_SEND and MC_REQUEST_TO_SEND

- X'0001' PARAMETER_CHECK

  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

    Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

  - X'00000002' BAD_CONV_ID

    Description: APPC/PC does not recognize the specified CONV_ID.

    Action: Check the CONV_ID parameter for validity.

- X'0002' STATE_CHECK

  - X'000000E1' R_T_S_NOT_RCV_STATE

    Description: The conversation is not in receive or confirm state.

    Action: Do not issue REQUEST_TO_SEND in send state.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED

- X'0019' CONVERSATION_TYPE_MIXED
- X'0000' OK.

SEND_DATA and MC_SEND_DATA

- X'0001' PARAMETER_CHECK
  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

    Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

  - X'00000002' BAD_CONV_ID

    Description: APPC/PC does not recognize the specified CONV_ID.

    Action: Check the CONV_ID parameter for validity.

  - X'00000006' DATA_AREA_ACROSS_SEGMENT

    Description: The send data area crosses a segment boundary.

    Action: Make sure the offset of the LOG_DATA address plus the LOG_DATA_LENGTH does not exceed 65535.

  - X'000000F1' BAD_LL

    Description: DATA contains an invalid logical record length (LL) value of hex 0000, 0001, 8000, or 8001.

    Action: Check the logical record length and the LL field.

- X'0002' STATE_CHECK

  - X'000000F2' SEND_DATA_NOT_SEND_STATE

    Description: The conversation is not in send state.

    Action: Issue the verb in send state.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'0003' ALLOCATION_ERROR
- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED
- X'0019' CONVERSATION_TYPE_MIXED
- X'0010' CONV_FAILURE_NO_RETRY
- X'000F' CONV_FAILURE_RETRY
- X'0005' DEALLOCATE_ABEND
- X'0006' DEALLOCATE_ABEND_PROG
- X'0007' DEALLOCATE_ABEND_SVC
- X'0008' DEALLOCATE_ABEND_TIMER
- X'0009' DEALLOCATE_NORMAL
- X'F004' INCOMPLETE
- X'F005' INCOMPLETE_ALTERED_VERB
- X'0000' OK
- X'000E' PROG_ERROR_PURGING
- X'0013' SVC_ERROR_PURGING.

SEND_ERROR and MC_SEND_ERROR

- X'0001' PARAMETER_CHECK

  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

    Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

- X'00000002' BAD_CONV_ID

  Description: APPC/PC does not recognize the specified CONV_ID.

  Action: Check the CONV_ID parameter for validity.

- X'00000006' DATA_AREA_ACROSS_SEGMENT

  Description: The log data area crosses a segment boundary.

  Action: Make sure the offset of the LOG_DATA address plus the LOG_DATA_LENGTH does not exceed 65535.

- X'00000102' LOG_LL_WRONG

  Description: The LL field of the log data does not match the specified length.

  Action: Check the length of the data and the LL field.

- X'00000103' BAD_TYPE

  Description: APPC/PC does not recognize the specified error TYPE.

  Action: Check the value of the TYPE parameter. The valid values are PROG (X'00') and SVC (X'01').

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'0003' ALLOCATION_ERROR
- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED
- X'0019' CONVERSATION_TYPE_MIXED
- X'0010' CONV_FAILURE_NO_RETRY
- X'000F' CONV_FAILURE_RETRY
- X'0005' DEALLOCATE_ABEND
- X'0006' DEALLOCATE_ABEND_PROG
- X'0007' DEALLOCATE_ABEND_SVC
- X'0008' DEALLOCATE_ABEND_TIMER

- X'0009' DEALLOCATE_NORMAL
- X'F004' INCOMPLETE
- X'F005' INCOMPLETE_ALTERED_VERB
- X'0000' OK
- X'000E' PROG_ERROR_PURGING
- X'0013' SVC_ERROR_PURGING.

SET_PASSTHROUGH

The SET_PASSTHROUGH verb does not provide a RETURN_CODE parameter.

TEST and MC_TEST

- X'0000' OK

  - X'00000000' POSTED_DATA

    Description: The LU has posted the connection and data is available in an LU's receive buffer.

    Action: This return code does not indicate an error condition.

  - X'00000001' POSTED_NOT_DATA

    Description: The LU has posted the conversation and information other than data is available in the LU's receive buffer. This return code is not an error condition.

    Action: Issue RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE to receive the information. This return code is not an error notification.

- X'0001' PARAMETER_CHECK

  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

- X'00000002' BAD_CONV_ID

  Description: APPC/PC does not recognize the specified CONV_ID.

  Action: Check the CONV_ID parameter for validity.

- X'0002' STATE_CHECK

  - X'00000122' NOT_RCV_STATE

    Description: The conversation is not in receive state and the program specified the POSTED option for the TYPE parameter.

    Action: Issue the verb in receive state.

- X'0014' UNSUCCESSFUL

  Description: For TEST(POSTED), the conversation has not been posted. For TEST (REQUEST_TO_SEND_RECEIVED), the REQUEST_TO_SEND indication has not been received.

  Action: Reissue the verb later.

- X'000B' POSTING_NOT_ACTIVE

  Description: Posting is not active for the specified conversations.

  Action: Issue POST_ON_RECEIPT before testing the conversation.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'0003' ALLOCATION_ERROR
- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY

- X'F001' APPC_DISABLED
- X'0019' CONVERSATION_TYPE_MIXED
- X'0010' CONV_FAILURE_NO_RETRY
- X'000F' CONV_FAILURE_RETRY
- X'000A' DATA_POSTING_BLOCKED
- X'0005' DEALLOCATE_ABEND
- X'0006' DEALLOCATE_ABEND_PROG
- X'0007' DEALLOCATE_ABEND_SVC
- X'0008' DEALLOCATE_ABEND_TIMER
- X'0009' DEALLOCATE_NORMAL
- X'F004' INCOMPLETE
- X'F005' INCOMPLETE_ALTERED_VERB
- X'000C' PROG_ERROR_NO_TRUNC
- X'000E' PROG_ERROR_PURGING
- X'000D' PROG_ERROR_TRUNC
- X'0011' SVC_ERROR_NO_TRUNC
- X'0013' SVC_ERROR_PURGING
- X'0012' SVC_ERROR_TRUNC.

TP_ENDED

- X'00000001' BAD_TP_ID

  Description: APPC/PC does not recognize the specified TP_ID.

  Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F0030000' APPC_ABENDED
- X'F0020000' APPC_BUSY
- X'F0010000' APPC_DISABLED
- X'F0040000' INCOMPLETE
- X'F0050000' INCOMPLETE_ALTERED_VERB
- X'00000000' OK.

TP_STARTED

- X'00000003' BAD_LU_ID

  Description: APPC/PC does not recognize the specified LU_ID.

  Action: Match this LU_ID with the one APPC/PC returns after executing the ATTACH_LU verb.

- X'00000243' TOO_MANY_TPS

  Description: APPC/PC is already running the maximum number of transaction programs that this LU can run concurrently (as defined with the MAX_TPS parameter in the ATTACH_LU verb). This return code indicates that the application subsystem cannot initiate the transaction program locally. The condition may be temporary if an incoming ALLOCATE was in the process of being rejected.

  Action: Terminate other transaction programs to make room, or issue the CHANGE_LU verb to increase MAX_TPS.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F0030000' APPC_ABENDED
- X'F0020000' APPC_BUSY
- X'F0010000' APPC_DISABLED
- X'00000000' OK.

TP_VALID

- X'00000001' BAD_TP_ID

  Description: APPC/PC does not recognize the specified TP_ID.

  Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or

the CREATE_TP record for a remotely initiated TP.

- X'00000002' BAD_CONV_ID

  Description: APPC/PC does not recognize the specified CONV_ID.

  Action: Check the CONV_ID parameter for validity.

- X'00000110' BAD_STATE

  Description: TP_VALID does not follow a GET_ALLOCATE.

  Action: Issue TP_VALID verb immediately after GET_ALLOCATE.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F0030000' APPC_ABENDED
- X'F0020000' APPC_BUSY
- X'F0010000' APPC_DISABLED
- X'F0040000' INCOMPLETE
- X'F0050000' INCOMPLETE_ALTERED_VERB
- X'00000000' OK.

TRACE

The TRACE verb does not provide a RETURN_CODE parameter.

TRANSFER_MS_DATA

- X'00000301' SSCP_PU_SESSION_NOT_ACTIVE

  Description: APPC/PC could not send the NMVT because the SSCP_PU session was not active.

  Action: Contact the host operator to activate an SSCP_PU session.

- X'00000302' DATA_EXCEEDS_RU_SIZE

  Description: The data exceeded the allowable RU size.

  Action: Reduce the size of data sent or check that the RU size is adequate.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'F0030000' APPC_ABENDED
- X'F0020000' APPC_BUSY
- X'F0010000' APPC_DISABLED
- X'00000000' OK.

WAIT

- X'0000' OK
  - X'00000000' POSTED_DATA

    Description: The LU has posted the conversation and data is available in an LU's receive buffer.

    Action: This return code does not indicate an error condition.

  - X'00000001' POSTED_NOT_DATA

    Description: The LU has posted the conversation and information other than data is available in the LU's receive buffer.

    Action: Issue RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE to receive the information. This return code is not an error notification.

- X'0001' PARAMETER_CHECK
  - X'00000001' BAD_TP_ID

    Description: APPC/PC does not recognize the specified TP_ID.

Action: Match the TP_ID with the TP_ID that APPC/PC returns after executing the TP_STARTED verb for a locally initiated TP or the CREATE_TP record for a remotely initiated TP.

- X'00000002' BAD_CONV_ID

  Description: APPC/PC does not recognize the specified CONV_ID.

  Action: Check the CONV_ID parameter for validity.

- X'0002' STATE_CHECK

  - X00000122' NOT_RCV_STATE

    Description: One of the conversations in the CONV_ID list is not in receive state (for TYPE = POSTED).

    Action: Check the TYPE field for accuracy. Issue verbs to place the conversation in receive state.

- X'000B' POSTING_NOT_ACTIVE

  Description: The program issued a WAIT verb before issuing POST_ON_RECEIPT verbs for any of the conversations included on the WAIT verb list.

  Action: The transaction program must issue a POST_ON_RECEIPT for at least one of the conversations on the WAIT verb list.

For descriptions of the following return codes, see "Common Return Codes" on page C-2.

- X'0003' ALLOCATION_ERROR
- X'F003' APPC_ABENDED
- X'F002' APPC_BUSY
- X'F001' APPC_DISABLED
- X'0019' CONVERSATION_TYPE_MIXED
- X'0010' CONV_FAILURE_NO_RETRY
- X'000F' CONV_FAILURE_RETRY
- X'000A' DATA_POSTING_BLOCKED

- X'0006' DEALLOCATE_ABEND_PROG
- X'0007' DEALLOCATE_ABEND_SVC
- X'0008' DEALLOCATE_ABEND_TIMER
- X'0009' DEALLOCATE_NORMAL
- X'F004' INCOMPLETE
- X'F005' INCOMPLETE_ALTERED_VERB
- X'0000' OK
- X'000C' PROG_ERROR_NO_TRUNC
- X'000E' PROG_ERROR_PURGING
- X'000D' PROG_ERROR_TRUNC
- X'0011' SVC_ERROR_NO_TRUNC
- X'0013' SVC_ERROR_PURGING
- X'0012' SVC_ERROR_TRUNC.

Appendix D. SYSLOG Type Codes

This appendix lists the SYSLOG type codes. SYSLOG codes represent error conditions including data errors reported by the transaction program, link errors, configuration errors, and system protocol errors. If protocol errors persist, refer to Appendix I, "Statement of Service," for information on obtaining program service.

Some of the terminology below is not defined in this manual. See the *SNA Format and Protocol Reference Manual*, for unfamiliar terms. The asterisk (*) in the following tables indicates that the SYSLOG type code signals a protocol error.

Type	Subtype	Meaning
X'0001'	Sense code	Conversation error generated from SEND_ERROR, MC_SEND_ERROR, DEALLOCATE (of an ABEND type), MC_DEALLOCATE (TYPE = ABEND). SYSLOG ADDITIONAL_INFO points to a byte containing one of the following: X'00' - locally generated conversation error X'01' - remotely generated conversation error If log data is present, SYSLOG DATA points to the data. If log data is not present, SYSLOG DATA is X'FFFFFFFF'.
X'0002'	*	Received an OAF-DAF address which duplicates one already active
X'0003'		Cannot route request to LU (The LU may not be ATTACHed

Type	Subtype	Meaning
		or the LU name or NAU address may be incorrect.)
X'0004'	*	OAF-DAF address space filled (Too many sessions may be active.)
X'0005'	*	Begin Bracket bit not set in RH
X'0006'	*	Received an unexpected RTR
X'0007'	*	Received an unexpected BIS reply
X'0008'	Sense Code *	Conversation-level protocol error detected at the basic conversation level. The session is deactivated with an UNBIND indicating a type X'FE' protocol error.
X'0009'	*	Conversation level protocol error detected for a mapped conversation. APPC/PC issues a DEALLOCATE TYPE(ABEND) for the conversation.
X'000A'	*	Received a BIND response with no outstanding request
X'000B'	*	NS header not recognized on an SSCP-LU or SSCP-PU session
X'000C'	*	Received an UNBIND request with syntax error
X'000D'	*	Received session control request with format error
X'000E'	*	Received UNBIND response with no outstanding request
X'000F'	*	Received INITSELF response with no outstanding request
X'0010'	Sense code *	Received BIND request with syntax, state, or semantic error
X'0011'	*	Received unexpected BIS request
X'0012'	*	Received invalid NS record
X'0013'	*	Session level protocol error detected
X'0014'		Received frame from DLC in error
X'0015'	*	Bracket error
X'0016'		Received route inop - line has gone down (This is the only logged SDLC error.) See "SDLC Problems" on page D-22 .
X'0017'		Network Management NMVT message (See "Network Management NMVT Message"

Type	Subtype	Meaning
		in this appendix.)
	X'00000000'	User-defined data
	X'00000001'	A complete NMVT (NS header, major vector, and appropriate subvectors)
	X'00000002'	TRANSFER_ALERT
	X'00000003'	TRANSFER_PDSTATS
X'0018'	*	Received Error Data GDS Variable from a partner on a mapped conversation. APPC/PC issues a DEALLOCATE(ABEND) for the conversation
X'0019'		Session Level Security Error
X'001A'		Conversation Level Security Error
X'001B'		Link Level Error (See *IBM Token-Ring Network Problems* on page D-5 and *IBM Token-Ring Network Problem Determination Guide*)
	X'00000101'	Invalid adapter
	X'00000102'	Command not recognized
	X'00000103'	Invalid link ID
	X'00000105'	Invalid parameter table
	X'00000106'	Required parameter missing
	X'00000107'	Invalid Parameter Table field
	X'00000108'	Invalid number of connections
	X'00000200'	Adapter not open
	X'00000300'	Duplicate command
	X'00000400'	Remote station not responding
	X'00000500'	DLC unsuccessful—retry
	X'00000600'	DLC unsuccessful—no retry
	X'00000700'	XID protocol error
	X'00000800'	Connection failure
	X'8xxxxxxx'	DLC-specific link level error (A 1 must appear in left-most bit; the meaning of each code is determined by the particular DLC.)

Note:

For link level errors, SYSLOG ADDITIONAL_INFO points to a DLC address of up to 26 characters, in the same format as in the ATTACH_LU verb.

Type	Subtype	Meaning
X'001C'		NMVT message too large for the RU size
X'001D'	*	Received non-normal UNBIND (Due to a protocol error or a DETACH_PU(HARD) verb)

The subcodes below refer to the UNBIND type:

Type	Subtype	Meaning
	X'00000003'	Session outage
	X'00000006'	Invalid parameters
	X'00000007'	Virtual route map
	X'00000008'	Route extension INOP
	X'00000009'	Hierarchical reset
	X'0000000A'	SSCP gone
	X'0000000B'	Virtual route deactivated
	X'0000000C'	Unrecoverable LU failure
	X'0000000E'	Recoverable LU failure
	X'0000000F'	Cleanup
	X'00000011'	Gateway node cleanup
	X'000000FE'	Protocol violation

Note:

For abnormal UNBIND errors, SYSLOG ADDITIONAL-INFO points to the 17 character fully qualified partner LU name, followed by the 8-character mode name.

Type	Subtype	Meaning
X'FFFF'		abnormal termination of APPC/PC

The abnormal termination subtypes below indicate either an invalid configuration file or an APPC/PC program error. Restore or reconstruct the configuration file from the APPC/PC menus, if possible, to correct these problems. (For

information on the APPC/PC menus refer to the *APPC/PC Installation and Configuration Guide.*)

Type	Subtype	Meaning
	X'00000000'	No available storage, APPC/PC will try to close adapters.
	X'00000001'	No available storage, APPC/PC cannot continue. If the IBM Token-Ring Network is left in an unstable state, the keyboard may be locked. The SYSLOG exit routine can choose to unlock the keyboard and not return to APPC/PC.
	X'00000002'	Process being created is unknown.
	X'00000003'	Process fell through to its end.
	X'00000004'	Sending to a nonexistent queue.
	X'00000005'	Receiving from a nonexistent queue.
	X'00000006' *	Invalid variant variable in configuration file.

* These SYSLOG type codes indicate protocol errors.

IBM Token-Ring Network Problems

APPC/PC reports failures on the IBM Token-Ring Network and the adapter using its SYSLOG facilities. APPC/PC uses two SYSLOG TYPE codes for these data-link-level errors. A SYSLOG of type LINK_ERROR (X'001B') indicates that either the IBM Token-Ring Network adapter or the component of APPC/PC that interfaces to the adapter has determined that it is unable to complete a command. A SYSLOG of type Network Management NMVT Message (X'0017') indicates the loss or impending loss of an adapter or a network resource.

Link-Level Errors

The SUBTYPE and ADDITIONAL_INFO fields of the SYSLOG record return information for link level errors.

Understanding Link Error Subtypes

The link error **subtype** identifies the specific link-level error that occurred.

The following list of link error **subtypes** describes the causes of these errors and provides recommended solutions.

- X'00000101' Invalid Adapter

  Description: APPC/PC tried to issue a command to the IBM Token-Ring Network containing an adapter ID which is not X'00' or X'01'.

  Action: Verify that the definition of Partner LUs on the ATTACH_LU verb specifies only adapter ID values X'00' or X'01'.

- X'00000102' Command Not Recognized

  Description: An unexpected internal APPC/PC software error occurred.

  Action: Refer to Appendix I, "Statement of Service," for information on obtaining program service.

- X'00000103' Invalid Link ID

  Description: **APPC/PC** Invalidly issued commands for an IBM Token-Ring Network link which was in the process of being disconnected, or after an unrecoverable software error occurred.

  Action: ALLOCATE a conversation as necessary to establish the link. If the error persists, refer to Appendix I, "Statement of Service," for information on obtaining program service.

- X'00000105' Invalid Parameter Table

  Description: An unexpected internal APPC/PC software error occurred.

  Action: Refer to Appendix I, "Statement of Service," for information on obtaining service.

- X'00000106' Required Parameter Missing

  Description: An unexpected internal software error occurred.

  Action: Refer to Appendix I, "Statement of Service," for information on obtaining program service.

- X'00000107' Invalid Parameter Table Field

  Description: The IBM Token-Ring Network adapter found an invalid value for a parameter table field.

  Action: If the error occurred during processing of the ACTIVATE_DLC verb, verify that the adapter address specified in your APPC/PC configuration is valid (that is, the high order bit in this address is not set).

- X'00000108' Invalid Number of Connections

  Description: The number of incoming link connections calculated was greater than the maximum number of link stations.

  Action: Retry the ACTIVATE_DLC verb. If the error persists, refer to Appendix I, "Statement of Service," for information on obtaining program service.

- X'00000200' Adapter Not Open

  Description: The program issued commands to an IBM Token-Ring Network adapter that the application subsystem has not opened.

  Action: Issue the ACTIVATE_DLC verb to open the adapter.

- X'00000300' Duplicate Command

  Description: The program issued two ACTIVATE_DLC verbs for the same IBM Token-Ring Network adapter, or an unexpected software error occurred.

Action: Verify that the program is not issuing two ACTIVATE_DLC verbs for the same adapter. If this action does not solve the problem, refer to Appendix I, "Statement of Service" for information on obtaining program service.

- X'00000400' Remote Station Not Responding

  Description: IBM Token-Ring Network adapter received no response on the network when attempting to locate a partner node.

  Action: Verify that the specification of the adapter address of the partner node is correct and that the node is operational on the network (that is, its adapter has been open and initialized).

- X'00000500' DLC Unsuccessful—Retry

  Description: The IBM Token-Ring Network adapter detected a sequence error.

  Action: Retry the conversation that failed.

- X'00000600' DLC Unsuccessful—No Retry

  Description: A catastrophic, unrecoverable failure occurred.

  Action: The application subsystem must reopen the adapter that failed by issuing the ACTIVATE_DLC verb.

- X'00000700' XID Protocol Error

  Description: An unexpected XID was received during the XID exchange process on the IBM Token-Ring Network link.

  Action: If error persists refer to Appendix I, "Statement of Service," for information on obtaining program service.

- X'00000800' Connection Failure

  Description: The IBM Token-Ring Network link that previously existed with a partner node has been unexpectedly disconnected.

  Action: Check that the partner node has not deliberately disconnected.

Understanding Link Error Data

In addition to a **subtype** code, the link error specifies an ADDITIONAL_INFO pointer. This pointer indicates the location of the following information:

DLC NAME: 'ITRN ' (8 ASCII characters)

ADAPTER NUMBER: X'00' if the error occurred on the primary adapter and X'01' if the error occurred on the secondary adapter.

DESTINATION ADDRESS: length (one byte)

DESTINATION ADDRESS: Token-Ring Network address of the partner node as defined in the ATTACH_LU verb. Not applicable on all link errors. (up to 16-byte hex address)

Network Management NMVT Messages

The following fields of the SYSLOG record contain information for Network Management NMVT messages:

SUBTYPE Specifies the NMVT message:

X'00000000' = USER DATA

X'00000002' = ALERT

DATA Provides a Network Management vector.

ALERTS

In addition to logging ALERTS locally, APPC/PC also sends the ALERT vector to an application program residing in the host computer if there is an active host SSCP-PU session when APPC/PC reports the ALERT. The application program displays information about the failure condition to an operator console.

The ALERT Network Management vector is composed of the ALERT Major Vector, the Correlator Subvector, the Product Set ID Subvector, the Basic Alert Subvector, and a variable number of Hexadecimal Detail Qualifier Subvectors.

The values for the Basic Alert Subvector and Hexadecimal Detail Qualifier Subvectors are unique to the type of error APPC/PC reports. The error categories that this section describes are:

- Adapter bring-up errors
- Adapter open errors
- Adapter check
- Ring status
- Ring status-remove received
- PC-detected
- Software errors.

It is beyond the scope of this document to describe the subvector formats in detail. Consult the *SNA Reference Summary* for a complete description of the structure and format of the ALERT Major Vector and the associated subvectors. Consult the *IBM Token-Ring Network PC Adapter Technical Reference Manual* for a description of the specific adapter errors contained in the Hexadecimal Detail Qualifier Subvectors.

In the following format descriptions, **h** represents a hexidecimal digit.

Adapter Bring-Up Errors

Description: Bring-up errors occur when the ACTIVATE_DLC verb cannot initialize the IBM Token-Ring Network adapter successfully. These errors indicate that the adapter has failed a hardware diagnostic test.

Action: Run the IBM Token-Ring Network adapter diagnostics as described in the manual *IBM Token-Ring Network PC Adapter Guide to Operations.*

For bring up errors, the ALERT subvectors are:

Basic Alert Subvector

• Subvector Length	X'0D'	
• Subvector Key	X'91'	
• Flags	X'00'	Not operator initiated
• Alert Type	X'01'	Permanent error
• General Cause Code	X'01'	Hardware or microcode
• Specific Component Code	X'0012'	Communication link adapter
• Alert Description Code	X'0040'	
• User Action Code	X'0040'	
• Detail Text Ref Code	X'0040'	

Hexadecimal Detail Qualifier

• Subvector Length	X'04'	
• Subvector Key	X'A1'	
• Qualifier Data	X'hhhh'	Adapter bring-up error code

Hexadecimal Detail Qualifier

• Subvector Length	X'03'	
• Subvector Key	X'A1'	
• Qualifier Data	X'hh'	Adapter (0 | 1)

EBCDIC Qualifier

• Subector Length	X'04'	
• Subvector Key	X'A0'	Type = qualifying data (hex)
• Qualifier Data	X'4040'	Null field = NPDA screen filler

Hex Qualifier

• Subector Length	X'12'	

• Subvector Key	X'A1'	Type = qualifying data (hex)
• Qualifier Data	6X'hh'	Adapter node address
	10X'hh'	Microcode EC level (10 bytes)

Adapter Open Errors

Description: The ACTIVATE_DLC verb has failed to open the IBM Token-Ring Network adapter. The adapter could not successfully enter itself into the network.

Action: Perform IBM Token-Ring Network problem determination procedures.

For open errors, the ALERT subvectors are:

Basic Alert Subvector

• Subvector Length	X'0D'	
• Subvector Key	X'91'	
• Flags	X'00'	Not operator initiated
• Alert Type	X'01'	Permanent error
• General Cause Code	X'0B'	Hardware
• Specific Component Code	X'0080'	Token-Ring adapter
• Alert Description Code	X'0041'	
• User Action Code	X'0041'	
• Detail Text Ref Code	X'0041'	

Hexadecimal Detail Qualifier

• Subvector Length	X'04'	
• Subvector Key	X'A1'	
• Qualifier Data	X'00hh'	

Hexadecimal Detail Qualifier

• Subvector Length	X'03'	
• Subvector Key	X'A1'	
• Qualifier Data	X'hh'	Adapter (0 | 1)

EBCDIC Qualifier

• Subvector Length	X'04'	
• Subvector Key	X'A0'	Type = qualifying data (hex)
• Qualifier Data	X'4040'	Null field = NPDA screen filler

Hex Qualifier

• Subvector Length	X'12'	
• Subvector Key	X'A1'	Type = qualifying data (hex)
• Qualifier Data	6X'hh'	Adapter node address
	10X'hh'	Microcode EC level (10 bytes)

Adapter Check

Description: The IBM Token-Ring Network Adapter has failed during operation.

Action: Perform IBM Token-Ring Network problem determination procedures.

For adapter check errors, the ALERT subvectors are:

Basic Alert Subvector

• Subvector Length	X'0D'	
• Subvector Key	X'91'	
• Flags	X'00'	Not operator initiated
• Alert Type	X'01'	Permanent error
• General Cause Code	X'01'	Hardware or microcode
• Specific Component Code	X'0012'	Communications Link Adapter
• Alert Description Code	X'0042'	
• User Action Code	X'0040'	
• Detail Text Ref Code	X'0042'	

Hexadecimal Detail Qualifier

• Subvector Length	X'0A'	
• Subvector Key	X'A1'	
• Qualifier Data	X'hhhh'	Adapter check error code
	6X'hh'	Adapter check parms 0-2

Hexadecimal Detail Qualifier

• Subvector Length	X'03'	
• Subvector Key	X'A1'	
• Qualifier Data	X'hh'	Adapter (0 | 1)

EBCDIC Qualifier

• Subvector Length	X'04'	
• Subvector Key	X'A0'	Type = qualifying data (hex)
• Qualifier Data	X'4040'	Null field = NPDA screen filler

Hex Qualifier

• Subvector Length	X'12'	
• Subvector Key	X'A1'	Type = qualifying data (hex)
• Qualifier Data	6X'hh'	Adapter node address
	10X'hh'	Microcode EC level (10 bytes)

Ring Status

Description: The IBM Token-Ring Network adapter has encountered a token-ring failure condition. If the failure was the fault of the adapter, the token ring can recover without intervention.

Action: Perform IBM Token-Ring Network problem determination procedures.

For ring status errors, the ALERT subvectors are:

Basic Alert Subvector

• Subvector Length	X'0D'	
• Subvector Key	X'91'	
• Flags	X'00'	Not operator initiated
• Alert Type	X'01'	Permanent error
• General Cause Code	X'0B'	Hardware or microcode
• Specific Component Code	X'0080'	Token-Ring Network error
• Alert Description Code	X'0043'	
• User Action Code	X'0041'	
• Detail Text Ref Code	X'0043'	

Hexadecimal Detail Qualifier

• Subvector Length	X'04'	
• Subvector Key	X'A1'	
• Qualifier Data	X'hhhh'	Ring status code

Hexadecimal Detail Qualifier

• Subvector Length	X'03'	
• Subvector Key	X'A1'	
• Qualifier Data	X'hh'	Adapter (0 | 1)

EBCDIC Qualifier

• Subvector Length	X'04'	
• Subvector Key	X'A0'	Type = qualifying data (hex)
• Qualifier Data	X'4040'	Null field = NPDA screen filler

Hex Qualifier

• Subvector Length	X'12'	
• Subvector Key	X'A1'	Type = qualifying data (hex)
• Qualifier Data	6X'hh'	Adapter node address

10X'hh'	Microcode EC level (10 bytes)	

Ring Status - Remove Received

Description: Another network entity forced the IBM Token-Ring Network adapter to remove itself from the network.

Action: If this problem occurs repeatedly, refer to Appendix I, "Statement of Service," for information on obtaining program service.

For ring status-remove received errors, the ALERT subvectors are:

Basic Alert Subvector

• Subvector Length	X'0D'	
• Subvector Key	X'91'	
• Flags	X'00'	Not operator initiated
• Alert Type	X'01'	Permanent error
• General Cause Code	X'0F'	Undetermined
• Specific Component Code	X'00FF'	Undetermined
• Alert Description Code	X'0044'	
• User Action Code	X'0044'	
• Detail Text Ref Code	X'0044'	

Hexadecimal Detail Qualifier

• Subvector Length	X'04'	
• Subvector Key	X'A1'	
• Qualifier Data	X'hhhh'	Ring status code

Hexadecimal Detail Qualifier

• Subvector Length	X'03'	
• Subvector Key	X'A1'	
• Qualifier Data	X'hh'	Adapter (0 | 1)

EBCDIC Qualifier

• Subvector Length	X'04'

• Subvector Key	X'A0'	Type = qualifying data (hex)
• Qualifier Data	X'4040'	Null field = NPDA screen filler

Hex Qualifier

• Subvector Length	X'12'	
• Subvector Key	X'A1'	Type = qualifying data (hex)
• Qualifier Data	6X'hh'	Adapter node address
	10X'hh'	Microcode EC level (10 bytes)

PC-Detected

Description: The IBM Token-Ring Network adapter detected an internal error.

Action: Reboot the program if possible; if the error persists, refer to Appendix I, "Statement of Service," for information on obtaining program service.

For PC-detected errors, the ALERT subvectors are:

Basic Alert Subvector

• Subvector Length	X'0D'	
• Subvector Key	X'91'	
• Flags	X'00'	Not operator initiated
• Alert Type	X'01'	Permanent error
• General Cause Code	X'01'	Hardware or microcode
• Specific Component Code	X'0012'	Communication link adapter
• Alert Description Code	X'0045'	
• User Action Code	X'0045'	
• Detail Text Ref Code	X'0045'	

Hexadecimal Detail Qualifier

• Subvector Length	X'04'

• Subvector Key	X'A1'	
• Qualifier Data	X'hhhh'	Ring status code

Hexadecimal Detail Qualifier

• Subvector Length	X'03'	
• Subvector Key	X'A1'	
• Qualifier Data	X'hh'	Adapter (0 | 1)

EBCDIC Detail Qualifier

• Subvector Length	X'04'	
• Subvector Key	X'A0'	
• Qualifier Data	X'4040'	Null Field

Hexadecimal Detail Qualifier

• Subvector Length	X'12'	
• Subvector Key	X'A1'	
• Qualifier Data	6X'hh'	Adapter node address
• Qualifier Data	10X'hh'	Microcode EC level (10 bytes)

Software Errors

Description: The IBM Token-Ring Network data link control software detected an internal error.

Action: Reboot the program if possible; if the error persists, refer to Appendix I, "Statement of Service," for information on obtaining program service.

For software errors, the ALERT subvectors are:

Basic Alert Subvector

• Subvector Length	X'0D'	
• Subvector Key	X'91'	
• Flags	X'00'	Not operator initiated
• Alert Type	X'01'	Permanent error
• General Cause Code	X'18'	Microcode or software
• Specific Component Code	X'00FF'	Undetermined
• Alert Description Code	X'0046'	
• User Action Code	X'0046'	
• Detail Text Ref Code	X'0046'	

Hexadecimal Detail Qualifier

• Subvector Length	X'04'	
• Subvector Key	X'A1'	
• Qualifier Data	X'hhhh'	Contents of IP register at detection

Hexadecimal Detail Qualifier

• Subvector Length	X'03'	
• Subvector Key	X'A1'	
• Qualifier Data	X'hh'	Adapter (0 | 1)

EBCDIC Detail Qualifier

• Subvector Length	X'04'	
• Subvector Key	X'A0'	
• Qualifier Data	X'4040'	Null Field

Hexadecimal Detail Qualifier

• Subvector Length	X'12'	
• Subvector Key	X'A1'	
• Qualifier Data	6X'hh'	Adapter node address
• Qualifier Data	10X'hh'	Microcode EC level

Hexadecimal Detail Qualifier

• Subvector Length	X'10'	
• Subvector Key	X'A1'	
• Qualifier Data	X'hhhh'	APPC/PC software error code
	X'hhhh'	Contents of register ES
	X'hhhh'	Contents of register BX
	X'hhhh'	Contents of register SI
	X'hhhh'	Contents of register DS
	X'hhhh'	Contents of register DI
	X'hhhh'	Contents of memory at ES:BX

USER DATA

As with ALERTS, APPC/PC logs USER DATA locally. Unlike ALERTS, however, APPC/PC does not send USER DATA information on the SSCP-PU session. The USER DATA Network Management vector contains the USER DATA Major Vector, the Correlator Subvector, and a variable number of USER DATA subvectors.

The fields and values for the subvectors are unique to the type of error APPC/PC reports. It is beyond the scope of this document to describe the contents of the adapter log and status block that APPC/PC returns in the USER DATA subvectors. For this information, consult the *IBM Token-Ring Network PC Adapter Technical Reference Manual.*

The user data subvectors fall into two categories:

- DLC link status
- Counter overflow.

The descriptions of these USER DATA subvectors are as follows:

DLC Link Status

Description: The IBM Token-Ring Network data link control software detected a link status that indicates an error condition.

For DLC link status errors, the USER DATA subvectors are:

- Adapter (0 | 1) X'hh'
- User Data Code X'10' condition = Link Status
- Qualifier Data
  - X'hhhh' DLC Status Block
  - 6X'hh' Remote adapter node address
  - X'00hh' Remote Service Access Point
  - 6'hh' Frame Reject (FRMR) data (when applicable)

Counter Overflow

Description: The IBM Token-Ring Network data link control software detected a statistical counter overflow.

For Counter Overflow errors, the USER DATA subvectors are:

- Adapter (0 | 1) X'hh'
- User Data Code X'0020' condition = Log Counter Overflow
- User Data Log ID
  - For Adapter Counter Overflow

    Log ID X'0001'

Qualifier Data	14X'hh'	Adapter Log (14 bytes)

- For DLC Link Counter Overflow

Log ID	X'0002'	
Qualifier Data	6X'hh'	Remote adapter node address
	X'hhhh'	Remote Service Access Point
	16X'hh'	DLC Link Log

- For DLC SAP Counter Overflow

Log ID	X'0003'	
Qualifier Data	18X'hh'	DLC SAP Log (18 bytes)

SDLC Problems

APPC/PC reports failures in an SDLC link through a SYSLOG of type X'0016' to indicate the failure of the adapter or a network resource. If a link failure occurs APPC/PC attempts to reactivate SDLC five times. If this is not successful you must fix the line problem and re-issue the ACTIVATE_DLC verb. The following timers within SDLC indicate when INOP conditions can occur.

The abbreviations used below are:

CTS: Clear To Send

DCE: Date Communication Equipment (a modem)

RTS: Request To Send

DTE: Data Terminal Equipment

DSR: Data Set Ready

DISC: Disconnect command

UA: Unnumbered Acknowledgement

30-millisecond CTS Dropout

If the CTS signal from the modem goes inactive during an active transmission (that is, when RTS from the DTE is active), SDLC initiates a 30-millisecond time-out. SDLC generates a link failure if DSR remains inactive after 30 milliseconds.

30-millisecond DSR Dropout

If there is an abnormal transition in DSR status from the modem (that is, the modem changes from active state to inactive state without the DTR changing from active to inactive state), SDLC will wait for 30 milliseconds for DSR to become active again. SDLC generates a link failure if DSR remains inactive after 30 milliseconds.

40-second Transmit Failure

If a transmit operation continues for more than 40 seconds, SDLC generates a link failure. This time limit applies for both primary and secondary link stations. This time interval is sufficient to transmit 4 kilobytes of data using the maximum supported frame size and the minimum supported speed of 1200 bits per second.

5-second DISC Not Received Time-out

After a secondary station receives a "close," it waits for 5 seconds for a DISC from the primary station. If the DISC arrives within this time limit, the secondary station responds with a UA and waits for the primary station to terminate the link in a normal fashion. If the DISC does not arrive within the 5 seconds, the secondary station terminates the link connection and stops responding to the primary station.

10-second Inactivity Time-out

A secondary SDLC station generates a link failure if it detects line inactivity (no polls from the primary station) for more than 10 seconds.

Appendix E. Sample Programs

The *APPC/PC Structures and Sample Programs* diskette included with the *APPC/PC Installation and Configuration Guide* contains a set of sample programs written in IBM PC Macro Assembler language and assembled using the IBM Macro Assembler Version 2.00. These programs represent two small application subsystems and the two sides of a cooperating transaction program. The sample programs run on two IBM PCs connected by the IBM Token-Ring Network. The diskette includes the following four executable programs:

SEND_AS	A sample sending side application subsystem.
SEND_TP	A sample sending side transaction program.
RCV_AS	A sample receiving side application subsystem.
RCV_TP	A sample receiving side transaction program.

The *APPC/PC Structures and Sample Programs* diskette also includes files containing the listings for each program: SEND_AS, SEND_TP, RCV_AS, and RCV_TP. These programs illustrate the design of an application subsystem and the coding of APPC/PC verbs using the macro-assembler verb- request parameter lists included on the diskette.

The application subsystem programs (SEND_AS.EXE and RCV_AS.EXE) are minimum application subsystems as described in Chapter 5, "Using Control Verbs." Both of these application subsystems follow the same basic operation sequence:

- Verify that APPC/PC is loaded.
- Translate (CONVERT) ASCII names to EBCDIC.
- Define the Passthrough exit to APPC/PC. APPC/PC uses this exit to communicate the TP_ID between the application subsystem and its transaction program. This exit also communicates the CONV_ID to remotely allocated transaction programs.
- Initialize the session using the following APPC/PC verbs:

  ATTACH_PU defines the Physical Unit.

ATTACH_LU defines the Logical Unit.
ACTIVATE_DLC opens the DLC adapter.
CNOS AUTO_ACTIVATEs the single session between the two nodes.

- Execute the transaction program.

  On the receiving side, the application subsystem performs a GET_ALLOCATE loop while it waits for an incoming ALLOCATE. When the application subsystem receives an ALLOCATE, it verifies the ALLOCATE and then initiates the receive side transaction program.

- After the sample transaction program terminates, the application subsystem takes down the session with the following APPC/PC verbs:

  CNOS sets the session limit to 0.
  DETACH_LU deactivates the Logical Unit.
  DETACH_PU deactivates the Physical Unit.

- The application subsystem then resets the Passthrough exit address, and

- Exits to DOS.

The sample transaction programs (SEND_TP.EXE, and RCV_TP.EXE) are small programs that demonstrate the use of conversation verbs. Both of these programs follow the same operation sequence:

- Inform (TP_INITIATE) APPC/PC and the application subsystem that the transaction program has begun, and in turn, receive the TP_ID (and in the case of the receive side, the CONV_ID) from the application subsystem (see Chapter 2, "Developing an Application Subsystem").

- Set up the conversation (ALLOCATE).

- Execute some conversation verbs (SEND_DATA and RECEIVE_AND_WAIT).

- Take down the conversation (DEALLOCATE).

- Inform APPC/PC that the transaction program is complete (TP_ENDED).

- Exit to the application subsystem.

Each transaction program requires a TP_ID and a CONV_ID in order to issue valid conversation verbs. On the send side, the TP_ID is returned on the TP_STARTED verb issued by the application subsystem, and the CONV_ID is returned on the ALLOCATE verb. The receive side, on the other hand, need only have these values communicated to it by the application subsystem when the receive side is initiated, since the ALLOCATE has already been issued by the send side.

To provide the same interface to both transaction programs, the application subsystem defines a user-defined verb for the sample application subsystem.

Sample User-Defined Verb

The transaction program uses this user-defined verb to request the TP_ID (and in the case of the remotely initiated transaction program, the CONV_ID) from the application subsystem.

TP_ INITIATE	Supplied Parameters: LOCAL_REMOTE_INDICATOR (variable) TP_NAME (variable) LU_NAME (variable)
	Returned Parameters: TP_ID (variable) CONV_ID (variable) RETURN_CODE (variable)
	;

Supplied Parameters:

LOCAL_REMOTE_INDICATOR specifies whether this transaction program is local (that is, the program that will issue the ALLOCATE), or remote (that is, the program allocated by a remote transaction program).

TP_NAME specifies the name of the transaction program the application subsystem is initiating.

LU_NAME specifies the name of the partner LU at which the transaction program is being initiated.

Returned Parameters:

TP_ID specifies the TP_ID.

CONV_ID specifies the CONV_ID, when a remote transaction program is executing the TP_INITIATE.

RETURN_CODE specifies the return code indicating whether the TP_INITIATE verb is acceptable (0) or not acceptable (non-0).

Sample Conversation

The sample conversation performed by the transaction program is as follows. The send side of the conversation executes the following verbs:

TP_INITIATE(LOCAL) causes the application subsystem to issue a TP_STARTED verb and return the TP_ID.

ALLOCATE causes APPC/PC to allocate a conversation and return the CONV_ID.

SEND_DATA sends data on the conversation.

DEALLOCATE deallocates the conversation.

TP_ENDED informs APPC/PC that the transaction program is complete.

The receive side of the conversation executes the following verbs:

TP_INITIATE(REMOTE) returns the TP_ID and the CONV_ID to the transaction program.

RECEIVE_AND_WAIT receives the transmitted data.

RECEIVE_AND_WAIT receives the deallocate.

TP_ENDED informs APPC/PC that the transaction program is complete.

Sample Program Execution

The procedure for running the sample program is:

1. Configure the hardware of two IBM PCs which are to run the sample program (they should each contain an IBM Token-Ring Network adapter).

2. Use the APPC/PC Configuration program to prepare two APPC/PC diskettes (one for each side).

   The send side diskette must contain:

   - An APPC/PC configuration file. Use the configuration program to create a configuration file as described in the *APPC/PC Installation and Configuration Guide.* Use the default values with the following changes:

     - On the "IBM Token-Ring DLC Parameters" screen, set the "Load Option" parameter to 1.

     - On the same screen, set the "Local Node Address" parameter to a soft address of 400000000001.

   - APPC/PC system files (APPC/PC commands and execution code)

   - The sample programs, SEND_AS.EXE (the sample application subsystem) and SEND_TP.EXE (the sample transaction program).

   The receive side diskette must contain:

   - An APPC/PC configuration file. Use the configuration program as described in the *APPC/PC Installation and Configuration Guide* to create a configuration file. Use the default values with the following changes:

     - On the "IBM Token-Ring DLC Parameters" screen, set the "Load Option" parameter to 1.

- On the same screen, set the "Local Node Address" parameter to a soft address of 400000000002.

- APPC/PC system files (APPC/PC commands and execution code).

- The sample programs, RCV_AS.EXE (the sample application subsystem) and RCV_TP.EXE (the sample transaction program).

3. Boot the two IBM PCs.

4. Type **TOKREUI** and press Enter to load the support interface code for the IBM Token-Ring Network adapter.

5. Load APPC/PC on each machine.

6. On side two, type **RCV_AS** and press Enter.

   The application subsystem prints a few messages to let you know that it has brought up the node and that it is waiting for an incoming ALLOCATE.

7. On side one, type **SEND_AS** and press Enter.

This action starts the sending application subsystem, which activates the node, and executes the transaction program.

When the transaction program begins execution, the receive side of the transaction program (RCV_TP.EXE) is automatically loaded and executed when the incoming ALLOCATE is received from the send side. After receipt of the ALLOCATE, the send side sends a message, and the receive side displays the received message. Each side terminates the transaction and brings down the session.

The sample program can be rerun by simply executing the application subsystem.

Modifying the Sample Program for SDLC

The sample program provided on the diskette is written to use the IBM Token-Ring Network. To run this sample program on an SDLC connection, modify the program as follows:

- Use the configuration program to define an SDLC Data Link Control (DLC).
- Change the application subsystem programs (SEND_AS.ASM, and RCV_AS.ASM) to refer to the correct DLC.

  The places in these programs that include DLC-specific information are as follows:

  In the ATTACH_LU control block (the PARTNER_LU structure), the following fields identify the DLC.

  dlcname — Identifies the DLC Adapter Name (for example, 'SDLC' or 'ITRN'). The value should match that entered on the configuration menu.

  adaptid — Indicates the adapter number (for example, 0 or 1). This field should have a value of 0 for SDLC operation.

  adp_adrlen — Indicates the length of the Partner LU adapter address in bytes. This field should have a value of 0 because the SDLC code does not use this field.

  The following fields in the ACTIVATE_DLC control block identify the DLC.

  dlcname — Identifies the DLC Adapter Name ('SDLC' or 'ITRN'). The value should match that entered on the APPC/PC configuration menus, and that found in the Partner LU portion of the ATTACH_LU control block.

  adaptid — Indicates the adapter number (for example, 0 or 1). This field should have a value of 0.

Note: If the program is to use a switched line, you should prompt the user to establish the switched connection (that is, dial the originating telephone). The application subsystem should display this prompt after issuing the ACTIVATE_DLC verbs and then it should wait for acknowledgment from the user in the form of a keystroke before continuing.

After entering these changes to both of the application subsystem programs, SEND_AS.ASM and RCV_AS.ASM, assemble each using the IBM Macro Assembler (Version 2.0). Then link each program using the latest version of the link program to create new EXE files. Execute the programs using the same instructions as for operation with an IBM Token-Ring Network.

APPC/PC—CICS Sample Program

The following program is an assembler-language program written as a transaction program to CICS. This program was written using the EXEC CICS GDS format and can be run against the receive side transaction program supplied for the sample for two IBM PCs.

Appendix F, "Sample CICS Host Configuration for APPC/PC" contains the CICS and VTAM definitions needed to run this program.

```
*ASM XOPTS(CICS,GDS,SOURCE)
*****************************************************************
*                                                               *
*           SAMPLE TRANSACTION PROGRAM      SEND SIDE           *
*                                                               *
* FUNCTION: This transaction program runs on the CICS           *
*           subsystem.  It issues the verbs necessary to transmit*
*           a simple string of data to a transaction program on  *
*           the receive side.                                   *
*                                                               *
* METHOD:  Issue the following verbs                            *
*              ALLOCATE                                         *
*              CONNECT PROCESS                                  *
*              SEND                                             *
*              SEND LAST WAIT                                   *
*              FREE                                             *
*****************************************************************
```

```
DFHEISTG DSECT
         USING *,2,3
         SPACE 2
SMP1     DFHEIENT CODEREG=(2,3)
         PRINT NOGEN
         B     INTRO
         SPACE 2
         DC    CL4'SMP1'       IDENTIFY PROGRAM
         SPACE 2
INTRO    DS    0H
*********************************************************
*                 ALLOCATE                              *
*********************************************************
         EXEC  CICS GDS ALLOCATE                                *
               SYSID('APPC')                                    *
               CONVID(CONVID)                                   *
               RETCODE(RCODE)                                   *
               MODENAME('normal')
         CLI   RCODE,X'01'
         BE    QUIT
         CLI   RCODE,X'02'
         BE    QUIT
         SPACE 2
*********************************************************
*             CONNECT PROCESS                           *
*********************************************************
         EXEC  CICS GDS CONNECT PROCESS                         *
               PROCNAME('rcvtp')                                *
               PROCLENGTH(5)                                    *
               SYNCLEVEL(0)                                     *
               CONVID(CONVID)                                   *
               CONVDATA(CONVDATA)                               *
               RETCODE(RCODE)
         CLI   RCODE,X'03'
         BE    QUIT
         CLI   RCODE,X'04'
         BE    QUIT
         CLI   RCODE,X'05'
         BE    QUIT
         SPACE 2
*********************************************************
*                   SEND                                *
*********************************************************
         EXEC  CICS GDS SEND                                    *
               FROM(REQ1)                                       *
               FLENGTH(45)                                      *
               CONVID(CONVID)                                   *
               CONVDATA(CONVDATA)                               *
               RETCODE(RCODE)
         CLI   RCODE,X'03'
         BE    QUIT
         CLI   RCODE,X'04'
         BE    QUIT
         CLI   RCODE,X'05'
         BE    QUIT
         SPACE 2
```

```
*********************************************************
*              SEND LAST WAIT                           *
*********************************************************
         EXEC  CICS GDS SEND LAST WAIT                                *
               CONVID(CONVID)                                         *
               CONVDATA(CONVDATA)                                     *
               RETCODE(RCODE)
         CLI   RCODE,X'03'
         BE    QUIT
         CLI   RCODE,X'04'
         BE    QUIT
         CLI   RCODE,X'05'
         BE    QUIT
         SPACE 2
*********************************************************
*                    FREE                                *
*********************************************************
         EXEC  CICS GDS FREE                                          *
               CONVID(CONVID)                                         *
               CONVDATA(CONVDATA)                                     *
               RETCODE(RCODE)
         CLI   RCODE,X'03'
         BE    QUIT
         CLI   RCODE,X'04'
         SPACE 2
QUIT     DS    0H
         EXEC  CICS RETURN
         EJECT
CONVID   DC    F'0'               CONVERSATION IDENTIFIER
RCODE    DC    XL4'00'            RETURN CODE INFORMATION
CONVDATA DC    XL24'00'           CONVERSATION RELATED DATA
APSYSID  DS    CL4                SYSTEM ID                          0
INLEN    DS    F
         EJECT
REQ1     DC  XL2'002D'
         DC  XL16'2A2A2A20446174612074726 16E736D69'
         DC  XL16'747465642066726F6D20434943532074'
         DC  XL11'6F205243565450202A2A2A'
         END
```

Appendix F. Sample CICS Host Configuration for APPC/PC

This appendix contains the CICS and VTAM definitions necessary to use APPC/PC with a connection between an IBM PC and a host system.

```
          DEFINITIONS FOR APPC/PC - CICS SAMPLE TEST

*******************************
*****  CICS DEFINITIONS  *****
*******************************


*****  PCT DEFINITION  *****

         DFHPCT TYPE=ENTRY,                                          *
                TRANSID=SMP1,                                        *
                PROGRAM=SMP1,                                        *
                TWASIZE=160,                                         *
                DTB=(YES),                                           *
                SPURGE=YES,                                          *
                TPURGE=YES,                                          *
                INBFMH=ALL,                                          *
                RSL=PUBLIC,                                          *
                RSLC=NO                                              *

*****  PPT DEFINITION  *****

         DFHPPT TYPE=ENTRY,PROGRAM=SMP1,PGMLANG=ASSEMBLER,RSL=PUBLIC


*****  TCT DEFINITION  *****

APPC     DFHTCT TYPE=SYSTEM,       LUTYPE62 IBM PC                   X
                TRMTYPE=LUTYPE62,                                    X
                ACCMETH=VTAM,                                        X
                CONNECT=AUTO,                                        X
                NETNAME=rcvlu (See note 1)                           X
                SYSIDNT=APPC,                                        X
                FEATURE=SINGLE,                                      X
                BUFFER=1024,                                         X
                RUSIZE=256,                                          X
                TCTUAL=255,                                          X
                TRMPRTY=0,                                           X
                TRMSTAT=TRANSCEIVE
```

```
*****************************
*****  VTAM DEFINITIONS  *****
*****************************


*****  NCP DEFINITION   (FOR GROUP,LINE,PU,LU)  *****

HDXGRP GROUP LNCTL=SDLC,          SDLC CONNECTION                          X
               DIAL=NO,           LEASED LINK (NON-SWITCHED)               X
               TYPE=NCP,          LINK DEDICATED TO NCP; I.E., NO PEP      X
               REPLYTO=3,         LINK IDLE FOR 3 MINUTES, TIMEOUT         X
               NRZI=YES           MODEM AND IBM PC MUST ALSO BE NRZI=YES
         SPACE 2
HDXLN1 LINE ADDRESS=020,          LINK ADDRESS FOR IBM PC ATTACHMENT       X
               CLOCKNG=EXT,       EXTERNAL MODEM PROVIDES CLOCKING         X
               SPEED=9600,        LINK SPEED                               X
               DUPLEX=FULL,       DUPLEX TRANSMISSION FACILITY             X
               ISTATUS=ACTIVE,    LINK, PU AND LUS INITIALLY ACTIVATED     X
               SSCPFM=USSSCS,     USE CHAR CODED MSGS ON SSCP-LU SESSION X
               MODETAB=LU62MODT,  USER DEFINED VTAM LOGON MODE TABLE       X
               DLOGMOD=SAMPLE     DEFAULT ENTRY IN LOGON MODE TABLE
         SPACE 2
     SERVICE ORDER=(HDX260A)
         SPACE 2
puname     PU   ADDR=1C,          SDLC STATION ADDRESS FOR IBM PC          X
               ANS=STOP,          STOP POLLING IF HOST SESSION FAILS       X
               MAXOUT=7,          ALLOW MAX OF 7 OUTSTANDING SDLC FRAMES X
               MAXDATA=265,       MAX DATA TRANSFER TO IBM PC              X
               PACING=7,          PACING WINDOW SIZE NCP TO IBM PC         X
               PASSLIM=8,         SERVICE ORDER TABLE PASS LIMIT           X
               PUTYPE=2,          IBM PC IS PU2.0 NODE TO NCP BOUNDARY     X
               RETRIES=(,1,4)     PAUSE 1 SECOND BETWEEN 4 RETRIES
         SPACE 2
rcvlu           LU   LOCADDR=1,ISTATUS=INACTIVE (See note 2)
APPCPC02        LU   LOCADDR=2,ISTATUS=INACTIVE
APPCPC03        LU   LOCADDR=3,ISTATUS=INACTIVE
APPCPC04        LU   LOCADDR=4,ISTATUS=INACTIVE
APPCPC05        LU   LOCADDR=5,ISTATUS=INACTIVE
APPCPC06        LU   LOCADDR=6,ISTATUS=INACTIVE
APPCPC07        LU   LOCADDR=7,ISTATUS=INACTIVE

*****  MODETAB DEFINITION  *****

LU62MODT MODETAB
         MODEENT LOGMODE=NORMAL
         MODEEND
         END

*****  APL DEFINITION FOR CICS  *****

SENDLU APPL  ABCNAME=SENDLU,AUTH=(ALQ,PASS,VPACE),                        X
             DLOGMOD=NORMAL,MODETAB=LU62MODT,                             X
             EAS=10,PARSESS=YES,VPACING=3,SORSCIP=YES
```

Notes:

1. The NETNAME parameter specifies the network name to identify the APPC system to ACF/VTAM. In APPC/PC this name is specified on the ATTACH_LU verb with the LU_NAME parameter.

2. The LOCADDR parameter specifies the LU local address of the session and is equivalent to a logical unit number. In APPC/PC this LU local address is specified on the ATTACH_LU verb with the LU_LOCAL_ADDRESS parameter.

Appendix G. APPC/PC Implementation of the LU6.2 Architecture

This appendix first describes the APPC architected optional functions supported by APPC/PC. It then describes the mapping of the APPC/PC verbs to the verbs as described in the *SNA Transaction Programmer's Reference Manual for LU Type 6.2.*

Base and Option Sets for APPC/PC

The APPC/PC program product supports the base set of APPC functions plus a list of optional subsetted functions. For a description of APPC function subsetting, see the topic "Product-Support-Subsetting" in Chapter 3 of *SNA Transaction Programmer's Reference Manual for LU Type 6.2.* Appendix A of the same manual describes each APPC subset in detail.

APPC/PC supports the following optional subsets:

- Immediate allocation of a session
- Session-level LU-LU verification
- User ID verification
- Program-supplied user ID and password
- PIP data (sending only)
- Logging of data in a system log
- Flush the LU's send buffer

- Prepare to receive
- Long locks
- Post on receipt with wait
- Post on receipt with test for posting
- Receive immediate
- Test for request-to-send received
- Get attributes
- Get conversation type
- Mapped conversation LU services component
- MIN_CONWINNERS_TARGET parameter
- RESPONSIBLE(TARGET) parameter
- DRAIN_TARGET(NO) parameter
- LU-parameter verbs
- LU-LU session limit
- Maximum RU size bounds
- Contention winner automatic activation limit

Basic Conversation Verbs

This section contains the cross-reference lists for basic conversation verbs and parameters.

Cross-Reference for Basic Conversation Verbs

The following table provides a cross-reference for Basic Conversation verbs:

LU Type 6.2 Architecture	APPC/PC Implementation
ALLOCATE	ALLOCATE
BACKOUT	
CONFIRM	CONFIRM
CONFIRMED	CONFIRMED
DEALLOCATE	DEALLOCATE
FLUSH	FLUSH
GET_ATTRIBUTES	GET_ATTRIBUTES
GET_TYPE	GET_TYPE
POST_ON_RECEIPT	POST_ON_RECEIPT
PREPARE_TO_RECEIVE	PREPARE_TO_RECEIVE
RECEIVE_AND_WAIT	RECEIVE_AND_WAIT
RECEIVE_IMMEDIATE	RECEIVE_IMMEDIATE
REQUEST_TO_SEND	REQUEST_TO_SEND
SEND_DATA	SEND_DATA
SEND_ERROR	SEND_ERROR
SYNCPT	
TEST	TEST
WAIT	WAIT

Cross-Reference for Basic Conversation Verb Parameters

The following tables provide a cross-reference for Basic Conversation verb parameters.

LU Type 6.2 ALLOCATE parameters	APPC/PC ALLOCATE parameters
	TP_ID
LU_NAME	PARTNER_LU_NAME
MODE_NAME	MODE_NAME
TPN	TPN
TYPE	CONVERSATION_TYPE
RETURN_CONTROL	RETURN_CONTROL
SYNC_LEVEL	SYNC_LEVEL
SECURITY	SECURITY
PIP	PIP
RESOURCE	CONV_ID
RETURN_CODE	RETURN_CODE

Note:

TP_ID specifies the identifier for the instance of the transaction program issuing this verb. The application subsystem receives this value from APPC/PC when it starts the program. For more information on the TP_ID parameter, see the descriptions of the "CREATE_TP" or "TP_STARTED" verbs in Chapter 5, "Using Control Verbs."

APPC/PC does not support MODE_NAME(SNASVCMG) from the API.

APPC/PC does not support the APPC RETURN_CONTROL(DELAYED_ALLOCATION_PERMITTED) option. It does, however, support an additional option: RETURN_CONTROL(WHEN_SESSION_FREE).

The only synchronization level values that APPC/PC supports are NONE and CONFIRM.

APPC/PC provides three security options: NONE, SAME, and PGM.

LU Type 6.2 CONFIRM parameters	**APPC/PC CONFIRM parameters**
	TP_ID
RESOURCE	CONV_ID
RETURN_CODE	RETURN_CODE
REQUEST_TO_SEND_RECEIVED	REQUEST_TO_SEND_RECEIVED

LU Type 6.2 CONFIRMED parameters	**APPC/PC CONFIRMED parameters**
	TP_ID
RESOURCE	CONV_ID
	RETURN_CODE

LU Type 6.2 DEALLOCATE parameters	**APPC/PC DEALLOCATE parameters**
	TP_ID
RESOURCE	CONV_ID
TYPE	TYPE

LU Type 6.2 DEALLOCATE parameters	APPC/PC DEALLOCATE parameters
LOG_DATA	LOG_DATA
RETURN_CODE	RETURN_CODE

Note:

APPC/PC does not support deallocation TYPE(LOCAL).

LU Type 6.2 FLUSH parameters	APPC/PC FLUSH parameters
	TP_ID
RESOURCE	CONV_ID
	RETURN_CODE

LU Type 6.2 GET_ATTRIBUTES parameters	APPC/PC GET_ATTRIBUTES parameters
	TP_ID
RESOURCE	CONV_ID
	RETURN_CODE
	LU_ID
OWN_FULLY_QUALIFIED_ LU_NAME	OWN_NET_ID_NAME OWN_LU_NAME
PARTNER_LU_NAME	PARTNER_LU_NAME
PARTNER_FULLY_QUALIFIED_ LU_NAME	PARTNER_FULLY_QUALIFIED_ LU_NAME
MODE_NAME	MODE_NAME
SYNC_LEVEL	SYNC_LEVEL
SECURITY_USER_ID	USER_ID

Note:

LU_ID specifies the identifier for the local LU under which the transaction program is executing. APPC/PC returns this value when the application subsystem identifies and initializes the LU by issuing the ATTACH_LU verb. For more information, see the description of the ATTACH_LU verb in Chapter 5, "Using Control Verbs."

USER_ID specifies the user identification field from the ALLOCATE or incoming ALLOCATE if conversation level security was specified.

LU Type 6.2 GET_TYPE parameters	APPC/PC GET_TYPE parameters
	TP_ID
RESOURCE	CONV_ID
	RETURN_CODE
TYPE	TYPE

LU Type 6.2 POST_ON_RECEIPT parameters	APPC/PC POST_ON_RECEIPT parameters
	TP_ID
RESOURCE	CONV_ID
FILL	FILL
LENGTH	MAX_LENGTH
	RETURN_CODE

LU Type 6.2 PREPARE_TO_RECEIVE parameters	APPC/PC PREPARE_TO_RECEIVE parameters
	TP_ID
RESOURCE	CONV_ID
TYPE	TYPE
LOCKS	LOCKS
RETURN_CODE	RETURN_CODE

LU Type 6.2 RECEIVE_IMMEDIATE parameters	APPC/PC RECEIVE_IMMEDIATE parameters
	TP_ID
RESOURCE	CONV_ID
FILL	FILL
LENGTH	MAX_LENGTH
RETURN_CODE	RETURN_CODE
	DATA_LENGTH
DATA	DATA_PTR DATA
WHAT_RECEIVED	WHAT_RECEIVED
REQUEST_TO_SEND_RECEIVED	REQUEST_TO_SEND_RECEIVED

LU Type 6.2 RECEIVE_AND_WAIT parameters	APPC/PC RECEIVE_AND_WAIT parameters
	TP_ID
RESOURCE	CONV_ID
FILL	FILL
LENGTH	MAX_LENGTH
RETURN_CODE	RETURN_CODE
	DATA_LENGTH
DATA	DATA_PTR DATA
WHAT_RECEIVED	WHAT_RECEIVED
REQUEST_TO_SEND_RECEIVED	REQUEST_TO_SEND_RECEIVED

In the *SNA Transaction Programmer's Reference Manual for LU Type 6.2*, the LENGTH parameter supplies the length of the buffer when the program issues the request. This parameter contains the actual number of bytes received when control is returned. APPC/PC uses the MAX_LENGTH parameter to indicate the size of the buffer and the DATA_LENGTH parameter to indicate the actual number of bytes received.

DATA_PTR specifies the address of the buffer which is to contain the received data.

LU Type 6.2 REQUEST_TO_SEND parameters	APPC/PC REQUEST_TO_SEND parameters
	TP_ID
RESOURCE	CONV_ID
	RETURN_CODE

LU Type 6.2 SEND_DATA parameters	APPC/PC SEND_DATA parameters
	TP_ID
RESOURCE	CONV_ID
DATA	DATA
LENGTH	DATA_LENGTH
RETURN_CODE	RETURN_CODE
REQUEST_TO_SEND_RECEIVED	REQUEST_TO_SEND_RECEIVED

LU Type 6.2 SEND_ERROR parameters	APPC/PC SEND_ERROR parameters
	TP_ID
RESOURCE	CONV_ID
TYPE	TYPE
	SENSE
LOG_DATA	LOG_DATA
RETURN_CODE	RETURN_CODE
REQUEST_TO_SEND_RECEIVED	REQUEST_TO_SEND_RECEIVED

LU Type 6.2 TEST parameters	APPC/PC TEST parameters
	TP_ID
RESOURCE	CONV_ID
TEST	TEST
RETURN_CODE	RETURN_CODE

LU Type 6.2 WAIT parameters	APPC/PC WAIT parameters
	TP_ID
RESOURCE_LIST	CONV_ID_LIST
RETURN_CODE	RETURN_CODE
RESOURCE_POSTED	CONV_POSTED

Mapped Conversation Verbs

This section contains the cross-reference lists for mapped conversation verbs and parameters.

Cross-Reference for Mapped Conversation Verbs

The following table provides a cross-reference for Mapped Conversation verbs:

LU Type 6.2 Architecture	APPC/PC Implementation
BACKOUT	
GET_TYPE	GET_TYPE
MC_ALLOCATE	MC_ALLOCATE
MC_CONFIRM	MC_CONFIRM
MC_CONFIRMED	MC_CONFIRMED
MC_DEALLOCATE	MC_DEALLOCATE
MC_FLUSH	MC_FLUSH
MC_GET_ATTRIBUTES	MC_GET_ATTRIBUTES
MC_POST_ON_RECEIPT	
MC_PREPARE_TO_RECEIVE	MC_PREPARE_TO_RECEIVE
MC_RECEIVE_AND_WAIT	MC_RECEIVE_AND_WAIT
MC_RECEIVE_IMMEDIATE	MC_RECEIVE_IMMEDIATE
MC_REQUEST_TO SEND	MC_REQUEST_TO_SEND
MC_SEND_DATA	MC_SEND_DATA
MC_SEND_ERROR	MC_SEND_ERROR
MC_TEST	MC_TEST
SYNCPT	
WAIT	

Cross-Reference for Mapped Conversation Verb Parameters

The following tables provide a cross-reference for Mapped Conversation verb parameters.

LU Type 6.2 MC_ALLOCATE parameters	APPC/PC MC_ALLOCATE parameters
	TP_ID
LU_NAME	PARTNER_LU_NAME
MODE_NAME	MODE_NAME
TPN	TPN
RETURN_CONTROL	RETURN_CONTROL
SYNC_LEVEL	SYNC_LEVEL
SECURITY	SECURITY
PIP	PIP
RESOURCE	CONV_ID
RETURN_CODE	RETURN_CODE

Note:

APPC/PC does not support the APPC RETURN_CONTROL(DELAYED_ALLOCATION_PERMITTED)

option. It does, however, support an additional option: RETURN_CONTROL(WHEN_SESSION_FREE).

The only synchronization level values that APPC/PC supports are NONE and CONFIRM.

APPC/PC provides three security options: NONE, SAME, and PGM.

LU Type 6.2 MC_CONFIRM parameters	APPC/PC MC_CONFIRM parameters
	TP_ID
RESOURCE	CONV_ID
RETURN_CODE	RETURN_CODE
REQUEST_TO_SEND_RECEIVED	REQUEST_TO_SEND_RECEIVED

LU Type 6.2 MC_CONFIRMED parameters	APPC/PC MC_CONFIRMED parameters
	TP_ID
RESOURCE	CONV_ID
	RETURN_CODE

LU Type 6.2 MC_DEALLOCATE parameters	APPC/PC MC_DEALLOCATE parameters
	TP_ID
RESOURCE	CONV_ID
TYPE	TYPE
RETURN_CODE	RETURN_CODE

Note:

APPC/PC does not support deallocation TYPE(LOCAL).

LU Type 6.2 MC_FLUSH parameters	APPC/PC MC_FLUSH parameters
	TP_ID
RESOURCE	CONV_ID
	RETURN_CODE

LU Type 6.2 MC_GET_ATTRIBUTES parameters	APPC/PC MC_GET_ATTRIBUTES parameters
	TP_ID
RESOURCE	CONV_ID
	RETURN_CODE
	LU_ID
LUW_IDENTIFIER	
OWN_FULLY_QUALIFIED_ LU_NAME	OWN_NET_ID_NAME OWN_LU_NAME
PARTNER_LU_NAME	PARTNER_LU_NAME
PARTNER_FULLY_QUALIFIED_ LU_NAME	PARTNER_FULLY_QUALIFIED_ LU_NAME
MODE_NAME	MODE_NAME
SYNC_LEVEL	SYNC_LEVEL
SECURITY_USER_ID	USER_ID

Note:

LU_ID specifies the identifier for the local LU under which the transaction program is executing. APPC/PC returns this value when the application subsystem identifies and initializes the LU by issuing the ATTACH_LU verb. For more information, see the description of the ATTACH_LU verb in Chapter 5, "Using Control Verbs."

USER_ID specifies the user identification field from the ALLOCATE or incoming ALLOCATE, if conversation level security was used.

LU Type 6.2 MC_PREPARE_TO_RECEIVE parameters	APPC/PC MC_PREPARE_TO_RECEIVE parameters
	TP_ID
RESOURCE	CONV_ID
TYPE	TYPE
LOCKS	LOCKS
RETURN_CODE	RETURN_CODE

Note:

TYPE(SYNC_LEVEL) is equivalent to TYPE(CONFIRM) if ALLOCATE SYNC_LEVEL is CONFIRM.

LU Type 6.2 MC_RECEIVE_AND_WAIT parameters	APPC/PC MC_RECEIVE_AND_WAIT parameters
	TP_ID
RESOURCE	CONV_ID
LENGTH	MAX_LENGTH
RETURN_CODE	RETURN_CODE
	DATA_LENGTH
DATA	DATA_PTR DATA
WHAT_RECEIVED	WHAT_RECEIVED
REQUEST_TO_SEND_RECEIVED	REQUEST_TO_SEND_RECEIVED
MAP_NAME	

Note:

In the *SNA Transaction Programmer's Reference Manual for LU Type 6.2* for MC_RECEIVE_AND_WAIT, the LENGTH parameter supplies the length of the buffer when the program issues the request. This parameter contains the actual number of bytes received when control is returned. APPC/PC uses the MAX_LENGTH parameter to indicate the size of the buffer and the DATA_LENGTH parameter to indicate the actual number of bytes received.

LU Type 6.2 MC_RECEIVE_IMMEDIATE parameters	APPC/PC MC_RECEIVE_IMMEDIATE parameters
	TP_ID
RESOURCE	CONV_ID
LENGTH	MAX_LENGTH
RETURN_CODE	RETURN_CODE
	DATA_LENGTH
REQUEST_TO_SEND_RECEIVED	REQUEST_TO_SEND_RECEIVED
DATA	DATA_PTR DATA
WHAT_RECEIVED	WHAT_RECEIVED
MAP_NAME	

LU Type 6.2 MC_REQUEST_TO_SEND parameters	APPC/PC MC_REQUEST_TO_SEND parameters
	TP_ID
RESOURCE	CONV_ID
	RETURN_CODE

LU Type 6.2 MC_SEND_DATA parameters	APPC/PC MC_SEND_DATA parameters
	TP_ID
RESOURCE	CONV_ID
DATA	DATA
LENGTH	DATA_LENGTH
MAP_NAME	
FMH_DATA	
RETURN_CODE	RETURN_CODE
REQUEST_TO_SEND_RECEIVED	REQUEST_TO_SEND_RECEIVED

LU Type 6.2 MC_SEND_ERROR parameters	APPC/PC MC_SEND_ERROR parameters
	TP_ID
RESOURCE	CONV_ID
RETURN_CODE	RETURN_CODE
REQUEST_TO_SEND_RECEIVED	REQUEST_TO_SEND_RECEIVED

LU Type 6.2 MC_TEST parameters	APPC/PC MC_TEST parameters
	TP_ID
RESOURCE	CONV_ID
TEST	TEST
RETURN_CODE	RETURN_CODE

Note:

APPC/PC does not support MC_TEST(POSTED).

Control Verbs

This section contains the cross-reference lists for control operator verbs and parameters.

Cross-Reference for Control Verbs to APPC/PC Verbs

LU Type 6.2 Architecture	APPC/PC Implementation
ACTIVATE_SESSION	
CHANGE_SESSION_LIMIT	
INITIALIZE_SESSION_LIMIT RESET_SESSION_LIMIT PROCESS_SESSION_LIMIT	CNOS
DEACTIVATE_SESSION	
DEFINE	ATTACH_LU
DISPLAY	DISPLAY
	ATTACH_PU
	DETACH_PU
	DETACH_LU
	CHANGE_LU
	ACTIVATE_DLC
	CREATE_TP
	TP_STARTED
	TP_ENDED
	GET_ALLOCATE
	TP_VALID
	SYSLOG
	ACCESS_LU_LU_PW

Cross-Reference for Control Verbs to APPC/PC Verb Parameters

LU Type 6.2 INITIALIZE_SESSION_LIMIT, RESET_SESSION_LIMIT and PROCESS_SESSION_LIMIT parameters (Note: The above verbs will be abbreviated below as ISL, RSL, and PSL respectively.)		**APPC/PC CNOS parameters**
		LU_ID
LU_NAME	ISL,RSL,PSL	PARTNER_LU_NAME
MODE_NAME	ISL,RSL	MODE_NAME_SELECT
		SET_NEGOTIABLE
LU_MODE_SESSION_LIMIT	ISL	PARTNER_LU_MODE_SESSION_ LIMIT
MIN_CONWINNERS_SOURCE	ISL	MIN_CONWINNERS_SOURCE
MIN_CONWINNERS_TARGET	ISL	MIN_CONWINNERS_TARGET
		AUTO_ACTIVATE
RESPONSIBLE	RSL	RESPONSIBLE
DRAIN_SOURCE	RSL	DRAIN_SOURCE
DRAIN_TARGET	RSL	DRAIN_TARGET
RETURN_CODE	ISL,RSL,PSL	RETURN_CODE

Note:

AUTO_ACTIVATE specifies the number of contention winner sessions that APPC/PC automatically activates before any transaction program ALLOCATE requests occur.

SET_NEGOTIABLE specifies whether the PARTNER_LU_MODE_SESSION_LIMIT specified in the CNOS verb will also be used to override the current settings for MODE_MAX_NEGOTIABLE_SESSION_LIMIT, as given in the ATTACH_LU verb (or previously overridden by an earlier CNOS verb with this parameter set to YES). In this case, a normal CNOS negotiation still takes place. If the CNOS verb has no parameter errors in it, the new value takes effect as the local value for this and future negotiations. The partner LU (in the parallel session case) can still negotiate the suggested values downwards.

Network Management Verb

LU Type 6.2 Architecture	APPC/PC Implementation
	TRANSFER_MS_DATA

Other APPC/PC Services

The following table list verbs not architecturally defined:

LU Type 6.2 Architecture	APPC/PC
	CONVERT
	PASSTHROUGH
	SET_PASSTHROUGH
	TRACE

Appendix H. ASCII/EBCDIC Translation Tables

This appendix describes the three kinds of ASCII/EBCDIC conversion tables you can specify for use by the CONVERT verb (for information on the CONVERT verb, see Chapter 9, "Other APPC/PC Services." The type A and type AE tables are internal to APPC/PC. The type G table is user-supplied; its file name is specified on the APPC/PC configuration menus. The conversion table types correspond to the symbol string types A, AE, and G described in the *SNA Transaction Programmer's Reference Manual for LU Type 6.2.* For information on the APPC/PC configuration menus, refer to the *APPC/PC Installation and Configuration Guide.*

The type A table converts characters including uppercase A-Z; numerics 0-9; and special characters $, #, and @. This table also converts lowercase ASCII characters to uppercase EBCDIC characters. When you specify this table, APPC/PC requires the first character in each string to be an uppercase letter or one of the three special characters. The string can contain trailing blanks.

The type AE table converts characters to the same characters as the type A table but also converts lowercase a-z and the period (.). APPC/PC does not place any restrictions on the first character of the string, and the string can contain trailing blanks.

The type G table is a user-defined table that converts any character. This appendix includes a sample of a type G table that you can edit to satisfy the specific conversion requirements of your program. The APPC/PC diskette includes a file named APPCGTAB.DAT, containing this sample type G table.

The format of a conversion table consists of 32 lines of 32 characters each. Each line represents 16 "printable" hexadecimal characters followed by a carriage return and line feed. The first 16 lines provide the information for ASCII to EBCDIC conversion and the second 16 lines provide the information for EBCDIC to ASCII conversion. The table must include all 32 lines.

When APPC/PC performs a conversion, it uses the numeric equivalent of each incoming character as a 0-origin index into the conversion table. This index specifies the table location containing the hexadecimal value of the converted character. For example, assume that the 48th position in the table contains a value of X'F0'. APPC/PC will then convert incoming characters with a value of 48 (X'30') to a value of 240 (X'F0').

- TABLE A

```
00000000000000000000000000000000
00000000000000000000000000000000
40000007B5B000000000000000000000
F0F1F2F3F4F5F6F7F8F9000000000000
7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6
D7D8D9E2E3E4E5E6E7E8E90000000000
00C1C2C3C4C5C6C7C8C9D1D2D3D4D5D6
D7D8D9E2E3E4E5E6E7E8E90000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
20000000000000000000000000000000
00000000000000000000002400000000
00000000000000000000000000000000
00000000000000000000002340000000
00616263646566676869000000000000
006A6B6C6D6E6F707172000000000000
000073747576777879 7A000000000000
00000000000000000000000000000000
00414243444546474849000000000000
004A4B4C4D4E4F505152000000000000
00005354555657585 95A000000000000
00000000000000000000000000000000
```

- TABLE AE

```
00000000000000000000000000000000
00000000000000000000000000000000
4000007B5B0000000000000000004B00
F0F1F2F3F4F5F6F7F8F9000000000000
7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6
D7D8D9E2E3E4E5E6E7E8E900000000000
00818283848586878889919293949596
979899A2A3A4A5A6A7A8A900000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
20000000000000000000002E00000000
00000000000000000000002400000000
00000000000000000000000000000000
00000000000000000000002340000000
00616263646566676869000000000000
006A6B6C6D6E6F707172000000000000
0000737475767778797A000000000000
00000000000000000000000000000000
00414243444546474849000000000000
004A4B4C4D4E4F505152000000000000
0005353455565758595A000000000000
303132333435363738390000000000000
```

- TABLE G

The following is a properly formatted sample of the type G table you can supply for APPC/PC:

```
00010203372D2E2F1605250B0C0D0E0F
101112133C3D322618193F27221D351F
405A7F7B5B6C507D4D5D5C4E6B604B61
F0F1F2F3F4F5F6F7F8F97A5E4C7E6E6F
7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6
D7D8D9E2E3E4E5E6E7E8E9ADE0BD5F6D
7981828384858687888991929394959 6
979899A2A3A4A5A6A7A8A9C06AD0A107
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
00010203FF09FF7FFFFFFF0B0C0D0E0F
10111213FFFF08FF1819FFFFFF1DFF1F
FFFF1CFFFF0A171BFFFFFFFFFF050607
FFFF16FFFF1EFF04FFFFFFFF1415FF1A
20FFFFFFFFFFFFFFFFFFFF2E3C282BFF
26FFFFFFFFFFFFFFFFFF21242A293B5E
2D2FFFFFFFFFFFFFFFFF7C2C255F3E3F
FFFFFFFFFFFFFFFFFF603A2340273D22
FF6162636465666768 69FFFFFFFFFFF
FF6A6B6C6D6E6F707172FFFFFFFFFFFF
FF7E737475767778797AFFFFFF5BFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFF5DFFFF
7B41424344454647484 9FFFFFFFFFFF
7D4A4B4C4D4E4F505152FFFFFFFFFFFF
5CFF535455565758595AFFFFFFFFFFFF
3031323334353637383 9FFFFFFFFFFF
```

Appendix I. Statement of Service

IBM will provide service for valid program-related defects in the IBM APPC/PC program product at no additional charge. Program service is available until June 30, 1987. However, service will be provided only for the current update level.

The way each customer obtains access to program service depends on the marketing channel through which the program was obtained.

In the United States and Puerto Rico, if the IBM APPC/PC program product was obtained through an authorized IBM Personal Computer dealer, requests for program service should be made through the dealer. If the program was obtained through the IBM National Accounts Division or the IBM National Marketing Division, requests for program service should be made through the service coordinator for the customer.

The IBM Support Center will help the service coordinator diagnose and solve problems. The service coordinator may call the IBM Support Center at any time, and will usually be called back within eight business hours. The IBM Support Center will contact the service coordinator Monday through Friday between 8 a.m. and 5 p.m., local customer time.

IBM does not guarantee service results, or that the program will be error-free, or that all program defects will be corrected.

When a report of a defect in an unaltered portion of a supported release of the program is submitted, IBM will respond by issuing one of the following:

- Defect correction information, such as corrected documentation, corrected code, or notice of availability of corrected code
- A restriction notice
- A bypass.

Corrected code is provided on a cumulative basis on diskettes; no source code is provided. Only one copy of the corrections with supporting documentation will be issued to the customer coordinator or authorized dealer. IBM will authorize various agents, such as IBM Personal Computer dealers and service coordinators of IBM National Accounts Division and IBM National Marketing Division customers, to make and distribute a copy of the corrections, if needed, to each IBM APPC/PC user that they serve.

The total number of copies of an update distributed to IBM APPC/PC users within a customer's location may not exceed the number of copies of the IBM APPC/PC program products that the customer has acquired.

IBM does not plan to release updates of IBM APPC/PC on a routine basis for preventive service purposes. However, should IBM determine that there is a general need for a preventive service update, it will be made available to all users through the same process used to distribute general IBM APPC/PC updates.

Following the discontinuance of all program services, this program will be distributed on an "as is" basis, without warranty of any kind either express or implied.

Glossary

abnormal termination. A system or program failure, or operator action that causes a program (APPC/PC, application subsystem or transaction program) to end unsuccessfully. See normal termination.

adapter. An IBM communications adapter interface card for the IBM Personal Computer.

Advanced Program-to-Program Communication (APPC). A set of protocols that provide communication capabilities between computer programs on diverse systems.

Advanced Program-to-Program Communication for the IBM Personal Computer (APPC/PC). A DOS extension that performs APPC functions for transaction programs running on an IBM PC.

alert. The primary network management message that communicates problem determination information to the network operator.

American National Standard Code for Information Interchange (ASCII). The standard interchange between data processing systems, data communications systems, and associated equipment. The code uses a coded character set consisting of 7-bit coded characters (8 bits including parity check). The set consists of control characters and graphic characters.

API. Application Program Interface.

APPC. Advanced Program-to-Program Communication.

APPC/PC. Advanced Program-to-Program Communication for the IBM Personal Computer.

application program. A program that processes transactions for a specific purpose. Also called a transaction program.

Application Program Interface (API). The set of commands, described in this manual, that the application subsystem and transaction programs use to communicate with APPC/PC.

application subsystem. A user-supplied set of programs that provide services to APPC/PC and transaction programs.

ASCII. American National Standard Code for Information Exchange.

basic conversation. A conversation between two transaction programs using the APPC/PC basic conversation API. in typical situations, service transaction programs use basic conversations and end-user transaction programs use mapped conversations. however, either type of program may use either type of conversation.

basic conversation verb. A verb that a transaction program issues when using the APPC/PC basic conversation API.

bind. The RU involved in activating an LU6.2 session.

bind session (BIND). In APPC, the process that APPC performs to establish a session between two LUs.

boundary function. Services provided to a peripheral node in an SNA network, by an intermediate or host node. These services include message re-formatting and address translation.

byte reversal. The IBM Personal Computer stores all numeric 16- or 32-bit values with the low-order (least significant) byte stored in the lower-numbered address. This format is the reverse of that used in larger IBM computer systems.

change-direction protocol. A data flow control protocol in which the sending logical unit stops sending requests, signals this fact to the receiving LU, and prepares to receive requests.

CICS/VS. Customer Information Control System for Virtual Storage.

class of service. A designation of the path control network characteristics, such as path security, transmission priority, and bandwidth, that applies to a particular session. The end-user program specifies the class of service when requesting a session by using a symbolic name that APPC maps into a list of virtual routes, any one of which can provide the requested level of service for the session.

communication adapter. Hardware that enables a processor to perform data communication.

configuration services (CS). One of the types of network services in a control point (SSCP or PNCP) and in the physical unit (PU). Configuration services activate, deactivate, and maintain the status of physical units, links, and link stations.

confirmation. A program's acknowledgment that the data has been received.

contention loser. The LU that must request and receive permission from the session partner LU to allocate a session.

contention-loser polarity. The designation that an LU is the contention loser for a session.

contention winner. The LU that can allocate a session without requesting permission from the session partner LU.

contention-winner polarity. The designation that an LU is the contention winner for a session.

control point. Refers to a system services control point (SSCP).

control verb. One of the verbs a program issues to set up the IBM PC hardware and APPC/PC software to perform transactions with a remote program. Typically, an application subsystem issues these verbs and a transaction program issues conversation verbs.

conversation. The communication between two transaction programs communicating over a Type 6.2 LU-LU session.

conversational transaction. Two or more programs communicating with each other using the services of LUs. Among the components in a conversational transaction are the transaction programs and the transaction resources associated with the programs (including the conversations connecting the transaction programs).

conversation type. A conversation can be either a basic conversation or a mapped conversation.

conversation verb. One of the verbs a transaction program issues to perform transactions with a remote program. Typically, a transaction program issues these verbs and an application subsystem issues control verbs.

CS. Configuration services.

Customer Information Control System for Virtual Storage (CICS/VS). A host (System/370) program product that can be used in a communications network.

data link. See *link.*

data link control (DLC) layer. The SNA layer that consists of the link stations that schedule data transfer over a link between two nodes and perform error control for the link. Examples of data link control are SDLC for serial-by-bit link connection and data link control for the IBM Token-Ring Network.

deadlock. A situation that occurs when two or more nodes are waiting for messages from each other and cannot continue processing verbs.

dependent LU. An LU with an active session to an LU within an SNA host system. A dependent LU cannot send BINDs.

distributed transaction processing services. The set of services that enable transaction programs to communicate with each other and access remote resources, and to aid in synchronization and error recovery.

DLC. Data link control.

EBCDIC. Extended Binary-Coded Decimal Interchange Code.

error log exit. A routine that APPC/PC can call to log errors during a conversation.

Extended Binary-Coded Decimal Interchange Code (EBCDIC). A coded character set consisting of 256 eight-bit characters.

flow control. The process of managing the rate at which data traffic passes between components of the network. Flow control optimizes the rate of flow of message units to minimize congestion in the network; that is, to neither overflow the buffers at the receiver or at intermediate routing nodes, nor leave the receiver waiting for more message units. See also *pacing*.

FMH. Function management header.

function management header (FMH). An optional field at the beginning of a request unit that carries certain logical unit control information. Different types of logical units use different types of FM headers:

Type 6.2 logical units use three types of headers:

- An Attach FM header (FMH 5) to specify the name of required characteristics of a partner transaction program
- An Error-description FM header (FMH 7) to describe a transaction program error or an ATTACH failure
- A security FM header (FMH 12) to carry LU-LU session password verification data.

GDS. The General Data Stream data format. (See *SNA Reference Summary* for more information.)

half-session. A component that provides data flow control and transmission control for one of the sessions of a network addressable unit. See also *session, primary half-session* and *secondary half-session.*

hang. See deadlock.

hexadecimal. Pertaining to a numbering system with a base of 16. Valid numbers are the digits 0 through 9 and the characters A through F, where A represents 10 and F represents 15.

host node. A subarea node that contains a system services control point (SSCP); for example, a system/370 computer with OS/VS2 and VTAM.

independent LU. An LU that does not have an active session to an LU within an SNA host system. An independent LU can send BINDs.

intermediate node. A node that provides intermediate routing services in an SNA network.

layer. A grouping of related functions that are logically separate from the functions in other layers; the implementation of the functions in one layer can be changed without affecting function in other layers.

link. The line connection and the link stations joining network or peer nodes.

link connection. The physical equipment providing two-way communication between one link station and other link stations.

link station. The hardware and software that enables a node to attach to and provide control for a link. See also *primary link station* and *secondary link station.*

local transaction program. The transaction program at the local LU.

logical unit. A set of logical services by which one user communicates with another, using sessions.

Logical unit Type 6.2. The architectural base for APPC. An LU Type 6.2 supports sessions between two applications in a distributed data processing environment.

LU. Logical unit.

LU 6.2. Logical unit 6.2.

mapped conversation. A conversation between two transaction programs using the APPC/PC mapped conversation API. In typical situations, end-user transaction programs use mapped conversations and service transaction programs use basic conversations. However, either type of program may use either type of conversation.

mapped conversation verb. A verb that a transaction program issues when using the APPC/PC mapped conversation API.

mode. The set of parameters defining the network properties of a session.

modem. A mechanism that modulates and demodulates signals transmitted over data communications facilities.

mode name. A name that a program uses to request a specific set of network properties of a session the program wants to use for a conversation. These properties include, for example, the highest synchronization level for

conversations on the sessions, the class of service for the sessions, and the session routing and delay characteristics. The network administrator establishes the mode names for a network.

multipoint line. A communication line or circuit interconnecting several nodes. Contrast with point-to-point line.

NAU. Network addressable unit.

negotiable BIND. An RU that can enable two LU-LU half-sessions to negotiate the parameters of a session when the LUs are activating a session.

network addressable unit (NAU). A logical unit, physical unit, or system services control point. An NAU is the origin or the destination of information transmitted through the path control network. See also *network name.*

network management services function. A set of programs to manage a network. The programs receive network management data through NMVTs.

network management verb. The TRANSFER_MS_DATA verb is used to provide network management information to a network management services function within an SNA network.

network name. The symbolic identifier by which the network refers to a network addressable unit (NAU), a link station, or a link.

NMVT. A Network Management Vector Transport RU provides alert, problem determination statistics, and other network management data to a network management services function.

node. An endpoint of a link or a junction common to two or more links in a network. Nodes can be host processors, communication controllers, or terminals. Nodes can vary in routing and other functional capabilities.

nonswitched line. A connection between systems or devices that does not have to be made by dialing. Contrast with switched line.

normal termination. Termination that results in successful execution of a program. Normal termination of APPC/PC is through successful execution of the APPCUNLD command. See abnormal termination.

NS header. The part of a network services RU that identifies the type of RU.

pacing. The technique of limiting the amount of data that a program can send or receive at one time to prevent overrunning the LU buffers. See also *flow control, receive pacing,* and *send pacing.*

pacing window size. The number of RUs that a program can send before getting permission to send more.

parallel sessions. Two or more concurrently active sessions between the same two logical units (LUs). Each session can have different session parameters.

partner transaction programs. Transaction programs corresponding on the same conversation.

path control network. The routing portion of an SNA network.

PD Stats. The problem determination statistics that enable the SNA network management services function to determine and diagnose problems associated with the communication links used for sessions.

peer-to-peer. Communication between two LUs that is not managed by a host.

peripheral node. A node that has no intermediate routing function, and is dependent upon an intermediate or host node to provide certain network services for its dependent LUs. APPC/PC provides a peripheral node capability.

physical unit (PU). The component that manages and monitors the resources of a node, as requested by an SSCP using an SSCP-PU session. Each node of an SNA network contains a physical unit.

physical unit (PU) services. The components within a physical unit (PU) that provide configuration services and maintenance services for SSCP-PU sessions.

PIP. Program initialization parameter.

PLU. Primary logical unit.

point-to-point line. A communication line or circuit that connects a single remote node to another node; it can be either switched or non-switched. Contrast with multipoint line.

polarity. See *contention-winner polarity* and *contention-loser polarity.*

posting. When a transaction program directs APPC/PC to post a conversation, APPC/PC enables the transaction program to check whether a specific amount of data is available in the local LU's receive buffer.

primary logical unit (PLU). The logical unit (LU) that contains the primary half-session for a particular LU-LU session. See also *secondary logical unit.*

primary half-session. The half-session on the node that sends the session activation request. See also *primary logical unit* and *secondary half-session.*

primary link station. The link station that is responsible for the control of the link. A link has only one primary link station. All traffic over the link is between the primary link station and a secondary link station. See also *secondary link station.*

PU. Physical unit.

receive pacing. The pacing of message units that a component is receiving. See also *send pacing.*

remote transaction program. The partner transaction program using the remote LU.

request/response header (RH). Control information, preceding a request/response unit (RU), that specifes the type of RU (request unit or response unit) and contains control information associated with that RU.

request/response unit (RU). A generic term for a request unit or a response unit.

request unit. A message unit that contains control information such as a request code, function management headers (FMHs), end-user data, or a combination of these types of information.

response unit. An LU's response to one or more request units, indicating successful receipt of data or an error condition.

responsible LU. The LU responsible for deactivating a session when it is no longer being used by two LUs for a conversation.

return code. A code APPC/PC returns to the issuer of a verb to indicate the results of verb execution.

RH. Request/response header.

RU. Request/response unit.

SDLC. See *synchronous data link control.*

secondary half-session. The half-session on the node that receives the session-activation request. See also *secondary logical unit* and *primary half-session.*

secondary link station. Any link station that is not the primary link station. A secondary link station can exchange data with the primary link station, but not with other secondary link stations. See also *primary link station.*

secondary logical unit (SLU). The logical unit (LU) that contains the secondary half-session for a particular LU-LU session. A logical unit may contain secondary and primary half-sessions for different active LU-LU sessions. See also *primary logical unit.*

send pacing. Pacing of message units that a component is sending. See also *receive pacing.*

sense code. The code that indicates the type of error that has occured. Sense codes are logged or sent to the partner node in function management headers or negative responses.

session. Communication between LUs. A logical connection between two network addressable units (NAUs). The connection can be activated, tailored to provide various protocols, and deactivated, as requested. See also *half-session*, *primary half-session*, and *secondary half-session.*

session activation. The process of exchanging a session activation request and a positive response between network addressable units (NAUs). See also *session deactivation.*

session deactivation. The process of exchanging a session deactivation request and response between network addressable units (NAUs). See also *session activation.*

session parameters. The parameters that specify or constrain the protocols (such as bracket protocol and pacing) for a session between two network addressable units (NAUs).

session partner. One of the two network addressable units (NAUs) participating in an active session.

single session. A session that is the only session connecting two LUs.

SLU. Secondary logical unit.

SNA. Systems network architecture.

SNASVCMG. The APPC/PC-defined mode, used only for CNOS negotiations.

SSCP. System services control point.

SSCP-LU session. A session between a system services control point (SSCP) and a logical unit (LU); the session enables the LU to request the SSCP to help initiate LU-LU sessions.

SSCP-PU session. A session between a system services control point (SSCP) and a physical unit (PU); SSCP-PU sessions enable SSCPs to send requests to and receive status information from individual nodes to control the network configuration.

state. The state of a conversation determines which verbs APPC/PC allows a program to issue. For example, in receive state the program can only receive data. The four states are reset, send, receive, and confirm.

switched line. A connection between two nodes that is established by dialing. Contrast with nonswitched line.

synchronization level. The specification indicating whether the corresponding transaction programs exchange confirmation requests and replies.

synchronous data link control (SDLC). A discipline for managing synchronous, code-transparent, serial-by-bit information transfer over a link connection. Transmission exchanges can be half-duplex over switched links, and full-duplex or half-duplex over non-switched links. The configuration of the link connection can be point-to-point, multipoint, or loop. SDLC conforms to subsets of the Advanced Data Communication Control Procedures (ADCCP) of the American National Standards Institute and High-Level Data Link Control (HDLC) of the International Standards Organization.

systems network architecture (SNA). The description of the logical structure, formats, protocols, and

operational sequences for transmitting information units through, and controlling the configuration and operation of, networks.

systems network architecture (SNA) network. The part of a user-application network that conforms to the formats and protocols of systems network architecture. It provides a reliable method of transferring data among programs and provides protocols for controlling the resources of various network addressable units (NAUs), boundary-function components, and the path control network.

Systems network architecture node. A node that supports SNA protocols. See also *node.*

systems services control point (SSCP). A control point in a host node. It provides network services for dependent nodes.

time-out condition. An error condition indicating that a specified amount of waiting time has elapsed without the occurrence of an expected event.

transaction. A unit of communication processing that accomplishes a particular result.

transaction program. An application program that performs transactions with one or more remote programs.

transaction service mode. See mode.

UNBIND. The RU involved in deactivating an LU Type 6.2 session.

verb. Usually, a request by a user program for an APPC/PC action. In a few cases, a request by APPC/PC for an application subsystem action. A verb requesting an APPC/PC action is performed by issuing a software interrupt accompanied by APPC/PC-specific information stored in registers and memory.

Virtual Telecommunications Access Method (VTAM). A set of programs that control communications

between nodes and application programs running on a host (System/370) system.

VTAM. Virtual Telecommunications Access Method.

Index

L

M

Q

R

S

T

U

V

W

Reader's Comment Form

Advanced Program-to-Program Communication for the IBM Personal Computer Programming Guide **61X3842**

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used for revisions.

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions regarding the IBM Personal Computer or programs for the IBM Personal Computer, or for requests for additional publications; this only delays the response. Instead, direct your inquiries or request to your authorized IBM Personal Computer dealer.

Comments:

Fold and tape
Please do not staple

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department E01
P.O. Box 12195
Research Triangle Park, N.C. 27709

Reader's Comment Form

Advanced Program-to-Program Communication for the IBM Personal Computer Programming Guide **61X3842**

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used for revisions.

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Please do not use this form for technical questions regarding the IBM Personal Computer or programs for the IBM Personal Computer, or for requests for additional publications; this only delays the response. Instead, direct your inquiries or request to your authorized IBM Personal Computer dealer.

Comments:

Fold and tape
Please do not staple

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department E01
P.O. Box 12195
Research Triangle Park, N.C. 27709

International Business
Machines Corporation
PO Box 12195
Research Triangle Park
North Carolina, 27709

Printed in the
United States of America

61X3842