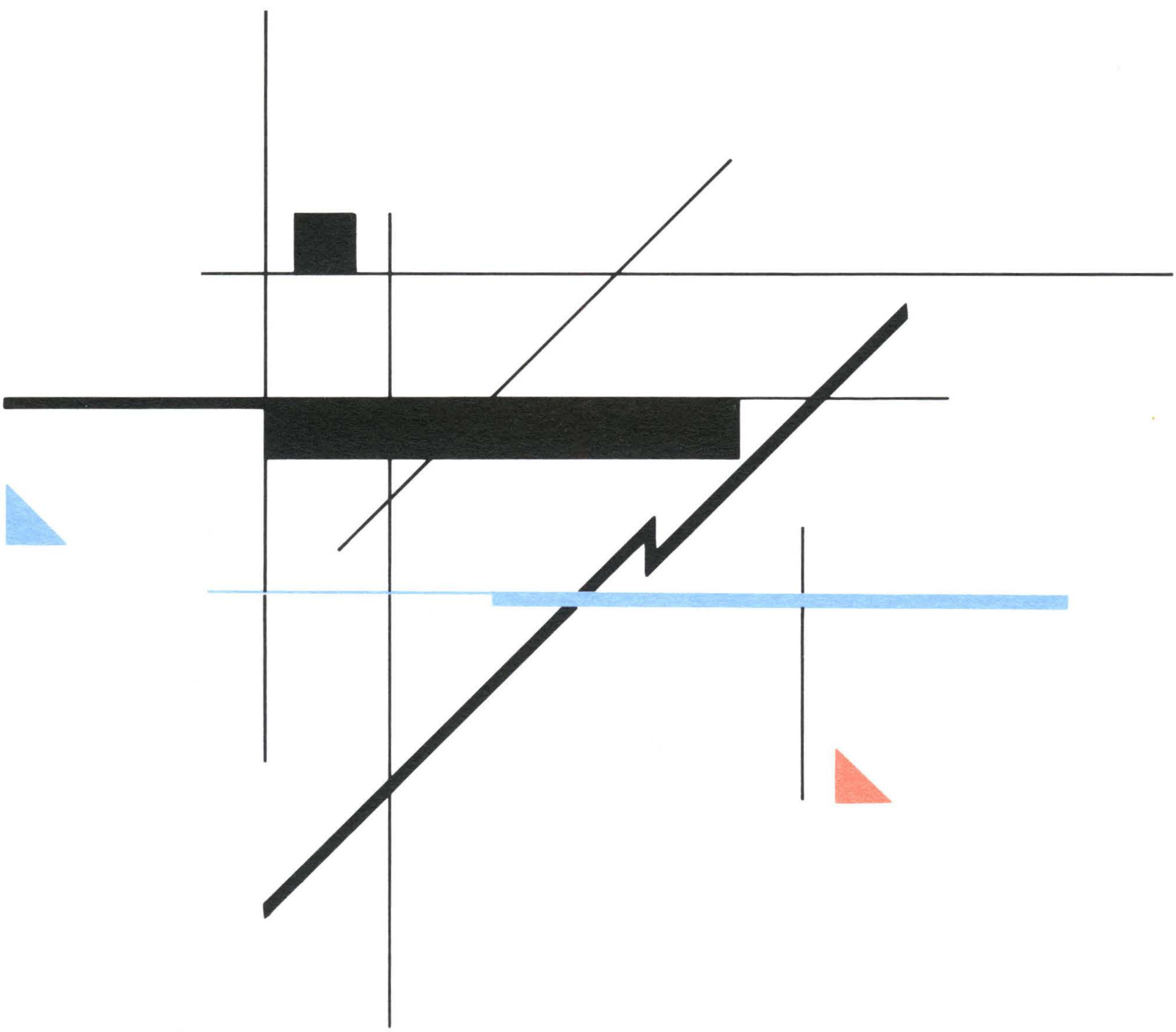
IBM

Systems Network Architecture

SC31-6808-0

LU 6.2 Reference: Peer Protocols



[redacted]

Systems Network Architecture

SC31-6808-0

LU 6.2 Reference: Peer Protocols

First Edition (September 1988)

Changes are made periodically to this publication; these changes will be incorporated into new editions of this publication. It is possible that this material may contain references to, or information about, IBM products (machines and programs) 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 or services in your country.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not of itself constitute or imply a grant of (i) any license under any patents, patent applications, trademarks, copyrights, or other similar rights of IBM or of any third party; or (ii) any right to refer to IBM in any advertising or other promotional or marketing activities. IBM assumes no responsibility for any infringement of patents or other rights that may result from use of the subject matter described in this document or for the manufacture, use, lease, or sale of machines or programs described herein, outside of any responsibilities assumed via the agreement for the purchase of IBM machines and the agreement for IBM licensed programs.

Licenses under IBM's utility patents are available on reasonable and nondiscriminatory terms and conditions. IBM does not grant licenses under its appearance design patents. Inquiries relative to licensing should be directed in writing to the IBM Director of Commercial Relations, International Business Machines Corporation, Armonk, New York, 10504.

The following sentence does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: International Business Machines provides this publication "As Is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. Within the United States, some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

Publications are not stocked at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Networking Architecture, Department E96, P.O. Box 12195, Research Triangle Park, North Carolina 27709, U.S.A. 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 is one of two books that describe, at the implementation level, the Systems Network Architecture (SNA) logical unit (LU) type 6.2 protocols. This book concerns the SSCP-independent LU 6.2 protocols (or peer protocols, not requiring mediation by a system services control point during LU-LU session initiation); the second book, SNA Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2, SC30-3269, concerns the SSCP-dependent LU 6.2 protocols (those protocols involving mediation by a system services control point during LU-LU session initiation). LU-LU protocols not related to session-initiation and -termination are common to both SSCP-dependent and -independent LU 6.2 protocols; these common protocols will be updated in the future only in this book, which therefore has precedence over the other book for information on these protocols.

This book does not describe any specific machines or programs that may implement SNA, nor does it describe any implementation-specific subsets or deviations from the architectural description that may appear within any IBM SNA product. These matters, as well as information on SNA product installation and system definition, are described in the appropriate publications for the particular IBM SNA machines or programs to be used.

The following books should be read in conjunction with this one.

COREQUISITE PUBLICATIONS

- SNA Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2, SC30-3269—reference information on SSCP-dependent protocols for LU 6.2.
- SNA Transaction Programmer's Reference Manual for LU Type 6.2, GC30-3084—reference information on LU type 6.2 verbs for programmers writing transaction programs to run on SNA.
- SNA Formats, GA27-3136—information on LU 6.2 and other SNA formats.

PREREQUISITE PUBLICATIONS

- SNA Concepts and Products, GC30-3072—basic information on SNA for those readers wanting either an overview or a foundation for further study.
- SNA Technical Overview, GC30-3073—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.

RELATED PUBLICATIONS

- SAA Common Programming Interface: Communications Reference, SC26-4399—description of Systems Application Architecture's[1] Communications Interface, which provides a high-level programming interface to LU 6.2.
- SNA Format and Protocol Reference Manual: Architectural Logic, SC30-3112—comprehensive information on the formats and protocols of SNA type 1, 2.0, 4, and 5 nodes.
- SNA—Sessions Between Logical Units, GC20-1868—reference information on SNA formats and protocols for LU types other than type 6.2.

---

[1] Systems Application Architecture is a trademark of International Business Machines Corporation.

- SNA Type 2.1 Node Reference (abbreviated T2.1 Node Reference), SC30-3422—reference information on type 2.1 node protocols.
- SNA Format and Protocol Reference Manual: Distribution Services, SC30-3098--reference information on formats and protocols for SNA Distribution Services.
- Document Interchange Architecture--Concepts and Structures, SC23-0759--reference information on Document Interchange Architecture.

CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1. INTRODUCTION

CHAPTER 2. OVERVIEW OF THE LU

CHAPTER 3. LU RESOURCES MANAGER

CHAPTER 4. LU SESSION MANAGER

CHAPTER 5.0. OVERVIEW OF PRESENTATION SERVICES

CHAPTER 5.1. PRESENTATION SERVICES--CONVERSATION VERBS

CHAPTER 5.2. PRESENTATION SERVICES--MAPPED CONVERSATION VERBS

CHAPTER 5.3. PRESENTATION SERVICES--SYNC POINT SERVICES VERBS

CHAPTER 5.4. PRESENTATION SERVICES--CONTROL-OPERATOR VERBS

CHAPTER 6.0. HALF-SESSION

CHAPTER 6.1. DATA FLOW CONTROL

CHAPTER 6.2. TRANSMISSION CONTROL

APPENDIX A. NODE DATA STRUCTURES

APPENDIX B. BUFFER MANAGER

APPENDIX N. FSM NOTATION

APPENDIX T. TERMINOLOGY: ACRONYMS AND ABBREVIATIONS

This page intentionally left blank

CHAPTER 1. INTRODUCTION

USE AND ORGANIZATION OF THIS BOOK

This book, in conjunction with the companion books listed in the Preface, provides a formal definition of Systems Network Architecture (SNA). It is intended to complement individual SNA product publications, but not to describe individual product implementations of the architecture.

SNA logical unit type 6.2 (hereafter generally referred to as LU 6.2, or simply LU) is defined here in the form of a functionally layered system, represented by a formal description, that is decomposable into components called protocol machines. Protocol machines generate output sequences in response to input sequences, in accordance with fixed rules, or protocols, governing distinct information transfers into, out of, and within the system.

The protocol machine definition of SNA uses the following basic notions:

- Finite-state machines: A finite-state machine (FSM) is an abstract device having a finite number of states (memory) and a set of rules whereby the machine's responses (state transitions and output sequences) to all input sequences are well defined.

- Routing and checking logic: Routing and checking logic performs a mapping of inputs (message units and FSM states) into outputs. It is used to verify validity of message units and to route them to FSMs.

- Block diagrams: A block diagram represents the decomposition of a protocol machine into its component submachines (which themselves are protocol machines) and the signaling paths between them. Each block in the diagram can be further decomposed into its constituent submachines.

- Protocol boundaries: A protocol boundary is a specification of the format and content requirements imposed on the signals exchanged between protocol machines within the same node.

The remainder of the book presents details of the SNA formats and protocols for LU 6.2, arranged as follows:

- Chapter 2 provides an overview of the functions and structure of the LU, as well as the sequences and message units exchanged between two communicating LUs.

- Chapters 3 and 4 describe LU services manager components; these components attach transaction programs as requested, allocate sessions to transaction programs, and coordinate the activation and deactivation of sessions involving LUs.

- Chapters 5.0 through 5.4 describe the general structure and detailed functions of presentation services—in particular the execution logic for LU 6.2 verbs.

- Chapter 6.0 provides an overview of the half-session, while Chapters 6.1 and 6.2 describe the data flow control and transmission control protocols, respectively, within half-sessions.

- Appendix A describes the data structures used in the formal description and the relationships among the control blocks.

- Appendix B describes the basic functions of the buffer manager and its protocol boundary with the LU.

- Appendix N describes the basic concept of, and notation for, finite-state machines.

- Appendix T provides a comprehensive list of abbreviations and acronyms used in the book.

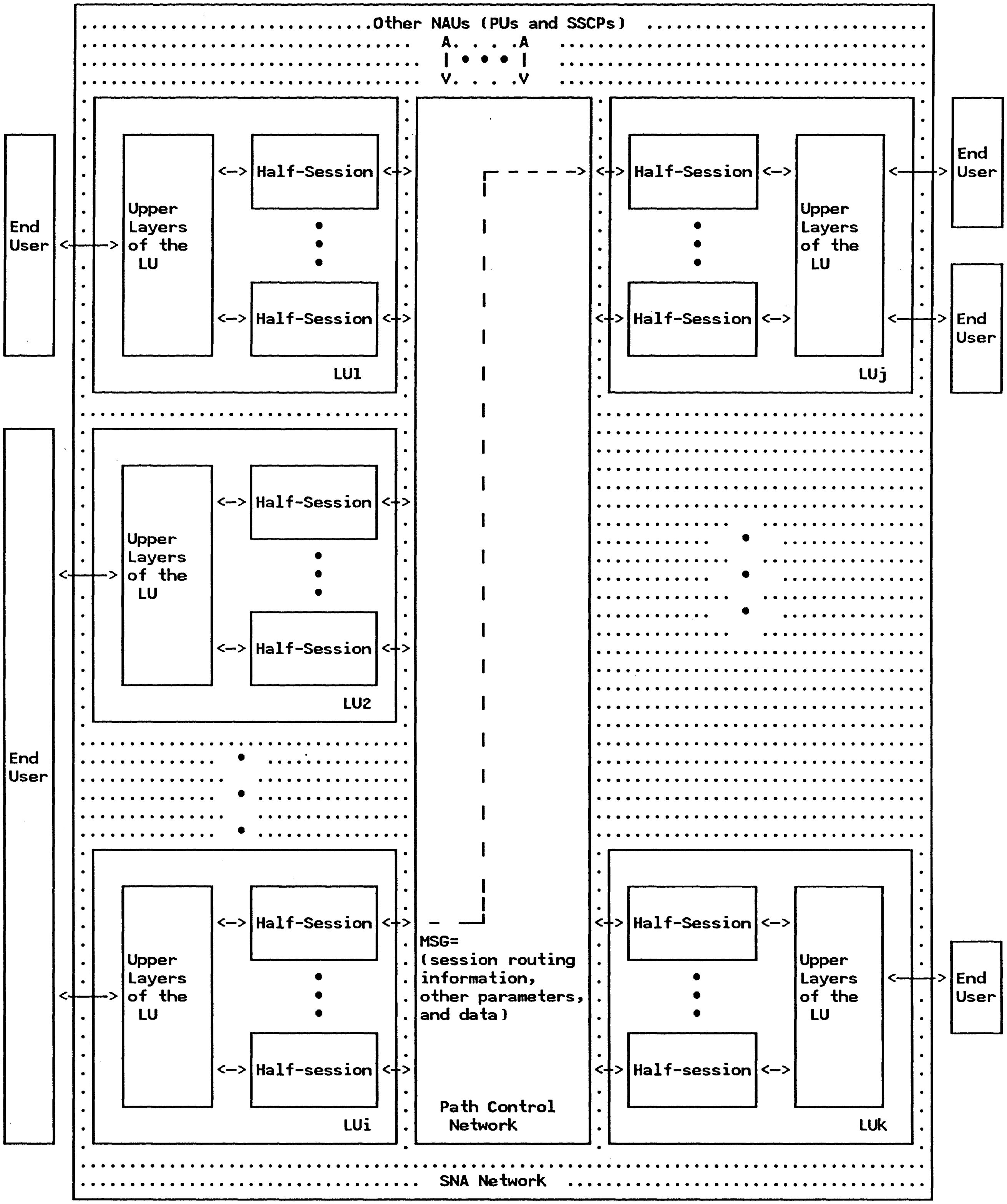


Figure 1-1. Overview of the SNA Network

DEFINITION OF AN SNA NETWORK

An SNA network:

- Enables the reliable transfer of data between end users (typically, terminal operators and application programs).
- Provides protocols for controlling the resources of any specific network configuration.

An SNA network consists logically of a set of network addressable units (NAUs) interconnected by an inner path control network consisting of the path control, data link control, and physical layers; Figure 1-1 on page 1-2 shows the general relationships. SNA networks functionally have a layered organization, the outermost layers of which form the NAUs, A NAU consists of the upper layers, transaction services (TS) and presentation services (PS), and one or more half-session protocol machines (consisting of the data flow control and transmission control layers), depending on the number of other NAUs with which it can be paired to form sessions.

Those NAUs serving end users are called logical units (LUs). An LU allows an end user to gain access to network resources (such as links, programs, and directories) and to communicate with other end users. An LU may also provide a service (such as for a control operator) wholly contained within the LU that is accessed from another LU via a session. Thus, in some cases, an LU-LU session has an end user only at one end. The presence of various services within an LU is a function of LU type, product design, and installation options.

In general, there need not be a one-to-one relationship between end users and LUs. The association between end users and the set of LUs is an implementation design option.

The LUs provide protocols allowing end users to communicate with each other and with other NAUs in the network. An LU can be associated with more than one network address (or with multiple, distinct local-form session identifiers); this allows two LUs (and therefore their end users) to form multiple, concurrently active sessions with each other.

Besides LUs, two other network addressable units are defined: physical units (PUs) and system services control points (SSCPs). These NAUs, in conjunction with one another, with control points (CPs) in T2.1 nodes, and with LUs, provide a variety of session, configuration, management, and network-operator services.

Message units are transported between NAUs by the path control network. These message units are of the general form:

MSG = (session routing information ,other parameters, and data) The path control network routes and delivers message units to naj in the same order as sent from nai.

The message units transferred within an SNA network generally have two components: end-user information and control information. The end-user information is passed by the SNA network and does not affect its state. Control information may sometimes be passed to the end users (as in the case of the Change Direction indication, which allows one end user to transfer the right to transmit data to the other); however, its main purpose is to change the state of the SNA network, thus effecting a normal control change (such as a change to a path control routing table) or a recovery from an exception condition.

NODES

The SNA network physically consists of nodes interconnected via links. An SNA node is a grouping of SNA-defined protocol machines. An SNA product node may consist of additional, product-specific protocol machines that use one or more SNA nodes. A user-application node may consist of additional, installation-defined protocol machines that use one or more SNA product nodes. These relationships are shown in Figure 1-2 on page 1-4. The abstraction of nested nodes is a useful reminder that each product exists in an environment that contains many design features that are not defined by SNA.

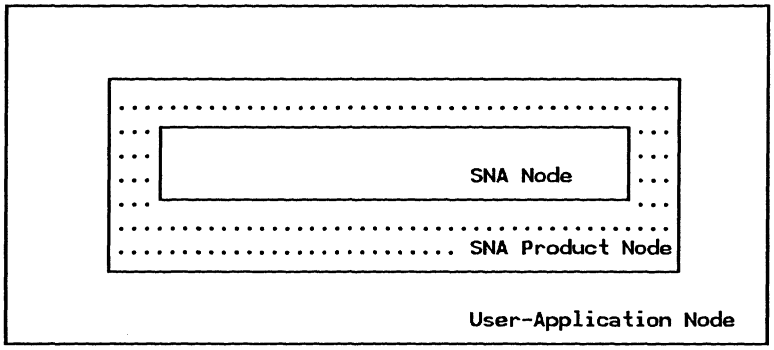
For specific details of nesting of SNA nodes and SNA product nodes within user-application nodes, see SNA Concepts and Products and SNA Technical Overview.

In this book, "node" is synonymous with "SNA node," and the qualifier will generally be omitted. Thus, end users and protocol machines not defined in SNA are external to the node, as that term is used hereafter.

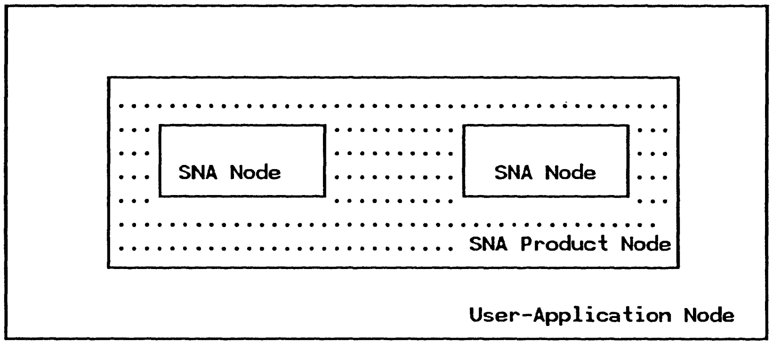
Various node types are defined in SNA: types 1, 2.0, 2.1, 4, and 5. They are distinguished by varying capabilities, such as for interconnection, and by the presence or absence of different NAU types.

For example, type 2.1 nodes can connect to the general subarea routing network or to other type 2.1 nodes directly. In the former case, subarea nodes (discussed below) provide general intermediate routing within the path control layer, allowing complex network configurations to be fashioned; in the latter case, two type 2.1 nodes can interconnect independently of other nodes, in a peer-to-peer relationship.

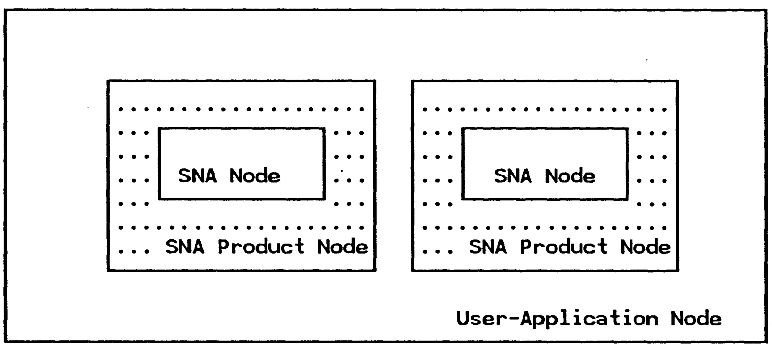
Type 1 and type 2 (i.e., 2.0 or 2.1) nodes are also referred to as peripheral nodes,



(a) Typical Case



(b) Two SNA Nodes within an SNA Product Node



(c) Two SNA Product Nodes within a User-Application Node

Figure 1-2. Examples of Nested Nodes

because they have limited addressing and path-control routing capabilities. They do not participate in the general network routing based on a global network address space. Instead, they depend on "boundary function" support in types 4 or 5 nodes to transform between the address forms, local to the peripheral nodes, and the network addresses used in the general routing portion of the path control network. Peripheral nodes are thereby insulated from changes in the global network address space resulting from reconfigurations.

Types 4 and 5 nodes are referred to as subarea nodes. (A subarea represents a partitioning of the network address space. It contains a subarea node and all the peripheral nodes attached to the subarea node.) Subarea nodes, besides also being sources and sinks of data, have more general path control capabilities. They can perform intermediate routing—passing message units received from one node on to another—and provide adaptive control of traffic flow within the subarea routing portion of the network.

NAUS AND NODE TYPES

Except for a T2.1 node, a node always includes a physical unit (PU), which controls the attached links and various other resources of the node. A PU has a type designation corresponding to the type (1, 2.0, 4, or 5) of node in which it resides. A T2.1 node includes the PU functions within its local control point (CP), described further below.

A node typically also includes logical units (LUs), through which end users attach to the node, and thus to the SNA network. From the vantage of this and the companion LU 6.2 book, node types 2.1 and 5 are of primary interest, as these are the only nodes that include LU 6.2 implementations. This book focuses on the SSCP-independent LU 6.2 protocols, and emphasizes interactions within the T2.1 node to support these peer protocols.

A subarea PU or subarea LU resides in a subarea node. A peripheral PU or peripheral LU resides in a peripheral node.

Type 5 nodes each contain a system services control point (SSCP). (Type 4 nodes do not—the primary architectural distinction between subarea node types.) An SSCP supports protocols for management and control of a domain. A domain consists of one SSCP and the PUs, LUs, links, and link stations that the SSCP can activate. Each PU, LU, link, and link station in a network belongs to one of the domains comprising the network, and some can belong to more than one domain—a feature referred to as "shared control." Each SSCP provides network services within its domain (basically for converting local names to global addresses) through protocols supported in conjunction with the PUs or LUs in the domain. The multiple SSCPs in a network jointly support network services across domains.

Type 2.1 (T2.1) nodes each contain a control point (CP), which provides services on a more local scale than an SSCP provides. In particular, a T2.1 CP can mediate LU-LU session-initiation requests (by doing partner-LU address look-up in its local data base) in the SSCP-independent LU 6.2 context just as an SSCP does in the SSCP-dependent LU 6.2 context.

THE PATH CONTROL NETWORK

The system consisting of all interconnected path control (PC) and data link control (DLC) components forms the path control network. The input/output streams of the path control network consist of streams of control information, such as addresses, and associated user data.

Each node has a PC element and NAUs. The node and link connections of the network, and the PC routing algorithms, combine to provide the following behavior for the path control network:

- An input to a PC element in node-i from a NAU is transmitted and routed by the path control network and emitted as output by the PC element in node-j to the destination NAU. (Since node-i and node-j can be the same node (i=j), NAUs within the same node can be connected by a session.)
- Message units with the same session identifiers are emitted by the path control network in the order submitted by the origin NAU.

Just as primary-secondary DLC asymmetries and other DLC details are hidden from PC, so the routing and other concerns of the path control network are not visible at the protocol boundary with the NAUs; in particular, the path control network conceals the node interconnections and the NAUs need only consider their logical connections (i.e., sessions) with other NAUs.

OTHER DEFINITIONS AND NOTATIONAL CONVENTIONS

This section describes some notational conventions widely used in both the figures and the text. (Additional conventions are defined within figure legends throughout the book.)

A naming convention, using qualifiers separated by periods to denote more specific components of a composite protocol machine, is used throughout the book. Component submachines are shown as blocks within a larger block that represents the composite machine.

In many cases, it is desirable to identify a qualifier by a phrase of multiple terms, in order to better convey the meaning of the qualifier. The multiple terms in the phrase are connected by underscores to indicate that they are part of a phrase rather than separate qualifiers representing further decompositions. The underscore convention is also used in names of states and data structures.

Each protocol machine in the book has a unique name consisting of a sequence of qualifiers. For example, (MACHINE.PRI.X_SEND, MACHINE.SEC.X_RCV) and (MACHINE.SEC.X_SEND, MACHINE.PRI.X_RCV) are examples of two basic protocol machine pairs. This naming convention produces protocol machine names that carry precise information on the role of the protocol machine and its relative position in the network structure.

Two other symbols, "|" and "&," are used in names and expressions. The "|" symbol indicates one of several (or "either...or"). For example, MACHINE.(PRI|SEC) means "either MACHINE.PRI or MACHINE.SEC." The "&" symbol is used to indicate composition. For example, MACHINE.(RCV&SEND) is the composite protocol machine consisting of MACHINE.RCV and MACHINE.SEND.

Some of the protocol machines defined in the book interact directly with undefined components. These undefined components, called undefined protocol machines (UPMs), represent implementation and/or installation options that are not architecturally prescribed (being product or user oriented).

Within block diagrams, the following conventions indicate the type of interaction between components:

- Solid arrows indicate data flow; between processes, this implies send/receive (asynchronous) logic.
- Dotted arrows indicate calling relationships.
- Dotted lines indicate data structure access.

Message units exchanged between SNA components are also denoted by special notation, particularly in sequence flow diagrams. A message unit is either a request or a response, depending on the RH coding (see SNA Formats); these are denoted respectively by a request-unit name (here designated generically by the term "RQ") and by RSP.

RQ(QUAL) denotes a request having the property described by QUAL; for example, RQ(Begin Chain), or simply RQ(BC), denotes a request whose RH is coded "Begin Chain." A similar convention applies to responses. For example, RSP(BIND) denotes a response to the BIND request—a response that echoes the request code "BIND."

The asterisk (*) character is used in sequence flows, as well as elsewhere, to mean "any value" (or "don't care"). For example, "*BC" means "BC or ¬BC"—where "¬" is the standard symbol for "NOT."

"Chapter 2. Overview of the LU" describes additional conventions used in sequence flow diagrams.

The procedural logic in the formal description uses simple English, some control-structure elements (e.g., if/then/else) common to most high-level languages, and a few straightforward conventions that are generally clear in context. For example, a call is frequently shown in the form: "Call PROCEDURE(X, Y, Z)"; this results in calling PROCEDURE and passing it the arguments X, Y, and Z. Perhaps, the only control-structure needing additional explanation is the select/when group: at most, one when-clause is executed in a given pass.

Abbreviations commonly used in the text are listed at the back of the book (Appendix T) for easy reference.

CHAPTER 2. OVERVIEW OF THE LU

INTRODUCTION

This chapter is an overview of logical unit type 6.2 (hereafter referred to simply as LU). The LU provides application programs with support functions for distributed transaction processing.

CONCEPTS AND TERMS

DISTRIBUTED TRANSACTION PROCESSING

Distributed transaction processing involves two or more programs, usually at different systems, cooperating to carry out some processing function. This involves program intercommunication to share each other's local resources such as processor cycles, data bases, work queues, or human interfaces such as keyboards and displays.

The LU supports distributed transaction processing by serving as the port between the programs and the path control network. It allows a transaction program (TP) to invoke remote programs and to exchange data with them.

All communication provided by the LU is program-to-program. Any end user that is not a program is represented to the LU by a program. For example, fixed-function terminals and their devices (e.g., keyboards and displays) present themselves as fixed programs (e.g., microcode) that use the same LU functions as user-written application programs. Human users at workstations do not interact directly with the LU but rather with local workstation programming support, which in turn interacts with the LU.

This program-to-program communication accommodates a variety of distributed processing connections, including peripheral node to subarea node, subarea node to subarea node, peripheral node to peripheral node through the subarea network, and direct T2.1 node to T2.1 node. For example, an application program at an outlying site (a terminal or a distributed processor) might communicate with a data-base management system at a central processor to maintain consistency between regional and central records. For another example, systems programs in workstations might exchange files and documents with each other.

Figure 2-1 on page 2-2 illustrates the role of the LU in relation to an SNA network. The LU connects transaction programs to the path control network. The LUs activate sessions between themselves. The component of a session in each LU is called a half-session. Two or more sessions between the same pair of LUs are called parallel sessions. Multiple sessions can concurrently use the same physical resources connecting the LUs.

The logical connection between a pair of transaction programs is called a conversation. A transaction program initiates a conversation with its partner with the assistance of the LUs. While a conversation is active, it has exclusive use of a session, but successive conversations may use the same session.

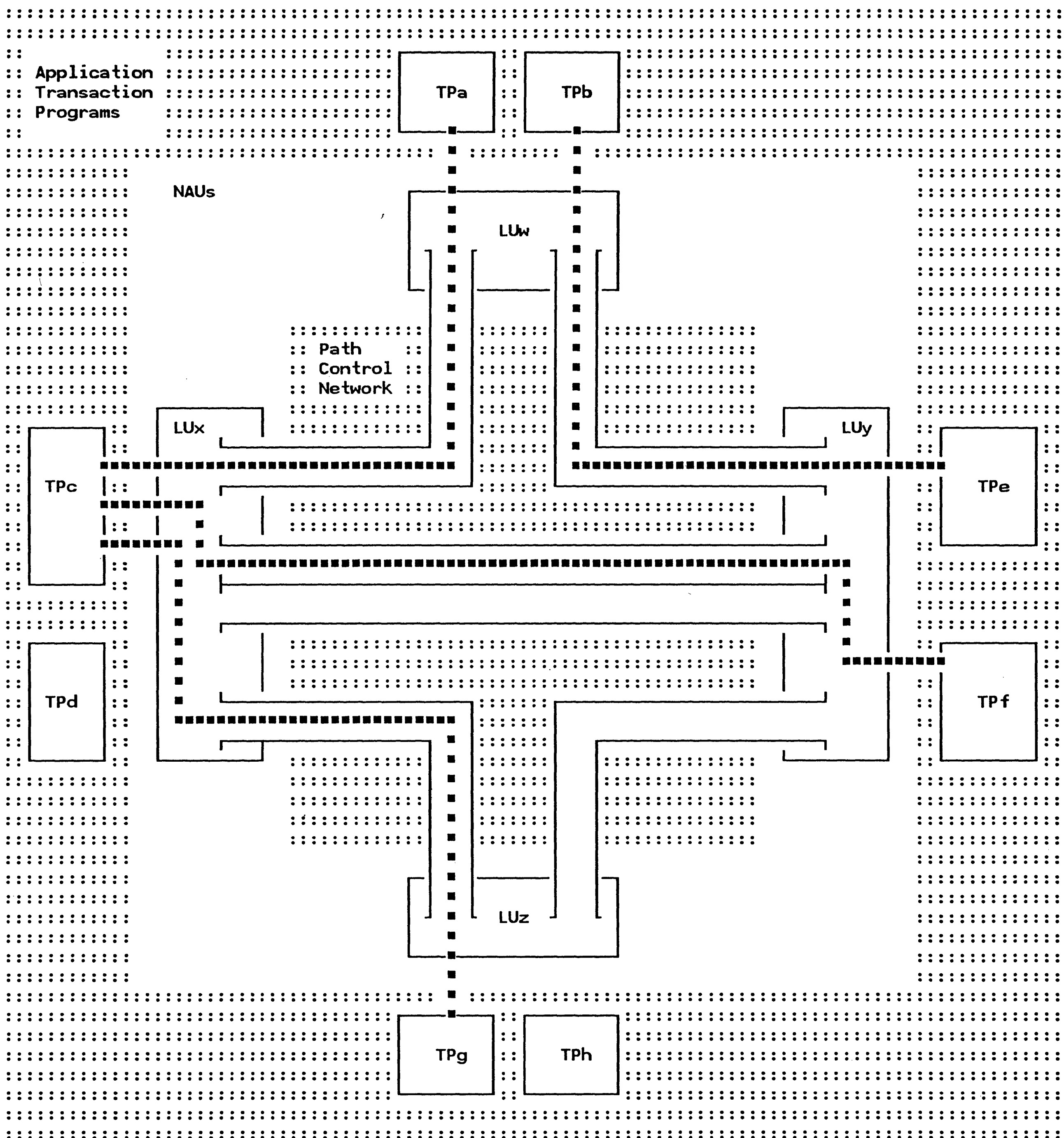
An LU may run many transaction programs successively, concurrently, or both. Each transaction program may be connected to one or more other transaction programs by conversations. Multiple conversations between different pairs of transaction programs can be active concurrently, with each conversation using a distinct session.

Conversations connect TPs in pairs, but any TPs directly or indirectly connected to each other by conversations are participating in the same distributed transaction. For example, if TP A and TP B are connected by a conversation, and, concurrently, TP B and TP C are connected by a conversation, then TPs A, B, and C all are participating in the same distributed transaction.

TRANSACTION PROGRAMS

The direct user of the LU is an application transaction program (application TP). Application TPs are provided by the end user to carry out functions of distributed applications.

A transaction program is distinguished from programs in general by two characteristics:



LEGEND:

Single Session (connecting two LUs)

Parallel Sessions (connecting two LUs)

Conversation(connecting two TPs)

Figure 2-1. Placement of LUs within the SNA Network (Example)

the way it is invoked, and the communication functions it initiates.

A transaction program is invoked by another transaction program by a mechanism called Attach. The invoking transaction program initiates a conversation with another named program. The invoked program is started running and is connected to the conversation with its invoker. (In the case of the initial program of a distributed transaction, the LU receives a START_TP record sent by a process external to the LU, e.g., the node operator facility [NOF], which prompts the LU to invoke a transaction program. For more information about NOF, refer to "Functions of Components of the Node External to the LU" on page 2-35.)

A transaction program uses the LU to communicate with other transaction programs by issuing transaction program verbs (which are described in SNA Transaction Programmer's Reference Manual for LU Type 6.2). (In some cases, internal LU components also issue transaction program verbs on behalf of transaction programs.)

Besides application transaction programs, distributed transactions can include transaction programs provided by the LU itself, called service transaction programs (service TPs). These are SNA-defined transaction programs within the LU that provide utility services to application transaction programs or that manage the LUs. They are attached by other transaction programs and they issue transaction program verbs to communicate with other transaction programs. For example, the LU includes service transaction programs for distributed operator control of the LU, by which control operators can determine the number of parallel sessions they will share, and for sync point resynchronization, which assists distributed transaction recovery following transaction failure in certain circumstances. Other service TPs provide document interchange services (using Document Interchange Architecture [DIA]), which allow processors and workstations to synchronously exchange files and documents. Furthermore, SNA Distribution Services (SNA/DS) service TPs provide asynchronous distribution of files and documents.

Different execution instances of the same transaction program could perform parts of the same distributed transaction at different LUs or parts of several different transactions at the same LU.

CONTROL OPERATOR

The LU control operator describes and controls the availability of certain resources (see "Resources"); for example, it describes network resources accessed by the local LU and it controls the number of sessions between the LU and its partners.

The LU control operator is represented to the LU by a control-operator transaction program that interacts with the LU on behalf of, or in lieu of, a human operator. The relationship between the control-operator transaction program and the LU control operator is implementation-defined.

The control-operator transaction program invokes operator functions by issuing control-operator verbs. These verbs are issued by the control-operator transaction program to convey operator requests to the internal components of the LU. Control-operator verbs are described in SNA Transaction Programmer's Reference Manual for LU Type 6.2.

RESOURCES

The LU provides several kinds of resources to support distributed transactions.

Conversations connect transaction programs and are used by the transaction programs to transfer messages. A conversation is activated when one transaction program attaches another.

Associated with each end of a conversation are protocol states that each LU maintains in order to coordinate interaction between the two TPs. These indicate (for example) which TP is sender and which is receiver at a given time.

The LU provides two types of conversations.

Mapped conversations allow the TPs to exchange arbitrary data records in any format set by the programmers.

Basic conversations allow TPs to exchange records containing a two-byte Length prefix.

Application transaction programs typically use mapped conversations, and service transaction programs typically use only basic conversations; however, either conversation type might be used by either program type.

Sessions provide relatively long-lived connections between LUs; a session can be used by a succession of conversations. Sessions are activated by LU pairs as a result of operator commands and transaction-program requests for conversations. Session awareness by the transaction program is unnecessary for successful communication. Most transaction programs need be concerned only with conversations, leaving the LU to manage sessions.

A mode is a set of characteristics that may be associated with a session. These characteristics typically correspond to different requirements for cost, performance, and so forth. Modes are defined by the control operator as a selection of path-control-network facilities and LU session-processing parameters.

One characteristic of mode is class of service. The path control network can offer different classes of service that correspond to particular physical links and routes and particular transport characteristics such as path security, transmission priority, and bandwidth

Other characteristics of mode include operator-selected processing parameters such as message-unit sizes and the number of message units sent between acknowledgments (pacing window sizes).

Each mode characterizes a group of sessions with a particular partner LU; multiple modes may exist for the same partner LU. Modes associated with different partner LUs are considered distinct, even if they represent similar sets of characteristic

A combination of partner LU and mode is called an (LU,mode) pair.

LU-accessed network resources constitute the relatively static environment that the LU or its containing node establishes as a result of installation definition. The principal components of this environment are the LU itself, the control point that serves the LU, the transaction programs that the LU can run, the potential partner LUs (remote LUs) with which the LU can communicate, and the modes of service available between the LUs.

Local resources are resources whose principal functions and operations are not defined by SNA, but which LU components use or interact with for some functions. These include local files, data bases, recovery and accounting logs, queues, and terminal components. For example, LU components interact with local data-base managers to coordinate distributed error recovery of data-base updates. Also, SNA distribution services uses queues to exchange messages between application transaction programs that provide document routing and distribution.

Protected resources are local resources, such as data bases, whose state changes are logged so that all resources changed by a transaction can be restored to a consistent state in the event of a transaction failure. The LU interacts with protected resources to provide the sync point function (see "Sync Point Function" on page 2-37) for distributed error recovery.

PROTOCOL BOUNDARIES

In order to accommodate LU implementations on different processors and transaction programs written in different programming languages, SNA defines the LU's interface to application transaction programs in generic terms only. This specification is called the transaction program protocol boundary. It consists of the set of LU functions that a TP may request, and the possible parameter values that may be supplied or returned for these functions.

SNA does not define a particular syntax or format for representing these functions and parameter values. Nevertheless, for purposes of discussion in SNA publications, the functions and parameters are represented generically by transaction program verbs; these are described in SNA Transaction Programmer's Reference Manual for LU Type 6.2.

Each LU implementation has one or more programming environments that provide these functions. Each such environment is called an applications programming interface (API).

The LU actually presents a partitioned protocol boundary to the transaction program; for example, there are separate subsets of the verbs for mapped conversations, for basic conversations, and for SNA/DS. When a hierarchical relationship exists between these subsets, e.g., when verbs from one set cause internal issuances of verbs from another set, this partition introduces sublayers within the LU.

A protocol boundary can be interpreted from two points of view.

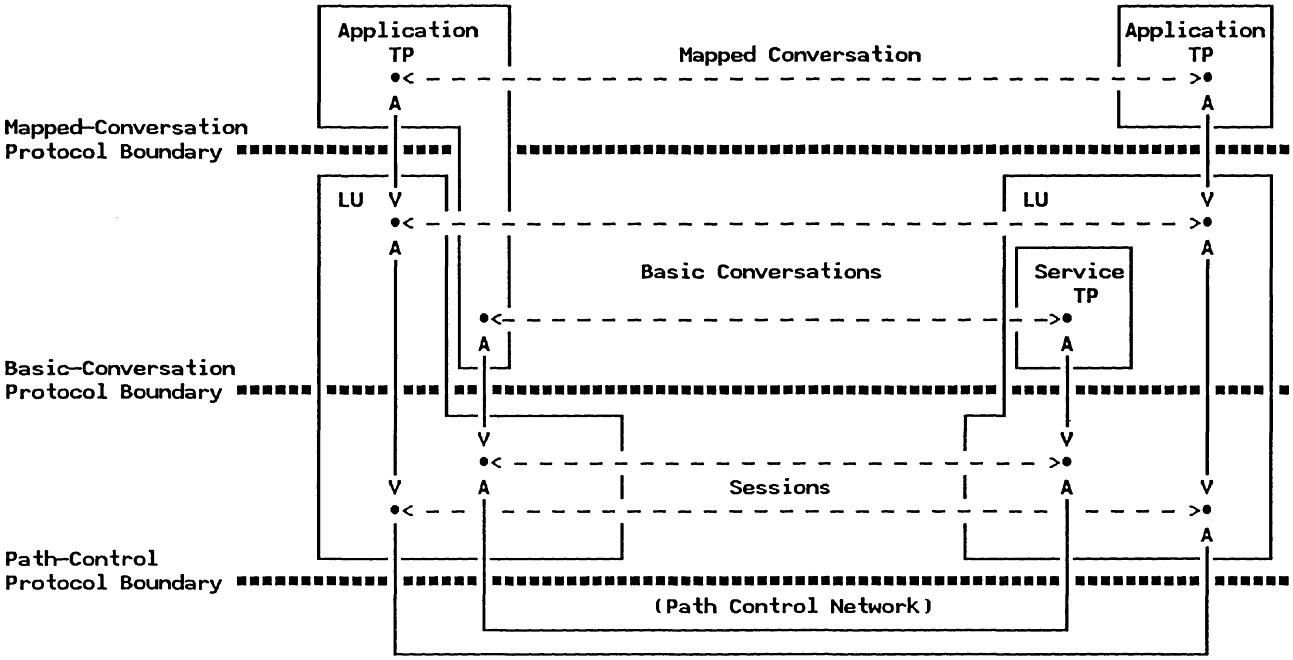
From one point of view, a protocol boundary is a boundary between two layers or sublayers of the node. For example, TPs exchange data with LUs across the TP-LU protocol boundary, and LUs exchange data with the path control network across the LU-path-control protocol boundary. From this viewpoint, the rules of exchange define protocols between adjacent layers.

But from another point of view, a protocol boundary is a boundary between two paired (or peer), but distributed, components of the same layer. In other words, the transaction program protocol boundary may be thought of as a direct boundary between one TP and another, and similarly, the path control protocol boundary may be regarded as a direct boundary between LUs.

Figure 2-2 on page 2-5 shows the principal protocol boundaries between the LU and external components. The figure illustrates how the protocol boundaries divide the LU into layers and sublayers, and how the conceptual flows between peer components are accomplished by successive adjacent-layer exchanges. In this example, the application TP has a mapped conversation with another application TP and a basic conversation with a service TP. The figure illustrates that the conceptual information flow between peer components at each layer is reduced to conceptual information flow at the next lower layer by actual information flow between layers and information transformation within layers. For example, the conceptual mapped conversation connection is reduced to a basic conversation; each basic conversation is reduced to a session; and finally, the sessions are reduced to connections in the path control network (which itself performs further layer transformations that are not shown).

NAMES

The LU allows transaction programs to refer to its resources, such as other TPs and LUs



LEGEND:
<- - -> conceptual flows between paired components
<-----> actual flows between adjacent layers
■■■■■■■ protocol boundary between layers or sublayers

Figure 2-2. Exchanges between Paired Distributed Components and between Adjacent Layers

and shared communication facilities, by installation-selected names. Thus, the programs need not be concerned with implementation and configuration details such as the actual network locations or transport characteristics. For example, when one transaction program invokes another, the invoking TP identifies the partner TP by a transaction program name, it identifies the partner LU by an LU name, and it identifies the desired set of session characteristics by a mode name.

Names are character strings that the installation associates with particular resources. They are specified by the control operator (on behalf of the installation management) subject to the SNA-imposed constraints, e.g., character set and length restrictions, described in SNA Transaction Programmer's Reference Manual for LU Type 6.2 (Within an LU implementation, the local resource names may differ from those that conform to SNA; for example, a program directory might use names of a different length or character set. In this case, the implementation always translates between its internal names and the SNA-conforming names that are used by transaction programs or that are transmitted outside the LU.)

The name of a particular resource is known within a particular environment. Within this environment, the name of each entity of a particular class is unique, but the same entity might have different names in different environments. For example, each LU allows local aliases for remote resource names, so that local transaction programs can be made insensitive to name changes elsewhere in the network. Of course, the control operator must change the LU's relevant name-translation tables whenever the remote names are changed.

Roles

Hereafter, the following terms are used to distinguish the roles of individual TPs and LUs of a pair. With respect to location, the term local means residing at the LU from whose perspective an activity is described; the term remote means residing at that LU's actual or potential session partner. With respect to a conversation, the source TP (or its LU) is the initiator of a conversation with the target TP (or its LU).

Transaction Program References

A source TP selects a target transaction program by its transaction program name (TPN) as defined at the source LU. In the simplest case, this is also the name of the TP as defined at the target LU. Optionally, however, the source LU can allow the two names to be different, in which case it converts the TP-supplied name into the TPN recognized at the target LU.

A TPN alone does not uniquely identify a transaction program instance. If the number of target transaction program instances does not exceed its instance limit, the target LU creates a new transaction program instance for each Attach it receives; otherwise, the target LU queues the Attach to await the freeing of a target transaction program instance.

LU References

Each LU provides a set of LU names by which its TPs may refer to remote LUs: these names are called local LU names (a local LU name is a local alias of a remote LU's name, not the local LU's own name). Local LU names are unique within each LU, but not necessarily outside an LU.

The control points involved in session initiation identify each LU by its network-qualified LU name This name consists of a network ID followed by a network LU name. The network ID is unique throughout a set of interconnected SNA networks; the network LU name is unique within a particular SNA network.

The path control network routes information to an LU by a routing identifier rather than by a name.

During session initiation, the LU supplies the network-qualified LU name to the control point. The control point provides a routing identifier for that network-qualified LU name. The correspondence between names and routing identifiers is established by the control point during session initiation. For more information on the relationship of LU names to routing identifiers (local-form session identifiers, or LFSIDs), refer to SNA Type 2.1 Node Reference.

The LUs themselves use their network-qualified LU names for certain purposes; for example, LUs resolve some race conditions by exchanging and comparing their network-qualified LU names.

Mode Names

A source TP can specify that the session selected for a conversation have a particular set of characteristics, or mode. It does this by specifying a corresponding mode name.

Mode names are shared between a given pair of LUs and are unique at an LU relative to a particular partner LU. Mode names for different partner LUs are independent: the same mode name can correspond to different sets of session characteristics for different partner LUs.

Internal Identifiers

The LU assigns internal identifiers to conversations and sessions once they are activated. These are called resource IDs and half-session IDs, respectively. TPs or the control operator use these identifiers for subsequent references to these entities. These identifiers are generated by the LU and passed back to the transaction program or to the control operator in the form required for subsequent verbs; the transaction program or operator need not interpret these identifiers.

CONVERSATION CHARACTERISTICS

Send/Receive Protocol

The LU normally allows TPs to exchange data in only one direction at a time, i.e., one TP sends and the other receives until the sending TP surrenders the right to send. This is called half-duplex flip-flop protocol. The LUs coordinate and enforce the send/receive state at each end of the conversation. LUs do allow some exceptions to strict alternation of send and receive: the receiving TP, at any time, can send an error indication, putting itself in send state; it can send the partner an attention indication, e.g., to request the right to send; and it can abnormally terminate the conversation.

Sender/Receiver Concurrency

Different applications require different degrees of concurrency between sender and receiver. For example:

- On-line inquiry applications might require real-time interaction.
- Status-reporting applications might require immediate transmission but no response.
- Document distribution applications might allow sending and receiving at the sender's and receiver's convenience, respectively, which might be separated by arbitrary periods of time.

For the first two cases, the LUs use direct conversations between the TPs.

For the real-time interactive case, the LU keeps the TP-TP connection active until the transaction is completed; both the source and target TPs are concurrently active. This is called synchronous transfer.

The LU treats the immediate-transmission, no-response case as a special case of synchronous communication, using a one-way conversation. The source LU allocates (initiates) a conversation as in the first case, sends the data, and deallocates (releases) the conversation. When the message reaches the target LU, it initiates the target TP, which receives the data and likewise deallocates the conversation. But since the source TP is expecting no reply, it might have terminated while the data is still in transit through the path control network, before the target TP is initiated. Thus, the source and target TPs are not necessarily active at the same time.

For the third case, the LU provides SNA Distribution Services (SNA/DS). In this case, the sender, called the origin TP, and the ultimate receiver, called the destination TP, are typically not active at the same time. Therefore, the data is stored at one or more locations en route between periods of active transmission. This mode of communication is called asynchronous transfer.

In SNA/DS, the origin application TP sends a message unit, ultimately intended for the destination TP, to a local service TP. The service TP at the origin stores the data in local permanent storage. When the appropriate time for sending the data arrives, e.g., when lower-cost transmission facilities become available or after compensating for time-zone differences, a service TP at the origin allocates a conversation to a service TP at the destination and sends the data. The receiving service TP at the destination LU stores the data in local permanent storage for later retrieval. Finally, an application TP at the destination retrieves the stored message.

SNA/DS also allows multiple intermediate service TPs between origin and destination. The origin service TP can allocate a conversation to an intermediate service TP, which would receive the data, store it, and later forward it to another intermediate service TP or to the ultimate destination service TP.

Each SNA/DS service TP can also duplicate the data and send it to multiple destinations or application programs.

Mapping

Two communicating TPs might process the same information using different internal data formats (presentation spaces), e.g., differently organized data structures or different sets of individual structures and variables. To assist the TPs in interpreting data in formats suited to their internal processing algorithms while providing a mutually understood format for the data transmitted over the conversation, some LUs provide an optional function of mapped conversations, called mapping. (Mapping concepts are discussed in "Mapping Function" on page 2-36.)

SESSION ALLOCATION

A principal function of the LU is to provide sessions between LUs for use by conversations between TPs.

Session Multiplicity

Only one transaction-program pair at a time can use a particular session. In order to allow multiple concurrent transactions, e.g., for a multiprogrammed processor or a multiple-user workstation, some LUs, called parallel-session LUs, allow two or more sessions with a given partner LU. Any session between a pair of LUs that both provide parallel sessions is called a parallel session, even if only one such session is currently active.

Some LUs, called single-session LUs, allow only one active LU-LU session with a given partner LU. A single-session LU may have more than one session concurrently, but each concurrent session is with a different partner. Any session involving a single-session LU is called a single session, whether the other partner is a single-session LU or a parallel-session LU.

Thus, all sessions between a pair of LUs are of the same type: single or parallel. Some LU protocols used on single sessions are different from those used on parallel sessions, but these differences are indistinguishable to transaction programs.

Session Pool

To avoid repeating session-activation processing for each conversation between the same pair of LUs, the LU allows successive conversations to use the same session.

When the LU activates a session or when a session previously in use by a conversation becomes free, the LU places the session in a session pool. When a transaction program initiates a new conversation, the LU allocates a session from this pool, if one is available.

Session Selection

Transaction programs do not select particular sessions, but specify only that the conversation be allocated a session with a particular partner LU and with a particular mode name. The LU partitions the session pool by partner LU and mode name; the LU allocates a session from only those sessions for the requested (LU,mode) pair.

Session Contention Polarity

Another session-selection criterion concerns the relative priority of the LU for use of the session. The LUs at each end of a session could both try to start a conversation at the same time. To resolve this contention, the LU operator specifies, for each session, which LU's TP will be allowed to use the session in such a case; this is called the session contention polarity. From the viewpoint of the local LU, a session for which that LU is designated to win an allocation race is called a contention-winner session (also referred to as a conwinner or a first-speaker session). A session that the local LU will surrender to the partner is called a contention-loser session (also referred to as a conloser or a bidder session--so called because a contention-loser LU will bid, i.e., request permission of the contention-winner LU to use the session).

Session Limits

The number of sessions in the session pool is constrained by operator-specified criteria, including several limits on the number of active sessions.

The total LU-LU session limit is the maximum number of sessions that can be active at one time at the LU.

The (LU,mode) session limit is the maximum number of LU-LU sessions that can be active at one time at an LU for that particular (partner LU,mode) pair.

The automatic activation limit for a particular (LU,mode) pair specifies the maximum number of LU-LU sessions that the LU will activate independently of requests for conversations. Automatically activated sessions constitute the initial session pool (additional sessions, within the other limits, are added to the pool on demand from conversation requests).

The local-LU minimum contention-winner limit for a particular (LU,mode) pair determines the minimum share of the total number of sessions for that (LU,mode) for which the local LU can be contention winner. Similarly, the partner-LU minimum contention-winner limit determines the minimum share of those sessions for which the partner LU can be contention winner.

Session limits are discussed in "Chapter 5.4. Presentation Services--Control-Operator Verbs" in more detail.

STARTING AND ENDING SESSIONS

Phases

Starting and ending sessions involves four phases of activity, although some phases are omitted in some circumstances.

Session-limit initialization and reset consists of issuing control-operator verbs (e.g., INITIALIZE_SESSION_LIMIT, RESET_SESSION_LIMIT) to specify the number of sessions the LU can have with a given partner, and to specify conditions for their activation and deactivation.

Session initiation and termination consists of control-point activity, such as supplying the network addresses corresponding to LU names, that mediates requests for session activation and deactivation.

Session shutdown consists of the LU activity to terminate conversation activity on a session prior to deactivating the session.[1]

Session activation and deactivation consists of creating or destroying the end-to-end logical connection between the LUs.[2]

SESSION USAGE CHARACTERISTICS

Session Activation Polarity

An LU activates a session with its partner by sending a message unit called BIND. The LU that activates a session (sends BIND) is called the primary LU (PLU); the LU that receives BIND is called the secondary LU (SLU). These terms are relative to a particular session: the same LU can be primary LU for one session and secondary LU for another.

The primary LU always has first use of the session, i.e., it can initiate the first conversation on the session, regardless of the session contention polarity. (When the first conversation completes, the principal right to initiate conversations reverts to the contention-winner LU.)

Session-Level Pacing

To prevent an LU from sending data faster than the receiving LU can process it (e.g., empty its receive buffers), the two LUs observe a session-level pacing protocol. At the time a session is activated, the LUs exchange the number (the pacing window size) and size (the maximum RU size) of the message units they can accept at one time. The sending LU will send no more message units than the receiver will accept (a pacing window) until the receiver sends an acknowledgment (pacing response) indicating that it can receive another pacing window. The pacing window size may be fixed for the duration of the session or varied adaptively in accordance with load and path congestion conditions. (For more information on pacing refer to "Chapter 6.2. Transmission Control")

---

1 Session shutdown protocols use data flow control RUs, e.g., BIS.
2 Session activation and deactivation protocols use session control RUs, e.g., BIND, UNBIND.

Profiles

Session traffic is characterized by a particular set of SNA-defined formats and protocols, identified by a function management (FM) profile and a transmission services (TS) profile (see SNA Formats). The profile used depends on the kind of session and the kind of node: FM profile 19 and TS profile 7 are used by LU 6.2 for LU-LU sessions.

```
              Primary LU                                              Secondary LU

                                       BIND (RD1)
     1             o------------------------------------------------------>

                                RSP(BIND, PW[RD1], RD2)
     2             <------------------------------------------------------o

                ┌                        UNBIND
     3a         |  o------------------------------------------------------>
                |
       --or--<
                |                    FMH-12 (PW[RD2])
     3b         |  o------------------------------------------------------>
                └

                ┌                        UNBIND
     4a         |  <------------------------------------------------------o
                |
       --or--<
                |                          .
     4b         |                          .
                |                          .
                └

        LEGEND:
             RDi       random data (i=1|2)
             PW        LU-LU password
             PW[RDi]   RDi enciphered using PW as cryptography key
```

Figure 2-3. LU-LU Verification

SECURITY

The LU provides three functions to assist the installation in providing security: partner LU verification, partner end-user verification, and session cryptography. Partner-LU verification is a session-level security protocol; it involves protocols at the time the session is activated. Partner end-user verification is a conversation-level security protocol, taking place at the time a conversation is started. Session cryptography is another session-level protocol, the parameters for which are exchanged at session activation.

Partner-LU verification is done by a three-flow exchange between the two LUs, with each LU using an LU-LU password and the Data Encryption Standard (DES) algorithm. This exchange is called LU-LU verification. LU-LU passwords (see SNA Transaction Programmer's Reference Manual for LU Type 6.2) are established by implementation and installation-defined methods outside of SNA. LU-LU passwords are on a partner-LU basis: one LU-LU password is established between each LU pair. This password is used for all sessions between the LU pair. It is recommended that each LU pair have a unique password; however, it is not an architectural requirement.

Figure 2-3 shows the LU-LU verification protocol exchanges. In the following discussion, the numbers in parentheses correspond to the numbers in that figure.

During session activation, random data (RD1) is sent in BIND from the primary LU to the secondary LU (1). The secondary LU enciphers this random data using the LU-LU password and the random data as input to the DES algorithm. The secondary LU returns (2) the now enciphered random data (PW[RD1]) to the primary LU along with its own randomly generated data (RD2) in RSP(BIND). The primary LU compares the received enciphered random data with its own copy of the random data that it enciphered using its LU-LU password and the DES algorithm. If the two versions of the enciphered random data do not compare equally (3a), LU-LU verification fails, session activation fails, and a security violation is logged. If the two versions of the enciphered random data compare equally (3b), the primary LU has verified the identity of the secondary LU and LU-LU verification continues.

Using the LU-LU password and the DES algorithm, the primary LU enciphers the random data received from the secondary LU. The primary LU returns this enciphered random data (PW[RD2]) in a Security FM header (FMH-12) to the secondary LU (3b). The secondary LU compares this enciphered random data with its own version of the enciphered random data. If the two versions of the enciphered random data do not compare equally (4a), LU-LU verification fails, the session is terminated, and a security violation is logged. If the two versions of the enciphered random data compare equally (4b), the secondary LU has verified the identity of the primary LU, and LU-LU verification is complete.

When the transmission links and LUs that make up the network are physically secure (as determined by the installation management), LU-LU verification may be omitted. Under this circumstance, LU-LU verification would not take place, yet the session would still be considered secure; therefore, access to secure resources would still be permitted following conversation-level security protocols (see below). Permission to use conversation-level security to gain access to secure resources is installation defined and communicated to the partner LU during session activation in the BIND/RSP(BIND) exchange.

When the network is not considered secure, LU-LU verification should be omitted, and access to secure resources via conversation-level security should not be permitted. Denial of permission to use conversation-level security is installation defined; an indication of this denial is communicated to the sender of the request during session activation in the BIND/RSP(BIND) exchange.

End-user verification (conversation-level security) is used to confirm the identity of the partner end user (e.g., transaction program). When a TP requests access to another TP, it must supply adequate security information in the request to satisfy the security requirements of the requested TP, or the request will be rejected. This could include a user ID and password (see access security information subfields in _SNA Formats_) supplied by the end user that initiated the request. When a user ID and password are supplied on the request, they are verified by the LU that receives them. If the end user has not supplied the correct user ID and password combination, the request is rejected.

An optional additional criterion for access to a specific TP is permitted. This criterion would be a check of an authorization list associated with the target transaction program. The keys to search the authorization list would be combinations of the user ID and an optional profile supplied on the request, along with the name of the partner LU from which the request originated. The authorization list could be made up of combinations of user ID, profile, and partner LU name. After the user ID is verified by the LU, the authorization list may be searched for access rights to the specific transaction program named in the request. If the additional criterion is not met, the request is rejected.

An intermediate transaction program (one started by another TP) that requires conversation-level security may need to access an additional TP that requires conversation-level security. In this case, an Already Verified indicator is set in the additional request; the user ID and optional profile in the first request, which initiated the intermediate transaction program, are supplied in the second request. For security reasons, the password that initiates the intermediate TP is never saved, but the user ID and optional profile that initiated the intermediate TP are saved. The Already Verified indicator can be used only if the sender of the indicator is trusted by the receiver of the indicator to have performed the proper verification of the user ID and password that initiated the sender. This level of trust is installation defined at the receiver of the indicator and communicated to the sender of the indicator during session activation in the BIND/RSP(BIND) exchange.

To help prevent data from being interpreted or modified during transit, the LU provides _session cryptography_, whereby all user data is enciphered at the source LU and deciphered at the target LU. The encryption algorithm uses a _cryptographic key_, supplied by the control point, and a _session seed_, generated by one of the LUs when the session is activated. (See "Chapter 6.2. Transmission Control" for a full discussion of session cryptography.)

ERROR HANDLING

Kinds of Errors

Errors affecting transaction processing are classified as follows:

Application Errors: These are errors related to the application data and processing, e.g., user input error or data-base record missing. Detection and recovery are the responsibility of the transaction programs.

Local Resource Failure: These are failures in non-SNA resources, e.g., a disk read error. If the resources are not protected resources, recovery is the responsibility of the transaction program or of the non-SNA support for the failing resource, e.g., a disk subsystem. If the resource is a protected resource, the TPs can use the LU sync point function (see "Sync Point Function" on page 2-37) to assist in recovery in conjunction with non-SNA support.

Recoverable System Errors: These are errors or exceptional conditions, e.g., races resulting from contention for use of a session, for which an SNA-defined recovery algorithm exists. The LU performs the recovery algorithm; the transaction programs are normally not aware of these errors, except as they affect timing.

Program Failures: These are errors that cause abnormal termination of a transaction program. The LU recovers from such errors by deallocating any active conversations for the TP that were not deallocated by the failed transaction program, thus freeing the sessions for use by other transaction programs. Any further recovery depends on transaction program logic and implementation-defined capabilities such as error exits.

Session Failure: These are failures caused by unrecoverable failure of the half-sessions, e.g., invalid session protocols received, or by failure of the underlying network components, e.g., the links. This case is reported to the LUs through session outage notification (SON).

If a conversation is active on the session at the time of failure, the failure is manifested to the transaction program as a conversation failure (see below); otherwise, these errors do not affect transaction programs. LUs report the conversation failure to the affected transaction programs.

Conversation Failures: These are failures caused by unrecoverable failure of the underlying session. The resulting conversation failure is reported to each transaction program by a return code on a subsequent verb. The same session and conversation cannot be recovered, but the LU can activate another session.

The operator or the transaction programs have the responsibility to recover the transaction. To recover from an interruption in transaction processing, for example, the source transaction program can allocate a new conversation, using another session, to a new instance of the target transaction program or to another transaction program.

LU Failure: This is a failure of an LU from such causes as malfunction of the implementing hardware or software. In many cases, such a failure appears to remote (non-failing) LUs as a session failure, and they recover as they would from any other session failure. In some cases, recovery is performed by the sync point function.

Program Error Recovery Support Functions

The LU assists TP recovery from application errors and local resource failures by supporting the protocols discussed below to exchange error information and to immediately end messages or conversations.

Confirmation: This function (e.g., CONFIRM verb) allows a TP to solicit positive or negative acknowledgment of a message unit from the partner TP. The interpretation of this positive or negative acknowledgment (CONFIRMED or SEND_ERROR verbs, respectively) is program dependent: for one application, confirmation might mean only that the data was received; for another, it might mean data was safely stored on disk; for a third, it might mean that the data represents a valid account record update; and so forth.

Program Error Indication: This function (SEND_ERROR verb) allows a TP to inform the partner TP of a program-detected error; this includes sending negative acknowledgment to a confirmation request.

This function also causes program-to-program transfer of the current message unit to cease. If a TP detects an error while receiving, issuing the SEND_ERROR verb directs the receiving LU to ignore any additional data in transit (i.e., to the end of the conversation message--see "Conversation Message" on page 2-14); this is called purging. Similarly, if a sending TP detects an error, issuing the SEND_ERROR verb informs the partner that the source TP has stopped sending. If the TP stops sending before reaching a predetermined application-program data boundary (i.e., the end of a logical record--see "Logical Record" on page 2-13), this is called truncation.

Sync Point: Many transactions require consistent, regular updates of distributed resources such as distributed data bases. While a transaction is in progress, however, the resources at different LUs can enter mutually inconsistent interim states. If one of the transaction programs encounters an error, some recovery action may be necessary to restore the resources to mutually consistent states. In order to verify or restore consistency among distributed resources, some LUs provide a distributed error-recovery function, called sync point. (Sync point concepts are discussed in "Sync Point Function" on page 2-37.)

Abnormal Conversation Deallocation: This function allows a TP to abnormally terminate a conversation. A TP might do this, for example, when an error is detected for which it has no recovery procedure and continuing the transaction would be meaningless. When this is received, the LU informs the TP that the conversation has been abnormally terminated.

When a component of the LU (e.g., an LU service TP) abnormally terminates a conversation that is being used by a TP, the LU uses DEALLOCATE TYPE(ABEND_SVC) to terminate the conversation. This allows the TP and its partner TP to distinguish between application-generated and LU-generated abnormal terminations.

LU Error Recovery Functions--Abnormal Session Deactivation

For some errors, the LU or operator initiates recovery.

If an unrecoverable session-protocol error occurs, the LU abnormally deactivates the session.

If the control operator detects an error, e.g., an apparent deadlock or loop, it can force immediate abnormal deactivation of a session.

Either of these cases are normally manifested to affected transaction programs as conversation failure.

BASE AND OPTIONAL FUNCTION SETS

The LU functions and protocols are organized into subsets. The function sets consist of a base function set, which provides basic communication services common to all LU implementations, and a small number of optional function sets, which may be used by implementations with more sophisticated additional requirements. These SNA-defined function sets are described in *SNA Transaction Programmer's Reference Manual for LU Type 6.2*.

All LU 6.2 implementations of a given function set provide that function in a way that conforms to the protocol boundary. Furthermore, an LU 6.2 implementation that provides one function in an option set provides all other functions in that option set as well. Thus, all LU 6.2 implementations can communicate using the base set, and any two implementations supporting functions in the same option set can communicate using that full option set.

Two kinds of optional functions exist. Send options determine what formats and protocols will be sent but do not affect what can be received; all formats and protocols sent using these options can be received by all LUs. Receive options determine what can be received as well as what can be sent. For receive options, the source LU and TP requirements are described in the BIND and the Attach; the receiving LU rejects the session or conversation if it, or the specified TP, does not support the required options.

All implementations of LU 6.2 include the functions described in this book in their entirety except were options are specifically defined. For additional definition of options see *SNA Transaction Programmer's Reference Manual for LU Type 6.2*. The principal base and optional functions are listed below.

Application Program Interface Implementations

Open-API implementations support arbitrary user-written transaction programs, e.g., a data-base management system running on a host processor. For these implementations, the API provides verbs and parameters for all of the base function set, and perhaps some optional function sets.

Closed-API implementations do not support user-written programs but provide only a fixed, implementation-determined set of service transaction programs, e.g., a DIA service transaction program for an office workstation. For these implementations, the API provides only the particular verbs and parameters that the transaction program set requires.

Principal Base Functions

Basic Conversations: All implementations provide receive support for all basic-conversation formats and protocols.

Open-API implementations provide basic conversation verbs, but not necessarily in all supported programming languages. For example, an implementation might support both basic- and mapped-conversation verbs in a systems-programming language such as Assembler, but provide only mapped-conversation verbs in high-level languages.

Mapped Conversations: All open-API implementations provide mapped conversations (primarily in high-level languages).

Principal Optional Functions

Mapping: This is an optional function for mapped conversations (see "Mapping Function" on page 2-36).

Sync Point: This is an optional function for basic and mapped conversations (see "Sync Point Function" on page 2-37).

Program Initialization Parameters (PIP): This is the means of passing initial parameters or environment setup information to a target TP.

Security: This is an optional function for verifying the identity of partner LUs and end users (see "Security" on page 2-9), and for protection of data in transit.

Performance Options: Several optional functions exist to maximize performance for specific transaction requirements. For example, an LU can optionally allow transaction programs to eliminate or accelerate certain acknowledgments, or to perform processing concurrently with certain conversation functions. These are send options, so TPs written for implementations that support these options will operate correctly with partner TPs and LUs that do not support them.

A message unit (MU) is any bit-string that has an SNA-defined format and is transferred between SNA components or sublayers.

Distributed transaction programs exchange MUs with each other by means of LUs. Transaction programs exchange application-oriented units of data, e.g., a customer record or a document, over a conversation. The LUs, in turn, exchange session-oriented MUs via the path-control network. But the content and format of an MU most appropriate for exchange between transaction programs is in general different from that most appropriate for transmission on a session. Whereas an application program typically uses a record size corresponding to logical groupings of the data, the LU typically uses MU sizes related to internal buffer sizes and efficient flow control. Furthermore, the LU may need to add encoded protocol information, such as confirmation requests or MU sequence numbers, to the program-supplied data.

The LU transforms program-oriented MUs used by the TP into network-oriented MUs used by the path control network, and vice versa. (Throughout this section, message-unit transformations are described from the sender's side, i.e., transaction program to LU to network; the process is inverted at the receiver.)

The message-unit transformation takes place in stages. Each stage transforms some of the information from the higher stage into a SNA-defined bit string. Typically, a stage reblocks (regroups) the MUs from the previous stage into differently sized units and converts the protocol information into formatted headers (prefixes) to the reblocked data, thus creating new MUs.

MAPPED-CONVERSATION MESSAGE UNITS

A data record, at the mapped-conversation protocol boundary, is a collection of data values that correspond to the DATA parameter of a single mapped-conversation MC_SEND_DATA verb issuance. The format of a data record is completely arbitrary within the constraints of the implementation and the transaction program. For example, it need not even be a contiguous byte string, but might be a collection of variables and structures.

A mapped-conversation record (MCR) is the elementary unit of information transferred between two TPs on a mapped conversation. A MCR contains the values of a data record represented as a string of contiguous bytes. It may be of arbitrary length. It contains no information for use by the LU; its internal format is significant only to the TP. The TP supplies needed protocol information, such as the mapped-conversation record length, in separate parameters of the verb, using representations appropriate to the programming language and processor being used.

(A MCR consists of data from a single verb issuance by the sender, but it may be received in one or more parts, each with a single verb issuance, depending on the receiving TP's receive buffer size).

BASIC-CONVERSATION MESSAGE UNITS

GDS Variables

Full connectivity among programs requires that all transaction programs interpret the records they transfer in the same way. To facilitate uniform interpretation of records among programs written for different processors, service transaction programs and some internal LU components, including mapped-conversation support, use the formats defined by general data stream architecture to represent records (see SNA Formats)

A general data stream (GDS) variable consists of a GDS header (LLID) followed by the data. The GDS header is a descriptive prefix containing a 2-byte length prefix (LL) that indicates the length of the variable, including prefix, and a format identifier called the GDS ID that indicates the GDS-defined format of the data. The LLs identify the boundaries of variable-length fields within a message unit of contiguous fields, and the GDS IDs identify the representation of the data. A GDS variable may be of arbitrary length. If the variable length exceeds the value that can be represented in the length prefix ($2^{15}-1$ = 32,767 bytes, including the prefix), the record consists of multiple segments, each with its own length prefix. Only the first segment contains an ID field. The length prefix also contains a continuation bit that indicates whether the corresponding segment is the last (or only) segment in the GDS variable.

All data transferred at the basic-conversation protocol boundary by service TPs and other internal LU components (but not necessarily data transferred by application transaction programs) is represented as GDS variables with SNA-defined formats (see SNA Formats).

Logical Record

A logical record is the elementary unit of information transferred between users of the basic-conversation protocol boundary. A logical record consists of a 2-byte length prefix (LL) followed by data. Its maximum length is 32,767 bytes, including the prefix.

The LL prefix of a logical record has the same format as the LL field in a GDS variable segment; thus, a GDS variable segment is also a logical record. The basic-conversation protocol boundary requires only the LL prefix, not a full GDS LLID. Thus, logical records generated by application TPs need not use ID fields; if they do, the application assigns and interprets the ID fields; the basic-conversation support of the LU treats everything following the LL prefix of the logical record as user data.

The logical record is the elementary unit for which the LU detects or reports truncation.

Buffer Record

It might be inconvenient for a transaction program to issue a single send or receive verb for each logical record. For example, the sender or the receiver might have limited buffer space or might not know ahead of time the maximum length of the records being sent. Or, the transaction program might prefer to send a group of small, related records with a single verb issuance. So, the unit of data that a program sends or receives with a single basic-conversation verb is of program-determined length. This unit is called a buffer record.

No SNA-defined limit exists on the length of a buffer record; for example, it could exceed 32,767 bytes. The buffer-record length can be different for each verb issuance.

No correspondence is necessary between the lengths or boundaries of logical records and those of buffer records, or between send buffer records and receive buffer records. Nevertheless, a receiving program may optionally specify that the LU begin a new receive buffer record for each new logical record received. The relationship between logical records and buffer records is illustrated in Figure 2-6 on page 2-18.

CONVERSATION MESSAGE-UNIT SEQUENCES

Certain sequences of message units are relevant to conversation protocols.

Conversation Message

A basic-conversation message consists of the sequence of logical records transferred in one direction from one TP to another without an intervening change of direction or confirmation. (The Attach FM header generated from the ALLOCATE verb is also considered part of the initial basic-conversation message.)

The end of a conversation message is determined, when sending, by a conversation state change caused by the verbs issued. For example, PREPARE_TO_RECEIVE, RECEIVE_AND_WAIT, CONFIRM, SYNCPT, and DEALLOCATE end a conversation message. When receiving, the end of a conversation message and conversation state change is determined from corresponding protocol information received from the sender. The information identifying the end of a conversation message and specifying the way it was ended is generically called the end-of-conversation-message indication.

A basic-conversation message is the elementary unit for which the LU supports confirmation or program-error reporting (e.g., SEND_ERROR) between sender and receiver, and for which it performs purging.

A mapped-conversation message is analogous to a basic-conversation message; that is, it consists of the sequence of mapped-conversation records (or data records) transferred in one direction from one TP to another without an intervening change of direction or confirmation, as understood at the mapped-conversation protocol boundary.

The unqualified term, conversation message, is used when the intended protocol boundary is clear from the context, or when both the mapped-conversation message and its corresponding basic-conversation message are designated.

Conversation Exchange

A conversation exchange consists of the complete set of mapped- or basic-conversation messages transferred between a pair of TPs using a particular conversation.

SESSION MESSAGE UNITS

Session message units are formatted for LU-LU protocols and for effective use of the path control network.

Function Management Headers

A function management (FM) header is a message unit generated by the LU to carry certain LU control information. The LU uses the following FM headers:

- An Attach FM header (FMH-5) specifies the name and required characteristics, e.g., option sets required, of the target TP.
- An Error-Description FM header (FMH-7) describes a transaction program error or Attach failure.
- A Security FM header (FMH-12) carries security information for LU-LU verification.

Basic Information Unit

A basic information unit (BIU) is the message unit transferred between two LUs. It consists of a request/response header (RH) and a request/response unit (RU).

The RH is a formatted prefix to the RU. It carries protocol information encoded from the TP verbs or generated internally by the LU. It may also appear without an RU in an IPR or IPM. SNA Formats gives further details.

Request RUs carry FM headers, TP-supplied data (formatted into logical records by the TP in basic conversations or by the LU in mapped conversations), and other protocol information. Response RUs carry limited information, such as the echoed request code or (in negative responses) sense data, but no TP-supplied data.

The LU uses the following RUs on an LU-LU session:

- Category FMD RUs, for transaction-program data
- Category DFC RUs, i.e., BIS, LUSTAT, RTR, SIG, and their responses
- Category SC RUs, i.e., BIND, UNBIND, CRV, and their responses

(For additional details, see SNA Formats.)

EXR also flows for some path-control-detected errors.

The LUs also transfer other information describing the BIU, such as the length and sequence number, which is formatted by path control in a transmission header (TH).

SESSION MESSAGE-UNIT SEQUENCES

The following sequences of BIUs are relevant to session protocols:

A (BIU) chain is a sequence of BIUs that constitute a single unidirectional transfer. The chain is the most elementary unit that can be independently confirmed or for which errors can be reported using SNA-defined LU protocols. It corresponds to a TP-TP conversation message.

A bracket consists of the set of all chains transferred on a particular conversation. It corresponds to a TP-TP conversation exchange. The first data RU in a bracket begins with an Attach FM header that identifies the target TP.

The total session traffic comprises a sequence of one or more brackets. Prior to bracket traffic, the session is activated (BIND protocols). Prior to normal session deactivation, bracket traffic is shut down (BIS protocols). All session traffic stops when the session is deactivated (UNBIND protocols), whether or not any brackets are in transit.

Figure 2-4 on page 2-16 illustrates the correspondence between the conversation message-unit sequences and session message-unit sequences. In the figure:

- The column labelled TP-TP shows the conversation message-unit sequences.

  (The corresponding conversation message-unit sequences for the partner TPs at LU Y are not shown; they are the reverse of those shown for TP A and TP B.)

- The column labelled LU-LU shows the session message-unit sequences.
- The column labelled LU X shows the relationship between the two sets of sequences.

MAPPED-CONVERSATION MESSAGE-UNIT TRANSFORMATION

The mapped-conversation support in the LU converts a data record into a GDS variable.

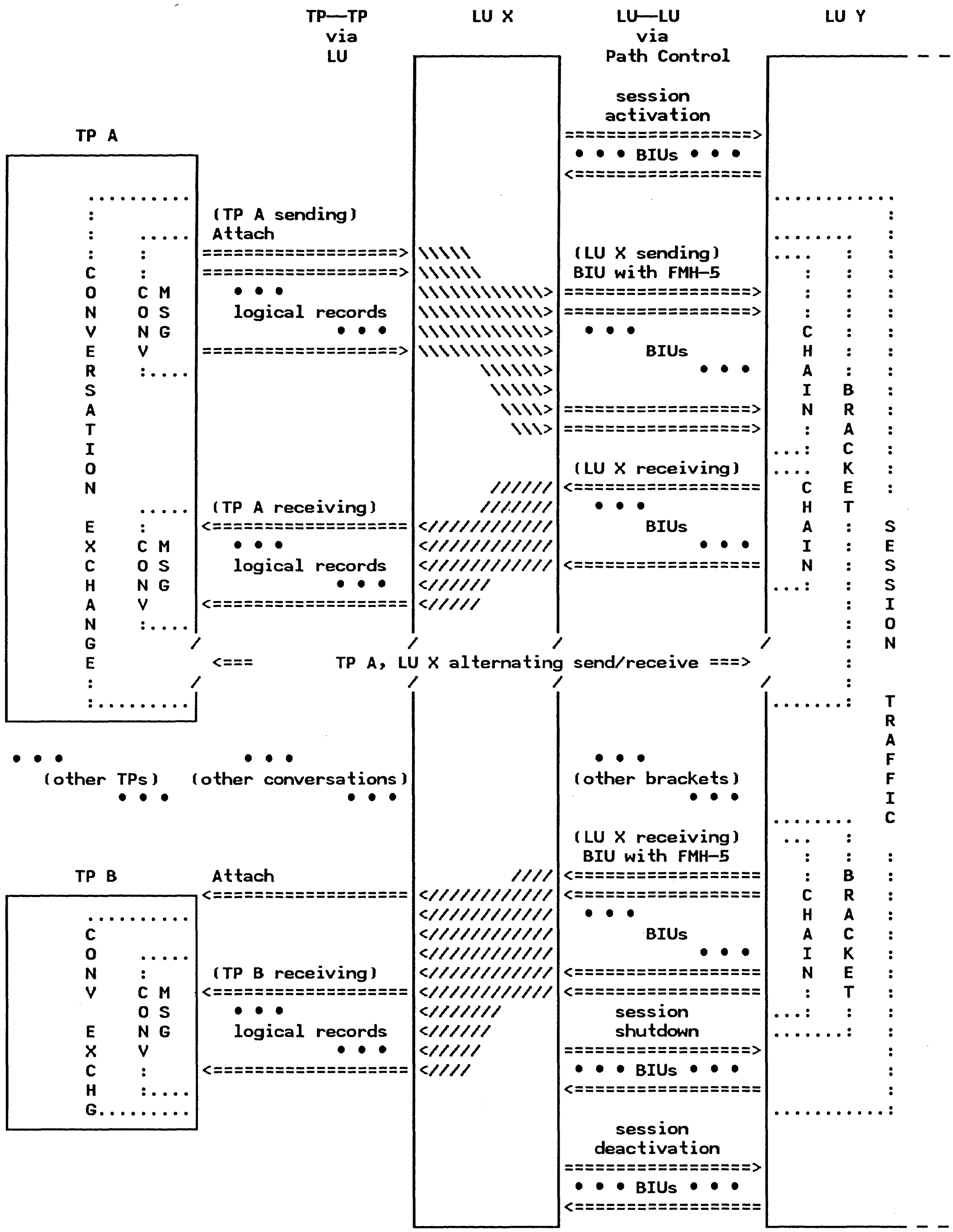
First, the LU optionally performs a TP-specified mapping transformation on the data record, producing a mapped-conversation record. If mapping transformations are not supported or if one is not specified, the TP supplies the data in MCR format (i.e., a contiguous byte string of TP-determined length).

The mapped-conversation support in the LU then segments the MCR into units of allowed logical-record length and adds LLID prefixes, thus producing a GDS variable consisting of a sequence of logical records. This is illustrated in Figure 2-5 on page 2-17.

BASIC-CONVERSATION MESSAGE-UNIT TRANSFORMATION

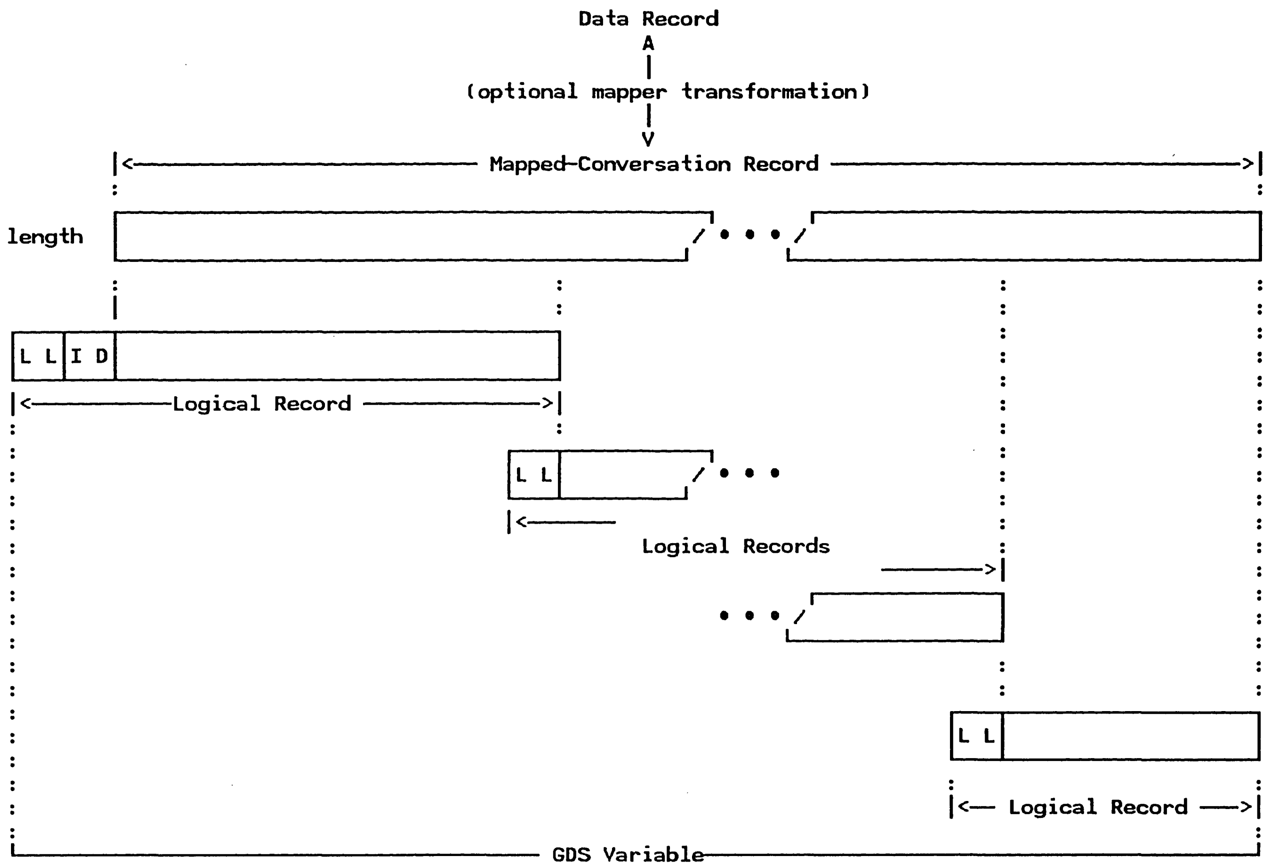
Above the basic-conversation protocol boundary, a TP, or an internal LU component such as the mapped-conversation support, generates a sequence of logical records constituting a conversation message. It passes this conversation message to the LU as a sequence of buffer records, by issuing basic-conversation verbs. Along with the buffer records, it passes unformatted protocol information such as the ALLOCATE verb parameters, from which the LU builds FM headers.

Conceptually, the LU assembles the sequence of FM headers and logical records into a complete conversation message. It then converts this conversation message into a chain of BIUs. Of course, the LU does not necessarily store a complete conversation message at one time; when it accumulates enough buffer records to build one or more BIUs, it builds



LEGEND:
<====> message-unit flows
\\\\\> conversion of logical records to BIUs
<//// conversion of BIUs to logical records
...... message unit sequence boundaries

Figure 2-4. Relationships of Sequences of Message Units (Example)



LEGEND:
data record: data supplied by the transaction program MC_SEND_DATA verb (arbitrary format)
length: length of the mapped-conversation record (after mapper transformation, if any)
LL: logical-record Length field; the first bit is the continuation field
ID: GDS ID field

Figure 2-5. Relationship of Data Records to Logical Records (Example)

those BIUs and sends them out, saving any residual data for the next BIU.

To build BIUs, the LU reblocks the FM headers and logical records into RU-sized units and generates the necessary RHs. The LU sets the RH indicators to correspond to functions or states specified by verb parameters; for example, it sets the chaining indicators (BCI, ECI) to indicate the first and last BIUs in the chain, and it sets the bracket indicators (BB, CEB) to indicate the first and last BIUs in a bracket. When necessary, the LU also generates Attach or Error-Description FM headers (FMH-5 and FMH-7) from verb parameters and includes these in the BIUs. The final result is a BIU chain. Along with the BIU, the LU generates parameter values for use by path control (to build the transmission header). The LU transfers the BIUs and the unformatted BIU parameters to path control for transmission to the partner LU. Figure 2-6 on page 2-18 illustrates the conversion process.

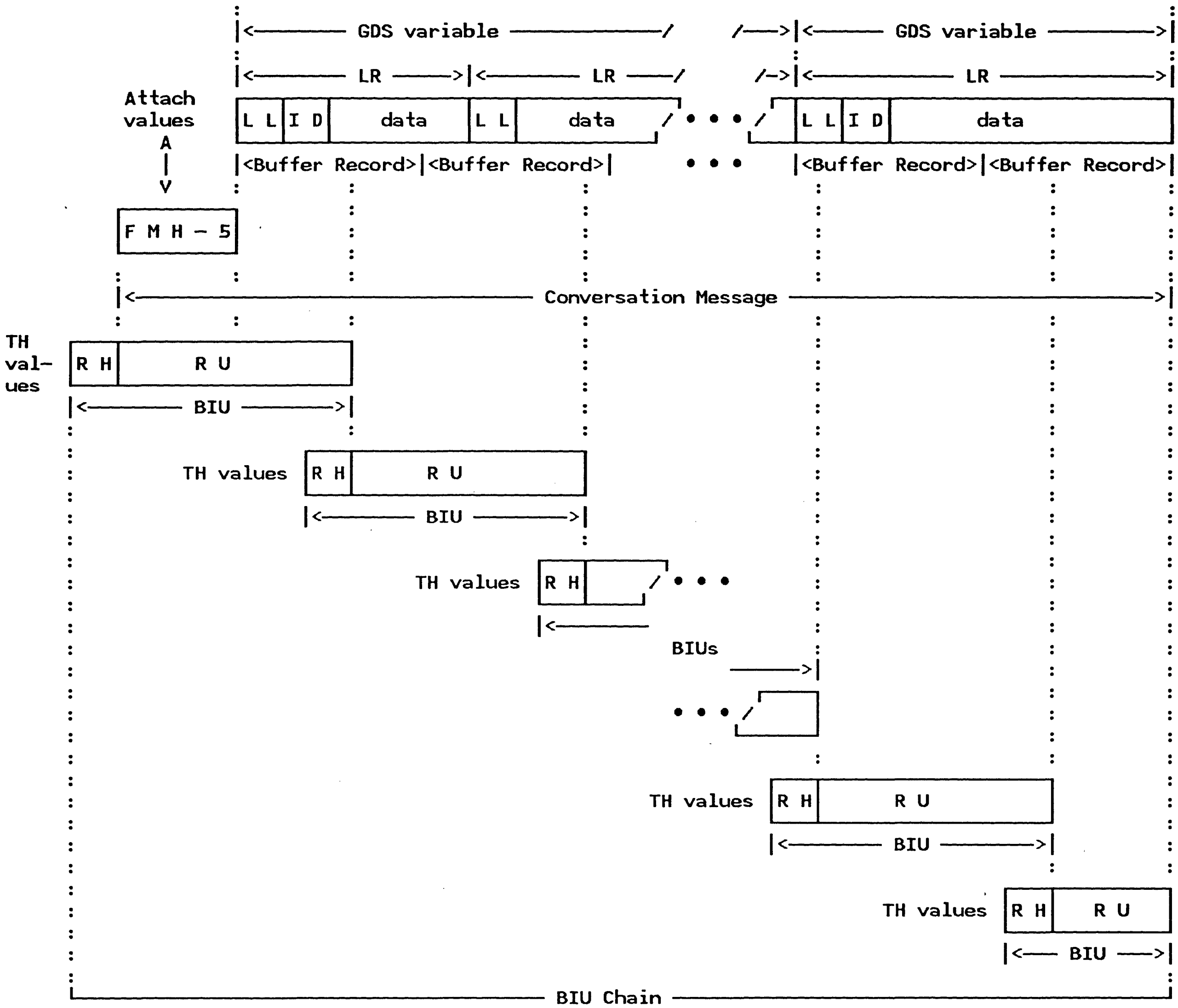
DATA EXCHANGE WITH THE CP

The LU also exchanges message units with the CP. These message units are listed in "Chapter 4. LU Session Manager" and are described briefly below.

LU-CP Records

In the model described in this book, the LU has a direct protocol boundary with the CP in its node.

The LU generates and uses session control RUs for session activation and deactivation. It sends these to the CP for routing to the remote LU.



LEGEND:

LR: logical record LL: Length field ID: GDS ID field
RH: request header RU: request unit BIU: basic information unit
FMH-5: Attach FM header (occurs only on first conversation-message of conversation)
Attach values: information for the Attach FM header, from the ALLOCATE verb.
TH values: protocol information generated by the LU; the TH is built by path control.

Figure 2-6. Relationship of Conversation Message to BIU Chain (Example)

EXTERNAL FLOW SEQUENCES FOR THE BASE FUNCTION SET

This section illustrates the correspondence between some typical basic-function-set transaction program verb sequences and the resulting flows of BIUs through the path control network. (The verbs are described in detail in SNA Transaction Programmer's Reference Manual for LU Type 6.2).

The correspondence is illustrated in Figure 2-7 on page 2-20 through Figure 2-23 on page 2-27. In the figures, the left column shows verbs issued by the invoking or initially-sending TP, and the right column shows verbs issued in response by the invoked or initially-receiving TP. The center column shows the contents of the resulting chain (RH indicator settings, RU data and FM headers).

The arrows indicate direction of BIU flow. A group of arrows in the same direction represents a chain, but no necessary correspondence exists between arrows in the figures and BIUs in the chain.

Each figure shows one of the following:

- The beginning of a chain, for chains that begin a bracket
- The end of one chain and the beginning of the next
- The end of a chain, for chains that end a bracket

"Allowable Combinations of Sequences" on page 2-22 shows how these flows can be combined, or sequenced, to form complete conversations.

Finally, "Error Flows" on page 2-24 shows asynchronous response cases.

NOTATION

The following notation is used in the figures. RH indicators:

The flow is labeled with the indicator values that are carried in the RH.

BB Begin bracket

CEB Conditional end of bracket

BC Begin chain

EC End chain

RQE1 Request exception response 1

RQE2 Request exception response 2 (in this case, DR1I = DR1|¬DR1; i.e., RQE3 is equivalent to RQE2).

RQD1 Request definite response 1

RQD2 Request definite response 2 (in this case, DR1I = DR1|¬DR1; i.e., RQD3 is equivalent to RQD2).

CD Change direction

+DR2 Positive response to RQD2

-RSP(0846) Negative response to chain

RU contents:

FMH-5 Attach FM header

FMH-7 Error-description FM header

The sense-data categories shown are:

0864 Abnormal deallocation

0889 Program-detected error

data User data in FMD RU

Verbs and Parameters

The returned RETURN_CODE parameter of the RECEIVE_AND_WAIT verb is not shown when it is set to OK; in that case, the returned WHAT_RECEIVED parameter is shown instead.

DATA_* represents either setting (DATA_COMPLETE or DATA_INCOMPLETE) of this parameter.

Data Transfer Description

Whenever a TP has the right to send, it issues SEND_DATA zero or more times. Similarly, a TP in receive state repeatedly issues RECEIVE_AND_WAIT, until it receives all of the data and the end-of-conversation-message indication. The receiver issues at least one receive verb; in the absence of errors, zero or more initial issuances of SEND_DATA by the source TP result in zero or more receive verb issuances (with WHAT_RECEIVED = DATA_INCOMPLETE) at the target. The final issuance receives the end-of-conversation-message indicator as WHAT_RECEIVED = DATA_COMPLETE. Since the buffer record sizes used at the sending TP and at the receiving TP may differ, the number of receive verb issuances does not necessarily match the number of send verb issuances.

All of the following figures begin or end with the data-transmission sequence just described. That sequence is represented in the figures as follows.

When the figure begins with (the end of) the data-transmission sequence, it shows (at the sending TP) a single SEND_DATA verb, and a corresponding data arrow, followed by vertical (two-dot) ellipsis marks (:). No RECEIVE_AND_WAIT verb is shown at the receiving TP.

When the figure ends with (the beginning of) the data-transmission sequence, it shows (at the receiving TP) vertical ellipsis marks (:), followed by a single RECEIVE_AND_WAIT verb with WHAT_RECEIVED = DATA_COMPLETE. "Data" is shown on the corresponding arrow, along with the end-of-conversation-message RH indicators. No SEND_DATA verb is shown at the beginning of the receiving-TP verb sequence.

ERROR-FREE FLOWS

The error-free flows for the base function set flows are described in terms of the verb sequences shown in Figure 2-7 on page 2-20 through Figure 2-14 on page 2-22.

```
SEQUENCE 1

 ALLOCATE
   SYNC_LEVEL(NONE)   BC,BB,FMH-5
                    ---------------------> (TP started)
 SEND_DATA                 data
                    --------------------->
   :                       :                   :
```

Figure 2-7. Start Conversation with Synchronization Level of NONE

```
SEQUENCE 2
   :                       :                   :
 PREPARE_TO_RECEIVE   EC,RQE1,CD,data       RECEIVE_AND_WAIT
   TYPE(FLUSH)      --------------------->    WHAT_RECEIVED=DATA_COMPLETE
                                            RECEIVE_AND_WAIT
                                              WHAT_RECEIVED=SEND
                       BC,data              SEND_DATA
                   <---------------------
   :                       :                   :
```

Figure 2-8. Conversation Turnaround without Confirmation: PREPARE_TO_RECEIVE is optional; when it is omitted, and a receive verb is issued from SEND state, the function of PREPARE_TO_RECEIVE is performed before any data is actually received.

```
SEQUENCE 3
   :                       :                   :
 DEALLOCATE           EC,RQE1,CEB,data      RECEIVE_AND_WAIT
   TYPE(FLUSH)      --------------------->    WHAT_RECEIVED=DATA_COMPLETE
      (local deallocation)                  RECEIVE_AND_WAIT
                                              RETURN_CODE=DEALLOCATE_NORMAL
                                            DEALLOCATE
                                              TYPE(LOCAL)
                                                 (local deallocation)
```

Figure 2-9. Finish Conversation without Confirmation

```
SEQUENCE 4

 ALLOCATE              BC,BB,FMH-5
   SYNC_LEVEL(CONFIRM)-------------------> (TP started)
 SEND_DATA                 data
                    --------------------->
   :                       :                   :
```

Figure 2-10. Start Conversation with Synchronization Level of CONFIRM: The SYNC_LEVEL parameter on ALLOCATE establishes the synchronization level for the entire conversation. References to SYNC_LEVEL on subsequent conversation verbs refer to the SYNC_LEVEL value established at the start of the conversation. In this sequence, that value is CONFIRM.

```
SEQUENCE  5
   :                          :                  :
 CONFIRM               EC,RQD2,¬CD,data      RECEIVE_AND_WAIT
                      ---------------------->  WHAT_RECEIVED=DATA_COMPLETE
                                             RECEIVE_AND_WAIT
                                               WHAT_RECEIVED=CONFIRM
                             +DR2            CONFIRMED
   RETURN_CODE=OK     <----------------------
 SEND_DATA             BC,data
                      ---------------------->
   :                          :                  :
```

Figure 2-11. Continue Conversation: Confirmation without Turnaround

```
SEQUENCE 6A
   :                          :                  :
 PREPARE_TO_RECEIVE                            RECEIVE_AND_WAIT
   TYPE(SYNC_LEVEL)    EC,RQD2,CD,data
   LOCKS(SHORT)       ----------------------->   WHAT_RECEIVED=DATA_COMPLETE
                                               RECEIVE_AND_WAIT
                                                 WHAT_RECEIVED=CONFIRM_SEND
                             +DR2              CONFIRMED
   RETURN_CODE=OK     <-----------------------
                        BC,data                SEND_DATA
                      <-----------------------
   :                          :                  :
```

Figure 2-12. Conversation Turnaround with SYNC_LEVEL = CONFIRM, using LOCKS(SHORT): When the receiving TP issues CONFIRMED after the LU has received RQD2--indicating CONFIRM LOCKS(SHORT)--the LU immediately sends a CONFIRMED response (+DR2). This allows the CONFIRM sender to resume processing immediately, so that, for example, it can release locks on its local resources.

(The receiving LU processes the RQD2 internally; it does not inform the receiving TP of the LOCKS parameter value.)

```
SEQUENCE 6B
   :                          :                  :
 PREPARE_TO_RECEIVE                            RECEIVE_AND_WAIT
   TYPE(SYNC_LEVEL)    EC,RQE2,CD,data
   LOCKS(LONG)        ----------------------->   WHAT_RECEIVED=DATA_COMPLETE
                                               RECEIVE_AND_WAIT
                                                 WHAT_RECEIVED=CONFIRM_SEND
                                               CONFIRMED
                                           (LU omits sending +DR2)
                        BC,data                SEND_DATA
   RETURN_CODE=OK     <-----------------------
   :                          :                  :
```

Figure 2-13. Conversation Turnaround with SYNC_LEVEL = CONFIRM, using LOCKS(LONG): When the receiving TP issues CONFIRMED after the LU has received RQE2--indicating CONFIRM LOCKS(LONG)--the LU does not send an immediate confirmation response. Instead, it continues processing until it has a complete BIU to send. The CONFIRM sender interprets receipt of BC without an intervening response as positive confirmation.

LOCKS(LONG) does not require the +DR2 response BIU that LOCKS(SHORT) requires, but it can cause the CONFIRM sender to wait longer before resuming processing.

```
SEQUENCE 7
  :                           :                  :
 DEALLOCATE           EC,RQD2,CEB,data        RECEIVE_AND_WAIT
   TYPE(SYNC_LEVEL)  ----------------------->   WHAT_RECEIVED=DATA_COMPLETE
                                              RECEIVE_AND_WAIT
                                                WHAT_RECEIVED=CONFIRM_DEALLOCATE
                            +DR2              CONFIRMED
   RETURN_CODE=OK    <----------------------
     Local Deallocation                       DEALLOCATE
                                                TYPE(LOCAL)
```

Figure 2-14. Finish Conversation, SYNC_LEVEL = CONFIRM

ALLOWABLE COMBINATIONS OF SEQUENCES

When a program issues one of the verb sequences shown above, that program is limited in its choice of the next verb sequence it can issue. The matrix in Figure 2-15 shows which verb sequences can follow a given verb sequence in the base function set. The matrix has the following meaning:

- The row numbers (left column) and column numbers (top row) in the matrix correspond to the sequence numbers in Figure 2-7 on page 2-20 through Figure 2-14.

  A row corresponds to the verb sequence just issued; a column corresponds to the verb sequence issued next.

  In the matrix, row 0 or column 0 represents the state in which no conversation exists, i.e., the state prior to ALLOCATE or subsequent to DEALLOCATE.

- A letter N or C in a cell indicates that the sequence corresponding to the column number can follow the sequence corresponding to the row number.

  - N--indicates a next sequence allowed for conversations allocated with either SYNC_LEVEL(NONE) or SYNC_LEVEL(CONFIRM), i.e., conversations started with sequences 1 or 4

  - C--indicates a next sequence allowed only for conversations allocated with SYNC_LEVEL(CONFIRM), i.e., conversations started with sequence 4

  - empty--indicates that the corresponding sequence order is invalid

- The Next-Sender column indicates which TP is initial sender (i.e., issues the verbs in the left column of the figure) for the next sequence:

  - SAME--the initial sender of the next sequence is the same as the initial sender of the previous sequence.

  - OTHER--the initial sender of the next sequence is the partner of the initial sender of the previous sequence.

Figure 2-16 on page 2-23 and Figure 2-17 on page 2-23 illustrate the application of these rules to generate allowable conversation sequences.

	0	1	2	3	4	5	6A	6B	7	Next-Sender
0		N			C					
1			N	N						SAME
2			N	N		C	C	C	C	SAME
3	N									
4			C	C		C	C	C	C	SAME
5			C	C		C	C	C	C	SAME
6A			C	C		C	C	C	C	OTHER
6B			C	C		C	C	C	C	OTHER
7	C									

Figure 2-15. Possible Next Sequence in Error-Free Cases

```
ALLOCATE
   SYNC_LEVEL(NONE)    BC,BB,FMH-5
                      ----------------------> (TP started)
SEND_DATA              data                   RECEIVE_AND_WAIT
                      ---------------------->   WHAT_RECEIVED=DATA_*
SEND_DATA                                     RECEIVE_AND_WAIT
DEALLOCATE             EC,RQE1,CEB,data         WHAT_RECEIVED=DATA_COMPLETE
   TYPE(FLUSH)        ----------------------> RECEIVE_AND_WAIT
      (local deallocation)                      RETURN_CODE=DEALLOCATE_NORMAL
                                              DEALLOCATE
                                                TYPE(LOCAL)
                                                   (local deallocation)
```

Figure 2-16. One-Way Conversation without Confirmation: Combines Sequences 1 and 3

The sequence shown in Figure 2-16 is generated as follows:

1. Begin in state 0.

2. Select a column containing a lettered cell in row 0.

   In this example, column 1 was chosen. This corresponds to sequence 1.

3. Supply an arbitrary number of SEND_DATA and RECEIVE_AND_WAIT verbs following sequence 1, as allowed by the the data-transfer convention.

   In this example, the ellipsis was replaced by one additional issuance of SEND_DATA and one additional issuance of RECEIVE_AND_WAIT.

4. Select a column containing an N in row 1.

   In this example, column 3 was chosen.

5. Orient sequence 3 according to the "next sender" column for the previous sequence.

   In this example, the next sender is SAME, so the left column of sequence 3 is issued by the same TP as the left column of sequence 1.

6. Select a column containing an N in row 3. The only choice is column 0, indicating the end of the sequence.

```
 ALLOCATE               BC,BB,FMH-5
   SYNC_LEVEL(CONFIRM) --------------------->(TP started)
 PREPARE_TO_RECEIVE    EC,RQE2,CD            RECEIVE_AND_WAIT
   TYPE(SYNC_LEVEL)   ---------------------->  WHAT_RECEIVED=CONFIRM_SEND
   LOCKS(LONG)                               CONFIRMED
                        BC,data              SEND_DATA
   RETURN_CODE=OK     <----------------------
 RECEIVE_AND_WAIT
    WHAT_RECEIVED=     EC,RQD2,CEB,data      DEALLOCATE
      DATA_COMPLETE   <---------------------   TYPE(SYNC_LEVEL)
  RECEIVE_AND_WAIT
    WHAT_RECEIVED=
     CONFIRM_DEALLOCATE
  CONFIRMED                     +DR2
                       --------------------->  RETURN_CODE=OK
  DEALLOCATE
    TYPE(LOCAL)
```

Figure 2-17. Two-Way Conversation with Confirmation: Combines Sequences 4, 6B, and 7.

The sequence shown in Figure 2-16 is generated as follows:

1. Beginning in state 0, select sequences 4, 6B, and 7, returning to state 0.

2. Supply some number of SEND_DATA and RECEIVE_AND_WAIT verbs following sequence 4.

   In this example, 0 instances of SEND_DATA were chosen. Thus, following the data transfer convention, the SEND_DATA verb and data arrow in sequence 4 are eliminated, as is the RECEIVE_AND_WAIT WHAT_RECEIVED = DATA_COMPLETE and the data on the EC arrow in sequence 6B.

3. The next sender following sequence 4 is SAME; therefore, sequence 6B has the same orientation as the preceding sequence.

4. Supply some number of SEND_DATA and RECEIVE_AND_WAIT verbs following sequence 6B.

   In this example, only one instance of each was chosen, corresponding exactly to the number in the sequence figures.

   (This figure illustrates that the arrows do not necessarily correspond to BIUs. For example, the CONFIRM, SEND_DATA, and DEALLOCATE might generate only one BIU, even though two arrows are shown in the figure.)

5. The next sender following sequence 6B is OTHER; therefore, sequence 7 is reversed to have the opposite orientation from that of the preceding sequence (i.e., since the left column of sequence 6B corresponds to the left column of the combined sequence, the left column of sequence 7 corresponds to the right column of the combined sequence).

6. The next row number is 0; therefore this completes the sequence.

```
  :                                 :                     :
SEND_DATA                 data                          RECEIVE_AND_WAIT
                 ---------------------------------->      WHAT_RECEIVED=DATA_*
SEND_DATA        BC,EC,SIGNAL (expedited flow)          REQUEST_TO_SEND
                 <----------------------------------
  REQUEST_TO_SEND_RECEIVED=YES                          RECEIVE_AND_WAIT
PREPARE_TO_RECEIVE   EC,RQE1,CD,data
  TYPE(FLUSH)    ---------------------------------->     WHAT_RECEIVED=DATA_COMPLETE
                                                        RECEIVE_AND_WAIT
                                                         WHAT_RECEIVED=SEND
RECEIVE_AND_WAIT       BC,data                          SEND_DATA
                 <----------------------------------
  WHAT_RECEIVED=DATA_*
  :                                 :                     :
```

Figure 2-18. Conversation Turnaround following REQUEST_TO_SEND (without Confirmation): REQUEST_TO_SEND issued by the receiving TP results in an expedited-flow one-RU chain. The TP sending data is notified via the REQUEST_TO_SEND_RECEIVED parameter of a subsequent verb. The interpretation of REQUEST_TO_SEND_RECEIVED is determined by the TP. In this example, the sending TP stops sending and issues RECEIVE_AND_WAIT.

EXCEPTION FLOW

Figure 2-18 illustrates the only non-error case for which a TP can send while in receive state. This flow represents issuing the REQUEST_TO_SEND verb and sending the SIGNAL RU.

This flow can be substituted for sequence 2. A similar sequence corresponding to sequence 6A or 6B exists, but is not illustrated here.

ERROR FLOWS

Figure 2-19 on page 2-25 through Figure 2-23 on page 2-27 illustrate flows resulting from transaction-program error recovery for the base function set. When the TP detects a TP-defined error (e.g., the received data fails an application validity check, or the partner sends more logical records than expected) it issues SEND_ERROR or DEALLOCATE TYPE(ABEND). When the LU detects a transaction program error, such as an Attach failure, it generates similar flows.
Three cases exist:

- Verb issued by sender
- Verb issued by receiver
- Verb issued by both (e.g., a SEND_ERROR race has occurred)

  (This case is not illustrated for DEALLOCATE.)

For cases not shown here, see "Component Interactions and Sequence Flows" on page 2-48.

```
  :                            :                   :
SEND_DATA
(TP detects                                      RECEIVE_AND_WAIT
 an error)
SEND_ERROR           data
                   ------------------------->  WHAT_RECEIVED=DATA_INCOMPLETE
SEND_DATA           FMH-7(0889),data         RECEIVE_AND_WAIT
                   ------------------------->    WHAT_RECEIVED=PROGRAM_ERROR_TRUNC
  :                            :                   :
```

Figure 2-19. SEND_ERROR Issued by Sender: The SEND_ERROR verb forces sending of accumulated data and begins a new RU with an FMH-7. The issuing TP remains in send state; it can, for example, send additional TP-determined data to further describe the error.

```
  :                            :                   :
SEND_DATA                data                 RECEIVE_AND_WAIT
                 --------------------->         WHAT_RECEIVED=DATA_*
                                              (TP detects an error)
                          -RSP(0846)          SEND_ERROR
                              ---------
SEND_DATA               data|                   Purge incoming BIUs
                 -----------|--------->           to end of chain
                            |                   "
  :                         :      :            :       :
                            |                   "
(LU ends chain) <-----------                    "
                  EC,RQE1,CD,no data            "
                 --------------------->         " (LU detects end of chain)
                                                RETURN_CODE=OK
                  BC,FMH-7(0889),data         SEND_DATA
                 <---------------------
   RETURN_CODE=
     PROG_ERROR_PURGING
RECEIVE_AND_WAIT
  :                            :                   :
```

Figure 2-20. SEND_ERROR Issued by Receiver: The SEND_ERROR verb causes a negative response to the incoming chain; the sending LU sends End-of-Chain and Change-Direction when it receives the response. Meanwhile, the receiver purges incoming RUs until the End-of-Chain indication is received, then it sends FMH-7 and leaves the issuing TP in send state so it can, for example, send additional TP-determined data describing the error.

```
  :                          :                   :
SEND_DATA               data              RECEIVE_AND_WAIT
             ----------------------->       WHAT_RECEIVED=DATA_*
                                          (TP detects an error)
(TP detects              -RSP(0846)       SEND_ERROR
 an error)                  ┌-----------
SEND_ERROR          data    |               Purge incoming BIUs
             ---------------|----------->     to end of chain
SEND_DATA            FMH-7(0889),data       "
             ---------------|----------->   "
                            |               "
  :                         :    :          :       :
                            |               "
(LU ends chain)   <---------┘               "
              EC,RQE1,CD,no data            "
             --------------------->         " (LU detects end of chain)
                                              RETURN_CODE=OK
              BC,FMH-7(0889),data          SEND_DATA
             <---------------------
  RETURN_CODE=
    PROG_ERROR_PURGING
RECEIVE_AND_WAIT
  :                          :                   :
```

Figure 2-21. SEND_ERROR Issued by both Sender and Receiver (SEND_ERROR Race): Each LU begins SEND_ERROR processing as in the no-race case, but since the receiver is purging to end of chain, the SEND_ERROR from the sender is also purged, so the receiver's SEND_ERROR takes precedence.

```
  :                          :                      :
SEND_DATA                                     RECEIVE_AND_WAIT
DEALLOCATE          data
  TYPE(ABEND_PROG) ------------------------>     WHAT_RECEIVED=DATA_*
                    EC,RQD1,CEB,FMH-7(0864)   RECEIVE_AND_WAIT
                   ------------------------>     RETURN_CODE=
                     +DR1                           DEALLOCATE_ABEND_PROG
    (response used <------------------------
     internally)
```

Figure 2-22. DEALLOCATE ABEND Issued by Sender: The flow is similar to SEND_ERROR in send state. The +DR1 response is required for internal processing.

```
   :                          :                         :
 SEND_DATA                  data                     RECEIVE_AND_WAIT
               ------------------->                    WHAT_RECEIVED=DATA_*
                          -RSP(0846)                 DEALLOCATE
                           ---------                   TYPE(ABEND_PROG)
 SEND_DATA              data|                          Purging
               ----------  |-------->                  "
                            |                          "
   :                        :     :                    :      :
                            |                          "
 (LU ends chain)      <-----                           "
                    EC,RQE1,CD,no data                 "
               ------------------->                    "(LU detects end of chain)
                BC,EC,RQD1,CEB,FMH-7(0864)
               <-------------------
   RETURN_CODE=
     DEALLOCATE_ABEND_PROG
                +DR1
               ------------------->                         (response used internally)
   :                          :                         :
```

Figure 2-23. DEALLOCATE ABEND Issued by Receiver: The flow is similar to SEND_ERROR in receive state. The +DR1 response is required for internal processing.

LU STRUCTURE

Figure 2-24 on page 2-28 illustrates the structure of the LU.

The upper protocol boundary of the LU is the transaction program protocol boundary (described in SNA Transaction Programmer's Reference Manual for LU Type 6.2). A transaction program processes end user data, and requests LU services to communicate with other transaction programs.

The lower protocol boundary of the LU is to the SNA path control network, which the LU uses to communicate with other LUs.

The LU also has protocol boundaries with the CP, buffer manager, NOF, and with initiator processes in the same node.

SNA LAYERS

The LU contains instances of the following four SNA layers:

- Transaction services
- Presentation services
- Data flow control
- Transmission control

Component Overview

The LU has two groups of components, one for its upper protocol boundary with transaction programs, and one for its lower protocol boundary with the path control network. Each group consists of a set of processes containing a pair of SNA layer-instances, and a manager component that creates, destroys, and otherwise manages these instances.

The upper group of components contains transaction processes, which contain instances of the following SNA layers:

- Transaction services
- Presentation services

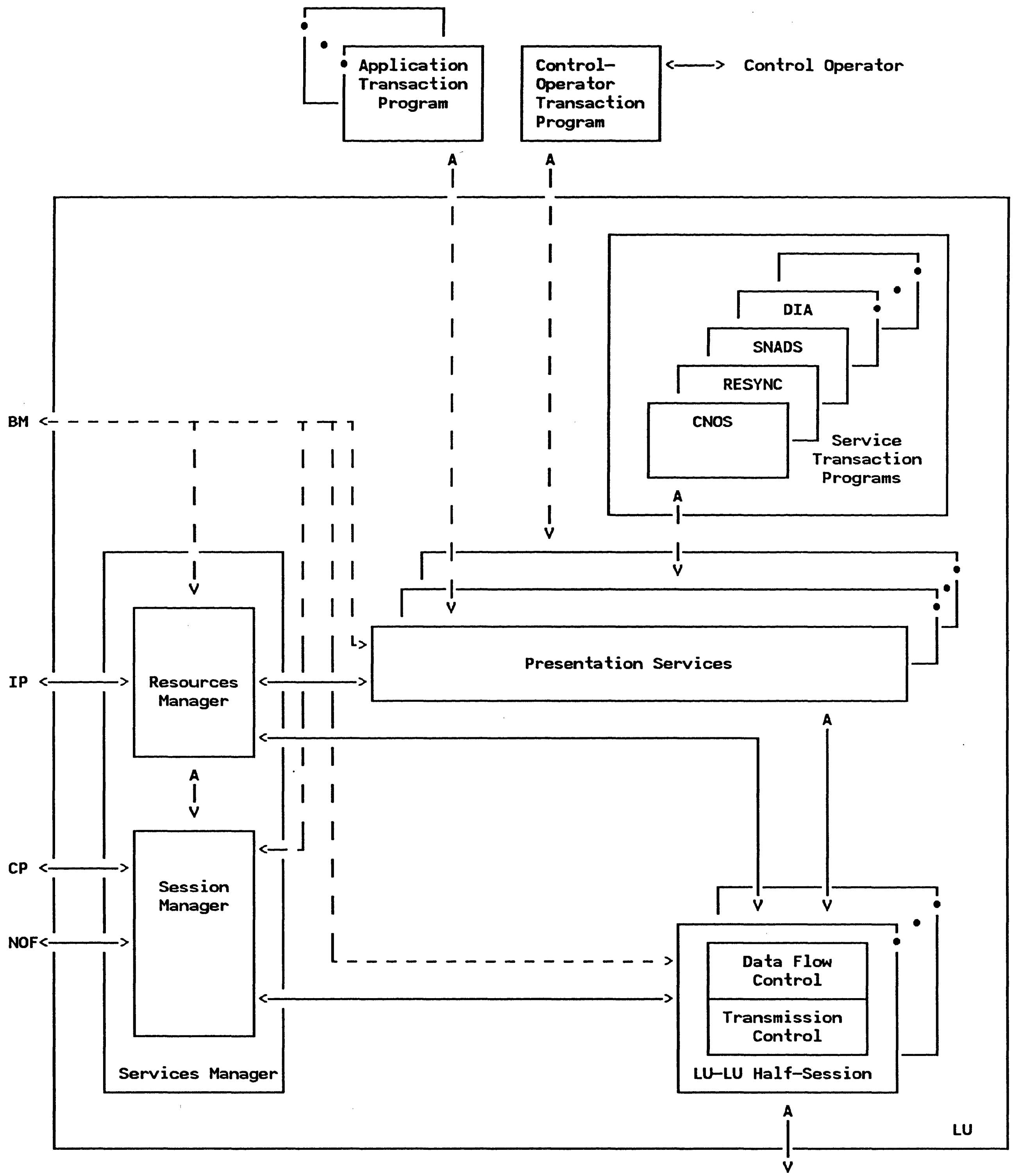
More concretely, each transaction process contains an execution instance of a transaction program and some presentation services components for processing the verbs issued by it. (See Figure 2-25 on page 2-29.)

This group of components is managed by the resources manager component (RM), which creates transaction processes (in response to Attaches received from remote LUs), destroys them after they have finished executing, and connects them with sessions (thus enabling them to participate in distributed transactions).

The lower group of components contains half-sessions (HSs), which contain instances of the following SNA layers:

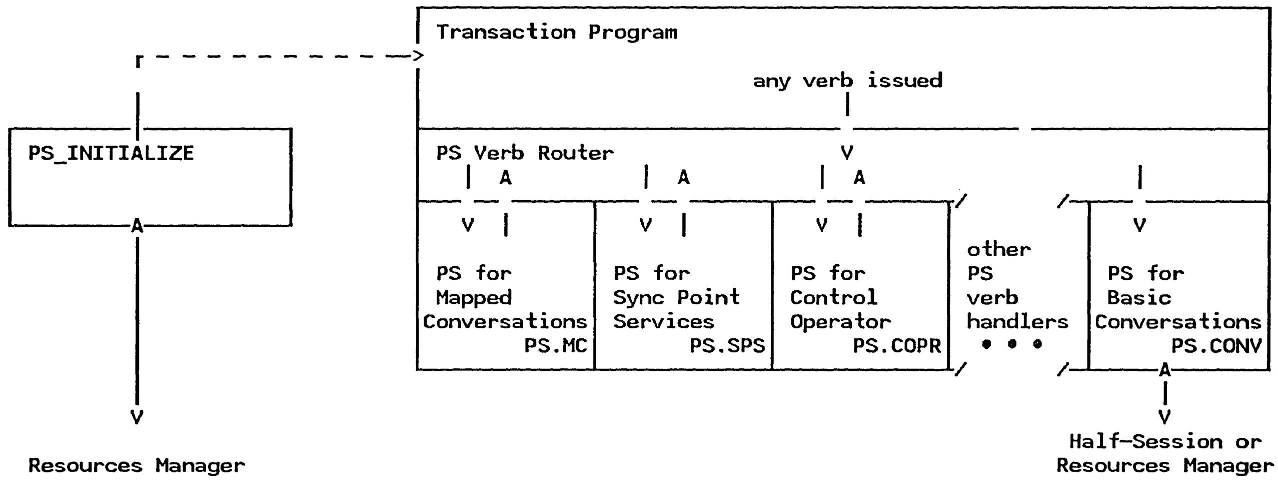
- Data flow control
- Transmission control

Half-sessions enforce protocol rules for conversation data exchange, and transform mes-



LEGEND:
<———> Send/Receive relationship
<- - -> Call/Return relationship
CNOS: Change Number of Sessions
SNADS: SNA Distribution Services
CP: Control Point
NOF: Node Operator Facility
RESYNC: Sync Point Resynchronization
DIA: Document Interchange Architecture Services
IP: Initiator process
BM: Buffer Manager

Figure 2-24. Overview of LU 6.2 Components



LEGEND:
- - -> Call/Return relationship (within a process)
<———> Send/Receive relationship (between processes)
Note: PS verb router is called recursively by PS verb handlers.

Figure 2-25. Structure of a Presentation Services Process

sage units between the format useful to conversing programs and the format appropriate for the path control network (this includes implementing session services such as pacing and cryptography).

This group of components is managed by the session manager component (SM), which creates and destroys half-sessions and interacts with components outside the LU (e.g., the control point).

The session manager component is created by the node operator facility (NOF) when it activates the LU. The session manager component then creates the resources manager component. They run continuously thereafter. (For more information see SNA Type 2.1 Node Reference and "Chapter 4. LU Session Manager".)

FUNCTIONAL SUMMARY BY FUNCTION

This is the first of two sections describing the functions and interactions of LU components. This section is organized by function; it concentrates on functions that involve multiple components. For each function, it explains in approximate time sequence the roles of the various LU components. The next section is organized by component, and covers functions performed principally by one component. A full description of each component is given in its corresponding chapter of this book.

For illustrations of the component interactions discussed in this section, including a variety of cases not discussed elsewhere in this chapter, see "Component Interactions and Sequence Flows" on page 2-48. In particular, Figure 2-33 on page 2-50 and Figure 2-34 on page 2-51 illustrate the interactions, at the source and target LUs, respectively, for a typical conversation; Figure 2-35 on page 2-52 and Figure 2-36 on page 2-53 illustrate typical interactions for session deactivation.

The LU manages the state and configuration of its local resources, including transaction programs, conversation resources, and half-sessions. It cooperates with other LUs, using shared sessions and conversations, to configure these resources to support distributed transactions. (An LU implementation might also manage other, non-SNA, resources such as processor execution cycles, storage, and data bases.)

The principal functions leading to LU transaction processing are the following, not necessarily performed in this order:

- Activating sessions between two LUs
- Invoking transaction programs
- Initiating conversations between the transaction programs
- Transferring message units between the transaction programs

EXAMPLE TRANSACTION PROGRAM

Figure 2-26 on page 2-30 outlines some typical verb issuances for an example pair of transaction programs.

SOURCE TP	TARGET TP
MC_ALLOCATE	
MC_SEND_DATA	MC_RECEIVE_AND_WAIT
"	"
"	"
"	"
MC_RECEIVE_AND_WAIT	"
	MC_SEND_DATA
"	"
"	"
"	MC_DEALLOCATE
MC_DEALLOCATE	

Figure 2-26. Example of Communicating Transaction Programs

The programs, running at different LUs, issue complementary sequences of verbs. The LUs convert these executed verbs into message-unit flows.

MESSAGE-UNIT TRANSFER

First, consider transfer of message units. Assume that two transaction programs are running at their respective LUs and are connected by a mapped conversation. For the programs to transfer data, one program must issue MC_SEND_DATA verbs while the other issues complementary MC_RECEIVE_AND_WAIT verbs.

The TP invokes PS for each transaction-program verb it issues. PS performs the function appropriate to the specific verb. For each verb, PS verifies that the verb is valid in the current conversation state, converts the verb parameters to an intermediate representation, and performs verb-specific processing that includes issuing appropriate requests to other LU components.

When sending, the data specified on the MC_SEND_DATA verb along with the chaining indicators is put into MUs by PS and sent to HS. HS encodes the protocol information into RHs and passes the resulting BIUs (with TH information) to path control.

When receiving, HS checks incoming BIUs for format and protocol validity and passes MUs containing data to PS. When the TP issues a MC_RECEIVE_AND_WAIT verb, PS checks the verb for validity and passes the requested amount of data and protocol information back to the TP.

The following sections discuss these functions in more detail. (Figure 2-4 on page 2-16, Figure 2-5 on page 2-17, and Figure 2-6 on page 2-18 illustrate the message-unit relationships discussed.)

Sending Data

For MC_SEND_DATA, PS verifies that the conversation is in send state. If mapping is being performed, PS maps the transaction-program data record into a mapped-conversation record (see "Mapping Function" on page 2-36). It transforms the MCR into a sequence of logical records of implementation-defined length by segmenting the supplied data and prefixing the appropriate GDS LLID fields. It calls the basic conversation SEND_DATA procedure as often as necessary (determined by the buffer-record size used by the PS.MC implementation) to send all the logical records. The mapped-conversation verb handlers in PS typically call one or more basic-conversation procedures to perform the function requested by a mapped-conversation verb.

When PS has first entered send state, it expects an LL at the beginning of the first buffer record. From then on, PS compares the accumulated length of the data passed on successive issuances of MC_SEND_DATA to the logical-record lengths specified in the LLs, thus verifying that the conversation message sent ends at a logical record boundary.

PS accumulates the data from successive buffer records in RU-sized units (the RU size for a session is determined by BIND negotiation when the session is activated). When the RU-size buffer is full, PS transfers the data to HS with an indication of whether it is the last of the data for a conversation message. When PS detects the end of a conversation message, e.g., a PREPARE_TO_RECEIVE, MC_RECEIVE_AND_WAIT, CONFIRM, SYNCPT, or MC_DEALLOCATE verb was issued, PS transfers its remaining accumulated data with an indication of how the conversation message was ended, e.g., confirmation request, conversation turnaround, or deallocation. It also places the conversation in the appropriate state.

Meanwhile, the HS process, also in send state, waits for data from PS. When PS passes the data, HS fills in the RH, a sequence number, and other TH information. If session cryptography is being used, HS enciphers the data.

HS encodes each RH to indicate the beginning or end of a bracket (corresponding to a complete conversation exchange) and the beginning or end of a chain (corresponding to a conversation message). For all but the last BIU in a chain, HS encodes the RH with RQE1.

For the last BIU for the conversation message, HS encodes the RH with EC (the end-of-conversation-message indicator) and other indicators selected by PS, such as CD (e.g., MC_PREPARE_TO_RECEIVE verb issued), RQD2 (e.g., MC_CONFIRM issued), RQD1

(MC_DEALLOCATE TYPE[ABEND]) issued), and CEB (MC_DEALLOCATE issued). HS changes the local session state accordingly.

HS passes each completed BIU and the corresponding TH information to path control for transmission to the receiving HS in the remote LU.

HS enforces both fixed and adaptive session-level pacing. The type of pacing for a given session is determined during session initiation and is communicated to HS when initialized by SM.

In fixed pacing, the sending HS sends at most one fixed-sized pacing window of BIUs before receiving a pacing response. It then requires a pacing response from the receiver before sending another window. The receiving HS sends a pacing response when it can receive another pacing window, e.g., when it has enough free buffers.

In adaptive pacing, the pacing window size varies depending upon the availability of buffers at the receiving node and the demand for buffers at the sending node. The availability of buffers is determined and controlled by the buffer manager (BM) at the receiving node.

The sending HS asks the receiving HS for the next-window size by setting the Pacing indicator to 1 (PAC) in the RH of the first message in a window. Also in the same RH, the sending HS may ask for a larger window size by setting the Request Larger Window indicator to 1 (RLW).

The receiving HS calculates the next-window size based upon the number of buffers given to it by the buffer manager and the demand for buffers by the sending HS. The receiving HS sends the next-window size to the sending HS in a pacing message (IPM), which corresponds to the pacing response (IPR) in fixed pacing. When additional buffers are available and the Request Larger Window indicator is RLW, the window size is increased. When additional buffers are not available, the window size remains the same. When fewer buffers are available, the window size is decreased. When buffers become critically scarce, the buffer manager prompts the receiving HS to send an unsolicited pacing message to the sending HS, which causes the sending HS to set its current-window size and next-window size to 0, thus stopping the sending HS from sending BIUs. When buffers again become available, the buffer manager prompts the receiving HS to send another pacing message with a next-window size greater than 0 to the sending HS. This allows the sending HS to resume sending BIUs. For more information on session-level pacing see "Chapter 6.2. Transmission Control".

Receiving Data

The HS process at the receiving LU receives BIUs and TH information from path control. It sends IPRs or IPMs when it has sufficient buffers to receive additional BIUs. If session cryptography is specified, it deciphers the data. It checks for correct session protocol. It checks BIU sequence numbers to detect lost or duplicate BIUs and to correlate responses with the correct bracket. If it detects any protocol error, it abnormally deactivates the session, i.e., it requests SM to issue UNBIND indicating a format or protocol error.

If the BIU is satisfactory, HS sends the MUs containing the Security FM header or the Attach FM header, if present, to RM, and sends all other MUs to PS. HS also sends PS an indication of significant state changes that were encoded in the received RH such as end of a conversation message (End-of-Chain), enter send state (Change-Direction), confirmation request (Definite-Response 2|3) and end of conversation (Conditional-End-of-Bracket). HS changes its own session state accordingly.

Meanwhile, the receiving TP issues MC_RECEIVE_AND_WAIT verbs to receive the conversation message. Each verb issuance calls PS.

For each MC_RECEIVE_AND_WAIT issuance, PS calls the basic conversation RECEIVE_AND_WAIT procedure until it receives enough data (in the form of logical records) to satisfy the data length requested on the MC_RECEIVE_AND_WAIT verb.

For each RECEIVE_AND_WAIT verb issuance (including the case in which RECEIVE_AND_WAIT is issued directly by a transaction program, i.e., for a basic conversation), PS waits for the data from HS. PS receives data from HS in the form of MUs. If more MUs are received than are currently necessary to satisfy a RECEIVE_AND_WAIT, PS queues the MUs.

While parsing the MUs to satisfy the RECEIVE_AND_WAIT, PS keeps track of the LL fields, to verify that the conversation message ends on a logical record boundary.

When the RECEIVE_AND_WAIT procedure returns to the MC_RECEIVE_AND_WAIT procedure, PS checks the length and continuation fields in the LLs to verify that a complete mapped-conversation record (MCR) has been received, strips the GDS LL and ID fields, and reblocks the data into an MCR. (If the TP receive buffer cannot contain the complete MCR, PS passes it to the TP in receive-buffer-sized segments, i.e., mapped-conversation buffer records.)

If PS receives an end-of-conversation-message indication, it does not forward this indication to the TP until after all logical records and MCRs have been received. It then returns the end-of-conversation-message indication alone on the next MC_RECEIVE_AND_WAIT verb issued, and places the mapped conversation into the appropriate state.

TRANSACTION PROGRAM INITIATION AND TERMINATION

Before the TPs can exchange message units, the TPs must be brought into execution.

Invoking a Remote Transaction Program

Assume that a source TP is already in execution. It requests invocation of a remote TP by issuing the ALLOCATE verb (or MC_ALLOCATE, which PS.MC converts into an ALLOCATE). It identifies the program to be invoked by specifying the remote transaction program name and remote LU name, and selects the desired transport characteristics by specifying a mode name.

Using the parameters from ALLOCATE, the source PS builds an Attach FM header and sends it to HS for transmission to the partner LU. When the target HS receives the Attach FM header, it passes it to its RM. This RM checks some parameters in the Attach FM header, including any security parameters in the Attach. If a format or protocol error is found, the Attach FM header is rejected by terminating the session that it arrived on. If no format or protocol error is found but the Attach contained invalid or inadequate information, RM sets a sense data field, creates a PS process and passes it the Attach FM header with the sense data. Upon finding the sense data, the new PS builds and sends an FM Header 7 containing the sense data, thus rejecting the Attach. If RM finds no errors, it creates a PS process and passes it the Attach FM header with no sense data. The new PS analyzes the Attach FM header further and, if an error is detected, rejects it; otherwise, PS selects and loads the specified transaction program code, and calls it, placing it initially in receive state for the conversation.

Once a target TP is invoked, it can act in turn as a source TP to invoke other TPs. If conversation-level security is required by the other TPs, the same security user ID that initiated the original target TP may be used, along with an Already Verified indicator in the Attach FM header, or the source TP may supply the required security parameters.

Initiating the Initial Local Transaction Program

The first TP activated for a distributed transaction is initiated by a START_TP record received by RM from an initiator process on the same system. Examples of an initiator process are an application, the node operator facility (NOF), a TP-PS process, a control-point process, or RM itself. The START_TP record contains information such as the name of the TP to be started; security tokens of the requester, e.g., user ID, password, profile; an indication as to whether a reply to the START_TP request is desired, and the initiating process's name and ID.

RM treats the START_TP much like an Attach: the requested TP-PS process is created and initialized; however, no conversation is associated with a START_TP request.

Terminating a Transaction Program

A TP ends by returning to PS.INITIALIZE. PS then performs any necessary final processing (such as deallocating the TP's remaining conversations), and notifies RM. If no queued START_TP or Attach requests exist for the TP, RM destroys the PS process.

CONVERSATION ALLOCATION AND DEALLOCATION

A source TP initiates a conversation with a target TP by issuing the ALLOCATE (or MC_ALLOCATE) verb.

The source PS satisfies the TP request in two steps.

First, PS sends RM a request to allocate a conversation. RM creates a conversation resource and notifies PS.

Second, PS sends RM a request to assign a session to the conversation. When RM has a session available for the conversation, RM connects the PS process of the issuing TP to the HS process of the session and notifies PS and HS. PS places the source end of the conversation (where the allocation was requested) initially in send state.

If a session is not immediately available, RM suspends the issuing process.

After a session is assigned to the conversation at the source LU, PS sends the Attach FM header to HS for transmission to the target LU.

When HS at the target LU receives the first BIU of the bracket, it notifies RM. RM receives the Attach from HS, creates the conversation resource, and makes it accessible to HS and PS. It places the target end of the conversation initially in receive state.

The following sections give further details of these functions.

Selecting a Session

RM maintains a list of allocation requests and a list of free sessions and their contention polarities. If RM has an allocation request (i.e., from an ALLOCATE(RETURN_CONTROL = WHEN_SESSION_ALLOCATED)) and a first-speaker (contention-winner) session is free (i.e., in between-brackets state), RM allocates that session to the conversation. If a first-speaker session is not free but a bidder (contention-loser) session is free, RM bids for the session. If no sessions are free, but the session limits have not been reached, RM requests that SM activate a new session. If no sessions are free and the session limits are reached, RM queues the allocation request to await the freeing of a session.

Bidding

RM requests HS to attempt to begin a bracket by sending an RU with BB; this is called bidding for the session.

RM always accepts a bid received on a bidder session.

If RM receives a bid on a first-speaker session, RM accepts or rejects the bid depending on whether any of its own transactions need to allocate the session for use by their own conversation (if they do, then it sends a negative response to the bid; otherwise, it sends a positive response to the bid).

Optionally, a negatively-responding RM will inform the partner when it is again willing to accept a bid.

Newly Active Session

When a session becomes newly active, it is initially in in-brackets state. If LU-LU verification is active, RM at the primary LU creates and sends (via HS) a Security FM header (FMH-12) to the secondary LU's RM for verification. The LU that activated the session (the primary LU, or BIND sender) has first right to send, regardless of the session contention polarity. If RM at the primary LU has no unsatisfied conversation request when a session becomes active, it requests HS to yield the session, i.e., to end the bracket.

Deallocation

When PS requests deallocation of the conversation, HS ends the current bracket, and RM deletes the conversation resource and places the session in the free-session list.

SESSION ACTIVATION AND DEACTIVATION

If RM has a conversation request for a session but no session is free and the session limits have not been exceeded, RM requests SM to activate a new session. RM also requests session activation as a result of operator commands (such as INITIALIZE_SESSION_LIMIT).

Starting a Session

Starting a session involves the following three activity phases: session limits initialization, session initiation, and session activation.

Initializing Session Limits: Prior to any transaction activity, the control operator sets limits on the maximum and minimum number, and contention polarity, of active sessions with particular partner LUs using particular mode names (see "Control-Operator Functions" on page 2-36 for details).

Session Initiation: When SM receives a session activation request from RM, SM sends an ASSIGN_PCID record to the session services (SS) component of the CP. SS responds by sending to SM an ASSIGN_PCID_RSP record containing the fully-qualified procedure correlator ID that uniquely identifies the potential session and the procedures related to that session.

SM then sends an INIT_SIGNAL record to SS, which directs the control point to mediate the initiation of the session. SS sends to SM a CINIT_SIGNAL record containing session characteristics and information to be included in the BIND.

SM then sends an ASSIGN_LFSID record to the address space manager (ASM) component of the CP. ASM responds with an ASSIGN_LFSID_RSP, which contains the local-form session identifier (LFSID) for the potential session. Refer to "Chapter 4. LU Session Manager" for more details of session initiation.

Session Activation SM then generates a BIND RU containing the desired session parameters. If security is used, the session parameters include randomly generated data for LU-LU verification and an indication of the amount of conversation-level security support that is defined for the secondary LU. Random and enciphered data are sent/received only when LU-LU verification is active. SM sends the BIND to its local CP for routing to the partner LU.

SM for the LU receiving the BIND (the secondary LU or SLU) negotiates the proposed session parameters to acceptable values; enciphers the received random data based upon the LU-LU password; saves the indication of the primary LU's conversation-level security support for the secondary LU; and creates a positive response to BIND. The positive response to BIND includes an indication of the secondary LU's conversation-level security support for the primary LU, randomly generated data, and the enciphered version of the random data received in BIND. SM sends this positive response to BIND via its local CP.

When the positive response to BIND is sent or received, the session manager at each end connects a new HS process to the path control network. If the session uses cryptography, the HSs exchange cryptography-verification

RUs. Then, each SM notifies its RM that a new session is available. If LU-LU verification is active, before the new session is available for conversations, the primary LU's RM enciphers the random data received on the response to BIND and returns it to the secondary LU's RM for verification.

If the LUs cannot agree on session parameters, or the enciphered random data comparison fails, the session activation fails.

Session Outage

If session outage occurs, SM notifies RM. If a conversation was active on the session, RM notifies PS, which notifies the transaction program of conversation failure. RM requests SM to activate another session if it has unsatisfied conversation requests or an unsatisfied auto-activation limit.

Ending a Session

Ending a session involves the following three activity phases: operator request, session shutdown, and session deactivation.

Operator Request: Sessions are not deactivated in the normal course of transaction program processing; they are deactivated normally only upon specific request from the control-operator transaction program. (Sessions are deactivated abnormally because of protocol violations and physical connectivity problems.)

When the LU operator at either end of a session determines that a session is to be deactivated, the control-operator transaction program issues a control-operator verb. The control operator can cause sessions to end in two ways.

The operator can issue a RESET_SESSION_LIMIT verb to reset the session limits to 0 for specified partner LUs and mode names. The LU proceeds with subsequent phases until there are no active sessions for the specified (LU,mode) pairs.

The operator can also issue a DEACTIVATE_SESSION verb to deactivate a specific session (this might be done, for example, to recover from certain error situations). This does not change the session limits, however, so the LU might activate another session to replace it.

When PS.COPR receives the verb, it issues a session-limit-change notification or a session-deactivation request to RM.

Session Shutdown: When RM receives a session-limit-change notification, RM first performs drain processing. If the operator has requested RESET_SESSION_LIMIT with drain indicated, then RM performs no deactivations until all requests for allocation of sessions with the specified mode name have been satisfied.

When drain is complete, or when RM receives a session-deactivation request, and an affected session next enters between-brackets state, RM initiates a bracket-termination protocol. This consists of an exchange of bracket-initiation-stopped (BIS) RUs assuring that all brackets have completed at both ends of the session, i.e., that no other BIUs are in transit between the LUs.

After receiving BIS, the partner LU drains its allocation requests and sends BIS in return.

When the BIS protocol is complete, the RM that initiated the BIS protocol instructs its SM to deactivate the session.

Session Deactivation: When SM receives a session-deactivation request from RM, it sends UNBIND, via the local CP, and awaits a response. When the partner SM receives an UNBIND, it unconditionally sends a positive response. When the response to UNBIND is sent or received, the corresponding SM disconnects the half-session process from the path control network, notifies the CP that the session is ended, and destroys the half-session process.

FUNCTIONAL SUMMARY BY COMPONENT

This section is organized by component; it reviews the specific functions of each principal component, and describes functions performed primarily in one component.

Presentation Services

PS manages transaction programs and controls conversation-level communication between TPs:

- Loads and calls the transaction program
- Maintains the conversation protocol state, e.g., send/receive state of the TP
- Enforces correct verb parameter usage and sequencing constraints
- Coordinates specific processing for each verb
- Performs mapping of transaction program data into mapped-conversation records

- Converts mapped-conversation records to GDS variables, and the reverse: it partitions the data into logical records and generates LLID prefixes

- Blocks data into RU-sized message units (MUs)

- Reblocks MU data from HS into logical records or buffer records as required by the TP

- Verifies logical-record length and boundaries

- Truncates or purges data when errors are reported or detected by the TP

- Generates and issues FM headers for Attaches and Error-descriptions

Half-Session

HS controls session-level communication between LUs:

- Builds RHs and enforces correct RH parameter settings

- Creates chains and enforces chaining as the unit of LU-to-LU error recovery

- Correlates responses with the correct bracket

- Enforces bracket protocol and purges rejected brackets

- Enforces protocols for the relevant FM and TS profiles for the session (FM profile 19 and TS profile 7)

- Generates and enforces sequence numbering to detect lost or duplicate BIUs

- Provides session-level pacing (none, fixed, or adaptive)

- Exchanges cryptography-verification RUs when session cryptography is being used

- Enciphers and deciphers data when session cryptography is being used

Resources Manager

RM manages presentation services and conversations

- Creates and destroys instances of presentation services

- Creates and destroys conversation resources and connects them to half-sessions and to presentation services

- Finishes LU-LU verification for session-level security by generating and processing Security FM headers (FMH-12s)

- Performs all conversation-level security checks, verifies conversation-level passwords, and controls access to protected transaction programs

- Maintains the data structures representing the dynamic relationships among conversation resources, half-sessions, transaction program instances, and transaction program code

- Chooses the session to be used by a conversation and controls contention for the session

- Performs drain action: allows session traffic to cease before requesting session deactivation

- Requests SM to activate and deactivate sessions

Session Manager

SM manages sessions:

- Coordinates session initiation in concert with the control point

- Sends and receives BIND

- Supplies and negotiates session parameters during BIND exchange

- Exchanges cryptographic key and session seed

- Exchanges random and enciphered data and performs initial LU-LU verification

- Notifies RM of session outage

- Creates and destroys half-session instances and connects them to path control instances

FUNCTIONS OF COMPONENTS OF THE NODE EXTERNAL TO THE LU

Buffer Manager: The primary objective of the node buffer manager is to manage buffer pools. The LU uses these facilities for session-level pacing. The facilities provided by the node buffer manager are:

- A mechanism to increase and decrease buffer resources used by a process based on fair sharing of limited storage
- A mechanism that notifies buffer users when buffer resources are in critically short supply
- A mechanism that allows processes to wait for buffer resources to become available

Type 2.1 Node Control Point (T2.1 CP): The T2.1 node control point allows peer-to-peer connection of distributed processors by assisting in the activation of links and sessions, e.g., it locates partner LUs, sets the path, assigns LFSIDs. For more information on T2.1 node control points refer to SNA Type 2.1 Node Reference.

Node Operator Facility (NOF): NOF manages the activation and deactivation of the LU.

Initiator Process: An initiator process is any process in the LU's local system that has addressability to the RM component of the LU. The initiator process is considered privileged in that it may use this addressability to send records to the LU that cause the LU to perform specific functions, e.g., to create a transaction program that initiates a distributed transaction.

FUNCTIONS OF SERVICE TRANSACTION PROGRAMS

Service transaction programs provide functions to the end user that require communication with another LU using a special SNA-defined pattern of verbs.

Service TPs form part of a distributed transaction similarly to other TPs. They have a transaction program name and are invoked by the Attach mechanism, and they exchange information with these other TPs by issuing transaction-program verbs.

Service transaction programs differ from user-application transaction programs in that they are SNA-defined and are considered part of the LU. The names of service transaction programs are SNA-defined. The records that service TPs send and receive are SNA-defined GDS variables.

Control-Operator Functions

All LUs have an implementation- or installation-defined control operator transaction program (COPR TP) that represents the LU control operator's interface to the LU. Using a program-selected means such as operator console input, this TP issues control-operator verbs to perform control-operator functions.

Control-operator verb functions include creation and modification of the data structures that describe the LU and the LU-accessed network resources: control points, transaction programs, partner LUs, and modes. Other control-operator verb functions limit the numbers and contention polarities of sessions with particular LUs for particular mode names, and also determine when sessions will be activated and deactivated.

For an LU that supports parallel sessions, there are additional transaction services components for the control operator. These LUs contain a change-number-of-sessions (CNOS) service transaction program. When processing CNOS verbs, the COPR TP at one LU exchanges GDS variables with the CNOS service TP at its partner to reach mutual agreement about limits on the number of parallel sessions between them.

(Control-operator functions are discussed in further detail in "Chapter 5.4. Presentation Services--Control-Operator Verbs" .)

SNA Distribution Services

SNA Distribution Services (SNA/DS) provides a set of verbs that an application TP may issue to request asynchronous distribution of data.

The service is provided by a network of distribution service units (DSUs) interconnected by conversations and sessions. Each DSU consists of PS verb handlers and a collection of service TPs within the LU. The service TPs provide data storage, routing, and distribution asynchronously with the origin or destination application programs.

SNA/DS is described in the publication SNA Format and Protocol Reference Manual: Distribution Services.

Document Interchange Services

Document Interchange Architecture (DIA) describes formats and protocols for synchronous exchange of documents by using basic-conversation verbs in a prescribed way. Document interchange services include service TPs for synchronous document transfer.

Document interchange architecture is described in the publication Document Interchange Architecture--Concepts and Structures.

OPTIONAL FUNCTIONS

This section describes the principal optional function sets.

Mapping Function

The mapping function is an optional function of mapped conversations (PS.MC) that allows a TP to select transformations, called maps, to be applied to TP data at the sending and receiving TP protocol boundaries. Maps are non-SNA-defined transformation tables or procedures that can be defined by the installation at both the source and target LUs. Maps can specify, for example, how fields of a mapped-conversation record are related to the TP variables (data record) referred to in protocol-boundary verbs.

Each LU can support multiple maps. Each map is identified by a map name. The maps to be applied are selected by the transaction program (via verb parameters) and by other maps (in an implementation-defined way), as shown in Figure 2-27 on page 2-37.

Three separate map-name name spaces exist (terms in parentheses correspond to those in the figure):

```
              __                                        __
             *  *                                      *  *
            *    *                                    *    *
            |*__*|                                    |*__*|
            |    | Sender map (map-1)                 |    | Receiver map (map-2)
             *__*                                      *__*
              |                                         |
              |                                         |
              V                                         V
           ________                                  ________
source TP sends: |        |  transferred on conversation:  |        |  target TP receives:
                 |        |                                |        |
map-name-1, data-1 |      |  map-name-2, data-2            |        |  map-name-3, data-3
------------------>|      |------------------------------->|        |------------------->
                 |        |                                |        |
                 | Send   |                                | Receive|
                 | Mapping|                                | Mapping|
                 |________|                                |________|
```

Figure 2-27. Map Name Usage by Mapped Conversations

1. Sender locally-known map name: This map name (map-name-1) is known to the TPs at the sending LU. It identifies a map (map-1) at the sending LU that defines the transformation performed by the sender from the format of the sending-program data (data-1) to the format of the MCR (data-2) that is sent on the conversation. This map also defines a correspondence between the sender locally-known map name (map-name-1) and the globally-known map name (map-name-2) described below.

2. Globally-known map name: This map name (map-name-2) is known at both the sending and receiving LUs, and is transferred on the conversation between sender and receiver. It identifies a map (map-2) at the receiving LU. This map defines the transformation performed by the receiver from the format of the MCR received on the conversation (data-2) to the format of the data presented to the receiving transaction program (data-3). This map also defines a correspondence between the globally-known map name (map-name-2) and the receiver locally-known map name (map-name-3) described below.

3. Receiver locally-known map name: This map name (map-name-3) is known to TPs at the receiving LU. This identifies the format of the data presented to the program (data-3), e.g., it allows the program to select the correct structure definition or format description for the data produced by the execution of the receiver map (map-2).

Mapping is performed by a PS.MC component called the mapper.

The mapper at the sender selects the send map specified by the sender locally-known map name, which is supplied as a parameter of the MC_SEND_DATA verb. It performs the send mapping on the TP-supplied data, producing a mapped-conversation record. Using the sender map, the mapper also selects the globally-known map name.

The LU sends the globally-known map name over the conversation in an SNA-defined map-name GDS variable (see SNA Formats), and sends the mapped-conversation record in a separate GDS variable.

The mapper at the receiver selects the receive map specified by the globally-known map name received. It performs the receive mapping on the mapped-conversation record it receives, resulting in data formatted for presentation to the TP. Using the receiver map, the mapper also selects the receiver locally-known map name. PS.MC passes the receiver locally-known map name and the reformatted data to the TP as returned parameter values for the next receive verb issued, e.g., MC_RECEIVE_AND_WAIT.

The receiving TP uses the receiver locally-known map name in a TP-determined way to interpret the received data.

The TPs supply or receive a map name parameter value for each send or receive verb issued, respectively. The LU, however, does not send another map-name GDS variable if the globally-known map name has not changed from that of the previous record sent. To accomplish this, the mapper at each LU retains the most recently sent and most recently received values of map-name-2 for the conversation (the send and receive map names can be different). The retained values for each direction persist until changed or until the end of the conversation, regardless of intervening turnarounds.

Sync Point Function

The sync point function allows all TPs processing a distributed transaction to coordinate error recovery and maintain consistency among distributed resources such as data bases.

The sync point functions affect protected resources. These include conversation resources and implementation- or installation-designated resources such as data bases. Any changes to a protected resource are logged so that they can be either backed out (reversed) if the transaction detects an error, or committed (made permanent) if the transaction is successful.

The transaction programs divide the distributed transaction into discrete, synchronized logical units of work (LUWs), delimited by synchronization points (sync points). (Corresponding sync points occur at each TP participating in the distributed transaction.) LUWs are sequences of operations that are indivisible units for the application, i.e., any failure in an LUW invalidates the entire LUW (all LUW processing by all TPs for the transaction), so the transaction is backed out to the previous sync point.

The LU components for the sync point function are shown in Figure 2-28 on page 2-39.

Highlights of the sync point function are discussed below. (See "Chapter 5.3. Presentation Services--Sync Point Services Verbs" for details.)

Sync Point Control: The sync point function at each LU is coordinated by PS.SPS.

For each TP process participating in the distributed logical unit of work, the corresponding PS.SPS tracks the state of that logical unit of work. To do this, PS.SPS has protocol boundaries with the TP and with the protection managers for each conversation and for each protected local resource allocated to that TP.

Logging: When processing a given logical unit of work, whenever a TP issues a verb that makes any changes to a protected resource, the corresponding resource protection manager logs the change so that, if necessary, the change can be backed out later.

The log manager maintains the log entries for each active LUW (i.e., for each active transaction) on non-volatile storage, using implementation-defined data-management functions. The same log is used to record all log entries for all the LU resources for the LUW.

Resources Manager: When it creates the PS process, RM provides PS.SPS with access to the log.

In some cases, a transaction program can terminate normally before its sync point log entries are erased. In these cases, RM assumes the function of the terminated sync point control to complete the protocol and to release (forget) the log entries.

Protection Managers: Each protected resource, e.g., a conversation or a local data base, has a protection manager that logs significant state changes during a logical unit of work, detects errors affecting the integrity of the changes, and commits or backs out the changes as determined by the sync point protocol.

The protection manager for a conversation is defined by SNA; protection managers for other (non-SNA) resources are defined by the implementation, but have a similar protocol boundary to PS.SPS. The protection managers form a sublayer between PS verb handlers and the resource-control components.

Sync Point Protocol: At the end of a logical unit of work, an application-designated TP initiates sync point. The LUs then carry out a protocol involving all local protected resources and conversations being used by the TP, and all partner LUs and TPs directly connected by those conversations, to determine whether any TP or protected resource detected an error in the LUW, and to propagate this result to the other LUs and TPs.

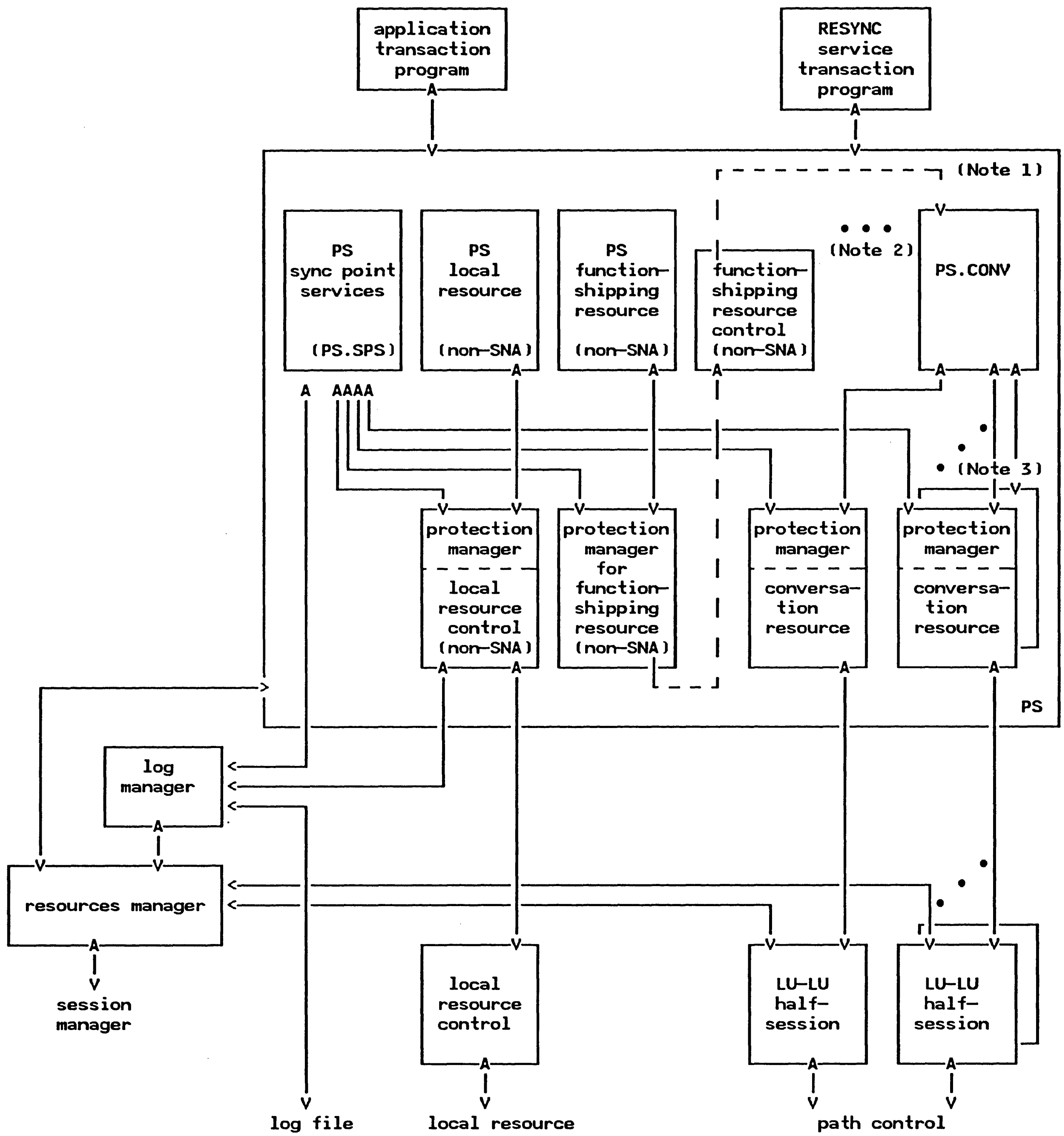
When a TP issues a verb that invokes the sync point function (e.g., SYNCPT, BACKOUT) its PS.SPS coordinates the sync point protocol. PS.SPS exchanges sync point commands, in the form of presentation services (PS) headers and FM headers, over the TP's conversations with other TPs. Each PS.SPS component for the transaction performs similar exchanges, in turn, with its TP's conversation partners. The PS.SPS components also determine the status of local non-SNA resources by exchanging appropriate commands across their internal protocol boundaries. These exchanges direct the protection managers to complete any pending log entries for the LUW.

The sync point protocol culminates with a mutual decision among all TPs processing the LUW either to commit or to back out the LUW.

Commitment and Back-Out: When the sync point protocol is complete at a particular TP, the resource control components use the LUW log entries to supply the information needed (e.g., data base change records) to perform the required commitment or back out. They then notify PS.SPS to erase the log entries for that LUW.

Resynchronization: An LU failure might occur during the sync point protocol, so that some LU never receives an expected LUW status report. To recover from this case, the other LUs can wait until the failing LU is reinitialized, and then the LUs perform a resynchronization (resync) protocol to complete the sync point processing at each LU. Resync uses service transaction programs to exchange sync point status among the LUs.

When the failing LU is reactivated, the LU completes the resync transaction before running any other transaction programs that require sync point. The resync service TP is initiated by RM at some LU, typically at the sync point initiator; this TP attaches the resync TP at its partners, which continue propagating the resync TP throughout the LUs that had been processing the distributed transaction.



NOTES:

1. Function-shipping resource control recursively calls PS to communicate with the partner. The conversation used for communication with the partner has its own protection manager.

2. PS components not relevant to sync point have been omitted from this figure.

3. A distinct protection manager exists for each conversation resource created by PS.

4. The non-SNA components are undefined protocol machines (UPMs).

Figure 2-28. Relationship of LU Components for Sync Point Functions

The first step of the resync transaction is to validate the integrity of the LU logs, i.e., to determine that all LUs' logs contain consistent entries for the same LUW. To do this, the resync service TPs exchange Exchange Log Name GDS variables on the conversation. Next, the service TPs exchange Compare States GDS variables to determine the status of the sync point protocol at the time of failure. PS.SPS then uses this information to complete the sync point protocol. (See SNA Formats for the SNA-defined format of the Exchange Log Name and Compare States GDS variables.)

DATA STRUCTURES

The LU maintains data structures representing the state and configuration of its resources.

Some system-definition data structure elements represent the LU-accessed network resources. These structures describe the characteristics of the LU itself, the transaction programs that the LU can run, the partner LUs with which the LU can communicate, and the modes characterizing possible sessions with particular partner LUs.

Other data structure elements represent the dynamic environment created by the LU. The principal components of this environment are the transaction program instances in execution (represented by transaction-program processes) the active sessions with other LUs (represented by half-session processes), and the active conversations (represented by conversation resources). This environment also includes the relationships of the dynamic components to the LU-accessed network resources and to each other.

LU-ACCESSED NETWORK RESOURCES

Figure 2-29 on page 2-41 illustrates the data structures that represent the LU-accessed network resources.

The LUCB structure (and some associated lists not shown) describe the local LU. This information includes the LU's fully qualified name and the set of optional functions (e.g., parallel sessions and mapping) that the LU supports. The LUCB is also the anchor for lists of data structures describing the other LU resources.

A TRANSACTION_PROGRAM structure (and associated lists not shown) describe the transaction programs at the local LU. This information includes the transaction program name, its current availability status, and the set of optional functions (e.g., sync point, mapping, and access control) that it supports.

A PARTNER_LU structure describes a remote LU (potential partner LU). This information includes the remote LU's names: local LU name, fully-qualified LU name, and uninterpreted LU name. It also includes the set of the LU's optional capabilities, such as parallel sessions and security. The PARTNER_LU structure also contains a list of mode descriptions.

A MODE structure describes a set of session characteristics that a group of sessions share. These characteristics include the name of the mode and the set of optional functions that are supported by the remote LU on a mode basis, e.g., sync point. It also includes the session parameters that characterize this mode, such as maximum allowed RU size, session-pacing window size, and session cryptography parameters. The MODE structure also indirectly describes link characteristics: the mode name is used by the control point as the key to tables identifying the links and routes to be used for sessions for that mode. Distinct partner LUs have distinct modes. The characteristics for sessions to different partner LUs may be different even if the sessions have the same mode name.

PROCESSES AND DYNAMIC RESOURCES

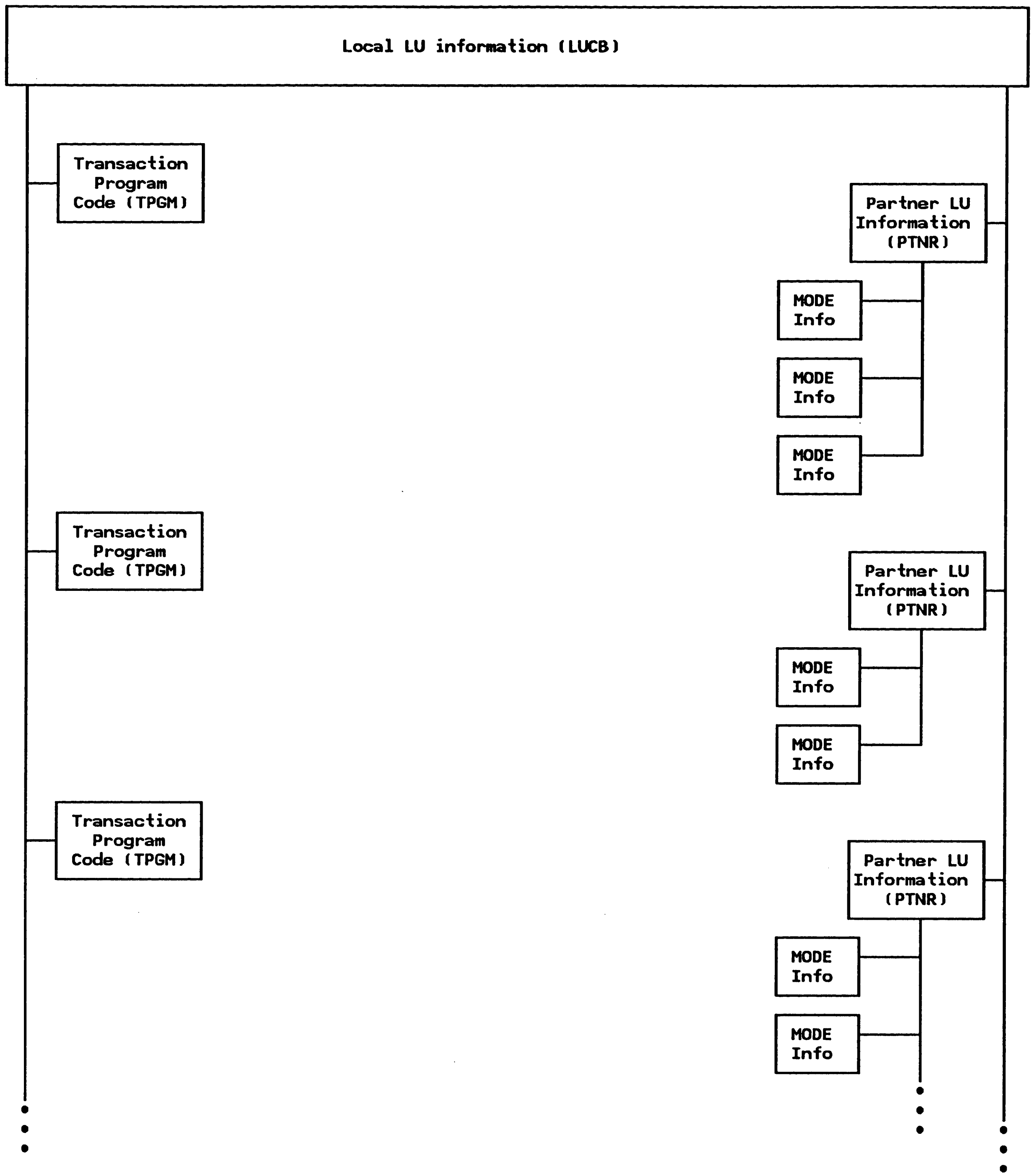
Figure 2-30 on page 2-42 illustrates the principal data structures and processes, and their relationships, that represent the dynamic environment. The formal description represents these relationships in various ways such as pointers between control blocks, keys of elements in lists, and intermediate dynamic control blocks.

The processes also contain state information used by LU functional components; this is described in more detail in chapters concerned with the relevant functional components.

The TP process represents a transaction program instance. It identifies the transaction program code that it is using. There may be multiple transaction program processes executing the same transaction program code.

The HS process represents a half-session. It identifies the remote LU and mode with which it is associated. A mode may be associated with many half-session processes, but each HS process is associated with only one mode.

The RCB structure represents a conversation resource. The RCBs are the central elements in the dynamic configuration of the LU: they represent the connection of a transaction program to a half-session; this connection is dynamically created and destroyed, and allows an asynchronous (Send/Receive) relationship between TP and HS. The RCB identifies the local TP using the conversation and the

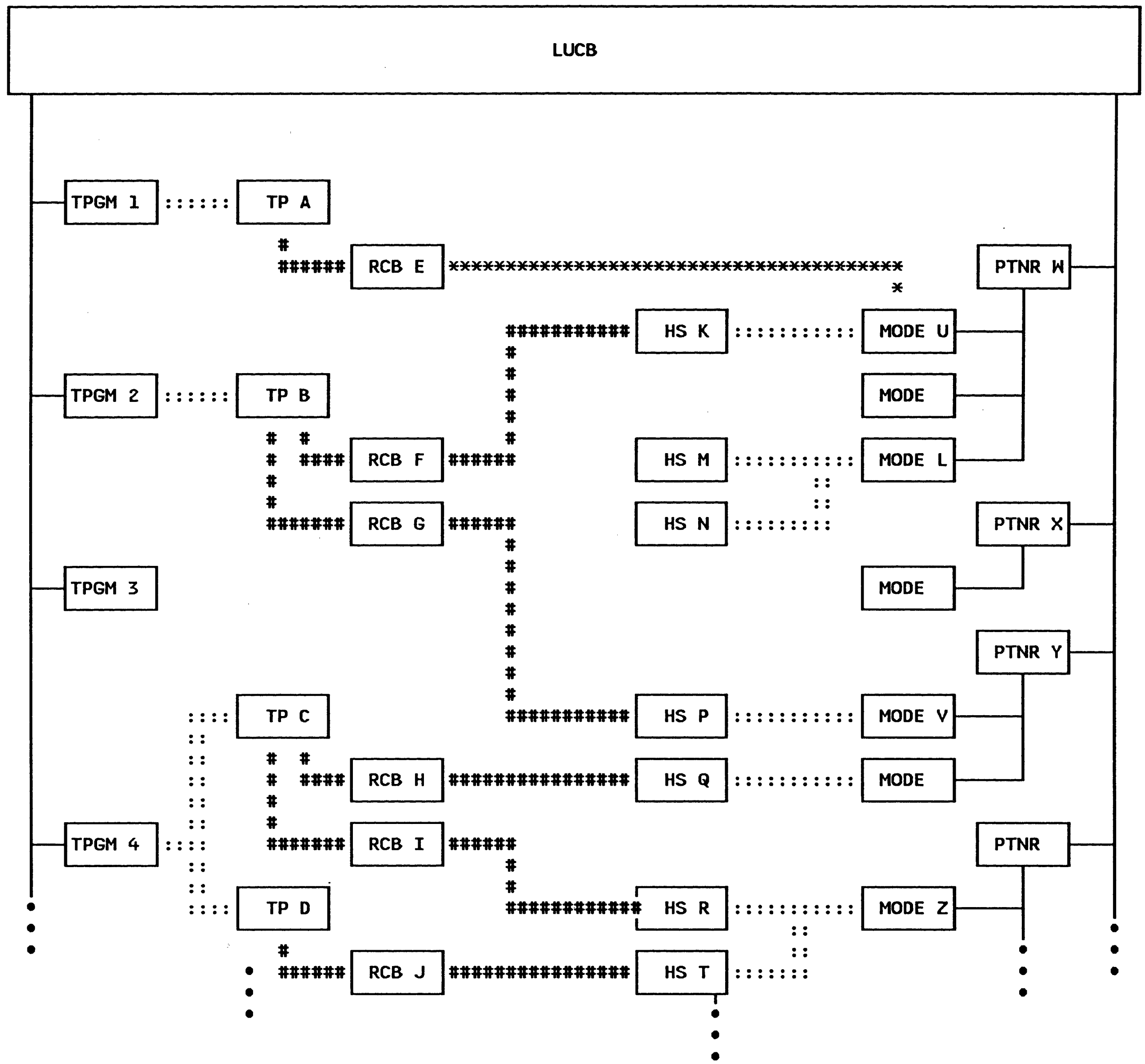


LEGEND:
Vertical lines represent lists of subordinate resources

<u>Associated Data Structure Name</u>

LUCB:	LUCB	PTNR:	PARTNER_LU
MODE:	MODE	TPGM:	TRANSACTION_PROGRAM

Figure 2-29. LU Static Data Structures (Example)



LEGEND:
Vertical lines represent lists of subordinate resources
:::: association of process to static data elements
association of processes via RCB dynamic data element
**** association of RCB with MODE in lieu of unavailable HS

Abbr.		Data Structure Name
LUCB:	Local LU information	LUCB
TPGM:	Transaction Program Code information	TRANSACTION_PROGRAM
PTNR:	Partner LU information	PARTNER_LU
MODE:	Mode information	MODE
TP:	Transaction program process	
RCB:	Conversation resource information	RCB
HS:	Half-session process	

Figure 2-30. LU Dynamic Data Structures and Processes (Example)

half-session being used, if any. Because a session might not be immediately available when a TP allocates a conversation, the RCB also identifies the remote LU (PARTNER_LU) and mode name (MODE) for the desired session. Many conversation resources, hence RCBs, may be associated with the same local TP, but each RCB may be associated with only one local TP, one partner LU, one mode, and one half-session.

Figure 2-30 on page 2-42 illustrates several of the possible relationships among these structures. In the figure:

- Active TP B for transaction program code 2 has two active conversations:
  - RCB F connects it to remote LU W via session K with mode name U.
  - RCB G connects it to remote LU Y via session P with mode name V.
- LU W has two free sessions, M and N, each with mode name L.
- Remote LU X has a single mode name with no active sessions.
- No active TP instances exists for transaction program 3.
- Two active TP instances exist for transaction program 4: TPs C and D.
- Two conversations G and H exist with remote LU Y, each using a different mode name.
- Two conversations I and J use separate sessions R and T, both with mode name Z.

RESOURCE RELATIONSHIPS IN A DISTRIBUTED TRANSACTION

In contrast to Figure 2-30, which illustrates the data structures for several transactions from the perspective of a single LU, Figure 2-31 on page 2-44 illustrates the relationships among data structures at several LUs from the perspective of a single distributed transaction. In this case, the paired half-sessions connect LUs, and the paired conversation resources, represented by RCBs, connect transaction program instances.

LU STARTUP AND SHUTDOWN

LU startup consists of three phases: creating the LU processes, initiating the control operator transaction program, and setting the LU definition and session limits. The LU then initiates programs and activates sessions in response to further operator, transaction program, or partner-LU actions.

To shut down the LU, the steps are reversed, but some can be omitted. The minimum required to terminate communication is to reset the session limits.

LU PROCESS CREATION AND TERMINATION

Figure 2-32 on page 2-45 shows the process creation and termination hierarchy for the LU. The node operator facility (NOF) creates the SM process. As part of SM's initial processing, it creates the RM process and then informs NOF of RM's successful creation. These processes continue running thereafter.

The TP and HS processes are discussed in "Running State" on page 2-44.

CONTROL-OPERATOR TRANSACTION PROGRAM INITIATION

As a result of receiving a START_TP record from NOF, RM creates a PS process and initiates the control-operator TP.

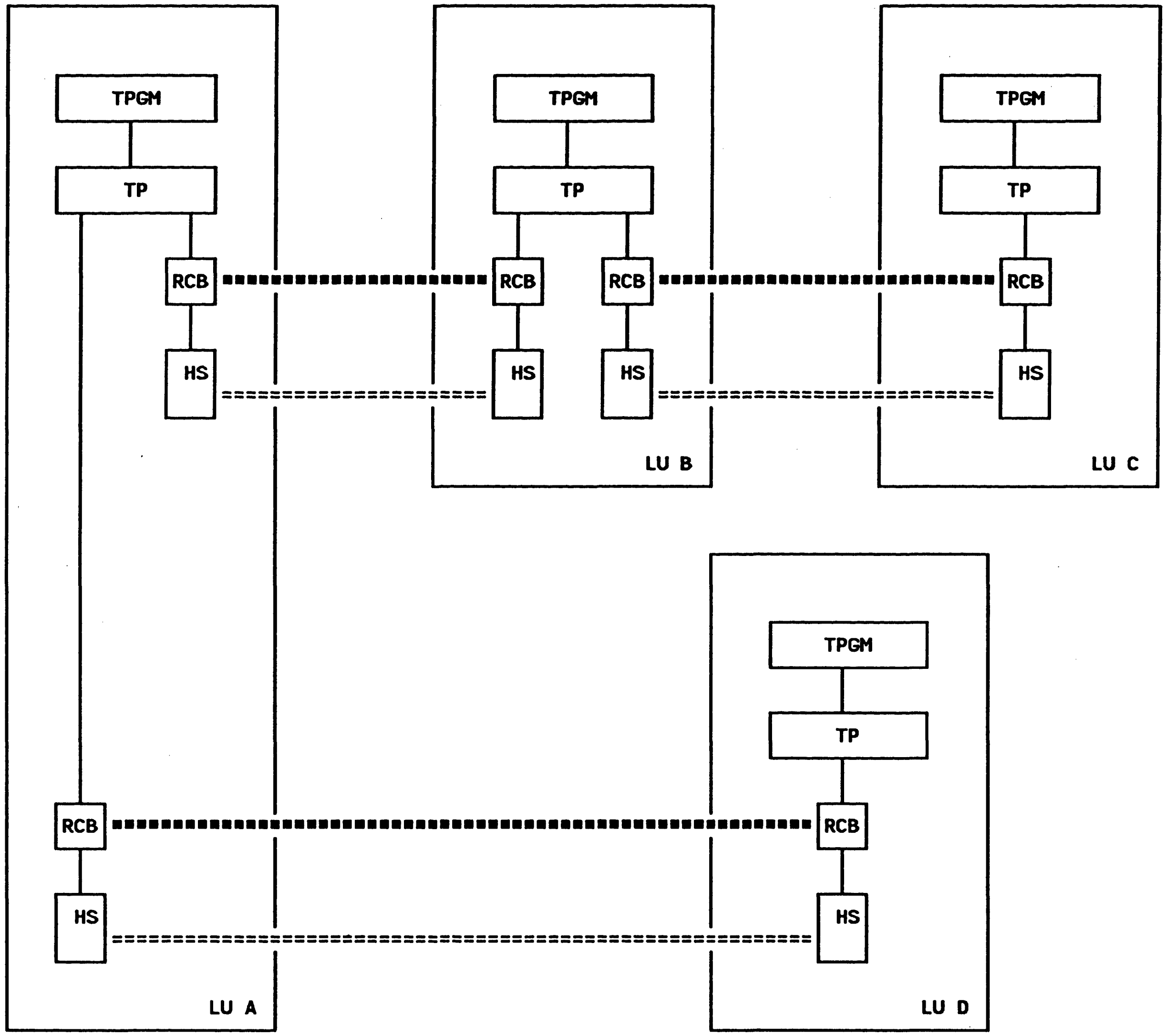
CONTROL-OPERATOR ACTIONS

The control operator specifies the LU definition describing the LU-accessed network resources: the transaction programs, partner LUs, and modes. (An implementation might provide this function without requiring explicit operator interaction, e.g., the LU definition might be specified at system-definition time.)

The operator initializes session limits with the partner LUs by issuing the INITIALIZE_SESSION_LIMIT verb for the relevant mode names. For parallel-session mode names, this verb activates an LU-LU session using the SNA-defined mode name SNASVCMG (if not already active) and establishes mutually agreeable session limits for other mode names by exchanging CNOS GDS variables on that session. This verb optionally causes activation of a predetermined number of sessions for the specified mode name.

When sessions are to be deactivated, the control operator issues RESET_SESSION_LIMIT for the mode name. For a parallel-session connection, this causes another CNOS GDS variable exchange to elicit the partner LU's cooperation in the session shutdown. In any case, this verb causes the LU to eventually cease initiating new transaction programs and activating new sessions (drain). As sessions become unused, RM and SM deactivate them.

The LU initiates no further actions to shut down the LU. Any further actions are at the initiative of NOF.



LEGEND:
——— Association of a process with its data structures
■■■■■■ Conversation (connection between transaction program instances [TPs])
====== Session (connection between LUs)
TPGM: Transaction program data structure (represents transaction program code)
RCB: Resource control block (represents a conversation)
TP: Transaction program process instance
HS: Half-session process instance

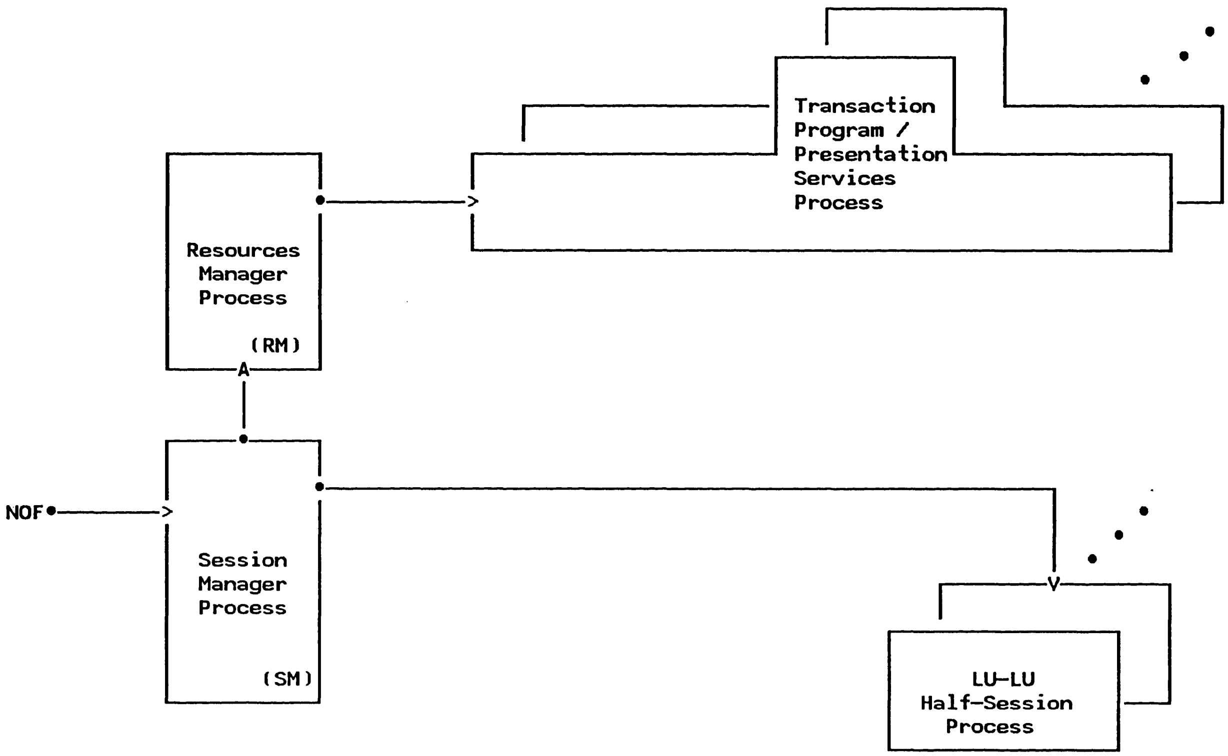
Figure 2-31. Data Structure Relationships among LUs for a Distributed Transaction (Example)

RUNNING STATE

Once the LU-LU session limits have been set, the LU is ready to process transactions.

RM creates a transaction-program process when it receives an Attach or an initial TP invocation request (START_TP); it destroys that process when PS indicates that the TP has completed and all its conversations have been deallocated.

Either RM or the partner LU can request session activation; in either case, SM performs the relevant processing. SM creates an HS process for an LU-LU session and connects it to a path control instance whenever it sends or receives BIND. SM destroys that process when it has sent or received a positive response to UNBIND, has disconnected the half-session from path control (by sending PC_HS_DISCONNECT, RSP(UNBIND), or UNBIND to the CP), and has notified the CP that the session is ended (by sending SESSEND_SIGNAL).



LEGEND:
•——> process creation (The arrow points from creator to created.)
NOF: Node operator facility

Figure 2-32. LU Process Creation and Termination Hierarchy

EXAMPLE

Figure 2-35 on page 2-52 and Figure 2-36 on page 2-53 illustrate typical interactions at the local and remote LUs, respectively, for an LU shutdown sequence. "Chapter 5.4. Presentation Services--Control-Operator Verbs" describes LU startup and shutdown in more detail.

This section lists the external message units and internal records exchanged by LU components. See "Appendix A. Node Data Structures" for full descriptions of these structures.

TRANSACTION PROGRAM VERBS AND INTERPROCESS SIGNALS

PS-TP Protocol Boundary: Transaction Program Verbs

Basic-Conversation Verbs

ALLOCATE
CONFIRM
CONFIRMED
DEALLOCATE
FLUSH
GET_ATTRIBUTES
POST_ON_RECEIPT
PREPARE_TO_RECEIVE
RECEIVE_AND_WAIT
RECEIVE_IMMEDIATE
REQUEST_TO_SEND
SEND_DATA
SEND_ERROR
TEST

Mapped-Conversation Verbs

MC_ALLOCATE
MC_CONFIRM
MC_CONFIRMED
MC_DEALLOCATE
MC_FLUSH
MC_GET_ATTRIBUTES
MC_POST_ON_RECEIPT
MC_PREPARE_TO_RECEIVE
MC_RECEIVE_AND_WAIT
MC_RECEIVE_IMMEDIATE
MC_REQUEST_TO_SEND
MC_SEND_DATA
MC_SEND_ERROR
MC_TEST

Type-Independent Verbs

BACKOUT
GET_TP_PROPERTIES
GET_TYPE
SYNCPT
WAIT

Control-Operator Verbs

ACTIVATE_SESSION
CHANGE_SESSION_LIMIT
DEACTIVATE_SESSION
INITIALIZE_SESSION_LIMIT
PROCESS_SESSION_LIMIT
RESET_SESSION_LIMIT

INTERCOMPONENT STRUCTURES

SM-CP Protocol Boundary

SM to CP Interprocess Signals

- ASSIGN_LFSID
- ASSIGN_PCID
- FREE_LFSID
- INIT_SIGNAL
- LFSID_IN_USE_RSP
- MU (contains the following RUs)
  - BIND
  - UNBIND
  - RSP(BIND)
  - RSP(UNBIND)
- PC_HS_DISCONNECT
- SESSEND_SIGNAL
- SESSST_SIGNAL

CP to SM Interprocess Signals

- ASSIGN_LFSID_RSP
- ASSIGN_PCID_RSP
- CINIT_SIGNAL
- INIT_SIGNAL_NEG_RSP
- LFSID_IN_USE
- MU (contains the following RUs)
  - BIND
  - UNBIND
  - RSP(BIND)
- SESSION_ROUTE_INOP

SM-HS Protocol Boundary

SM to HS Interprocess Signals

- INIT_HS

HS to SM Interprocess Signals

- ABEND_NOTIFICATION
- ABORT_HS
- INIT_HS_RSP

SM-NOF Protocol Boundary

SM to NOF Interprocess Signal

- RM_CREATED

SM-BM Protocol Boundary

SM-BM Calls[3]

[3] Each buffer manager protocol boundary (here and following) is a synchronous (calling) invocation of the buffer manager by the components of the LU; the names in the list refer to request identifiers modeled as parameters in the Call.

ADJUST_POOL
FREE_BUFFER
GET_BUFFER
RESERVE_BUFFER

HS-PC Protocol Boundary

HS to PC Interprocess Signal

MU(outgoing data)

PC to HS Interprocess Signal

MU(incoming data)

HS-BM Protocol Boundary

HS-BM Calls

ADJUST_POOL
FREE_BUFFER
GET_BUFFER
TRANSFER_BUFFER

PS-HS Protocol Boundary

PS to HS Interprocess Signals

CONFIRMED
REQUEST_TO_SEND
MU(SEND_DATA_RECORD)
SEND_ERROR

HS to PS Interprocess Signals

CONFIRMED
MU(incoming data)
RECEIVE_ERROR
REQUEST_TO_SEND
RSP_TO_REQUEST_TO_SEND

PS-RM Protocol Boundary

PS to RM Interprocess Signals

ABEND_NOTIFICATION
ALLOCATE_RCB
CHANGE_SESSIONS
DEALLOCATE_RCB
GET_SESSION
RM_ACTIVATE_SESSION
RM_DEACTIVATE_SESSION
TERMINATE_PS
UNBIND_PROTOCOL_ERROR

RM to PS Interprocess Signals

MU(FMH-5)
CONVERSATION_FAILURE
RCB_ALLOCATED
RCB_DEALLOCATED
RM_SESSION_ACTIVATED
SESSION_ALLOCATED
START_TP

PS-BM Protocol Boundary

PS-BM Calls

FREE_BUFFER
GET_BUFFER

RM-HS Protocol Boundary

RM to HS Interprocess Signals

BID_RSP
BID_WITHOUT_ATTACH
BIS_REPLY
BIS_RQ
BRACKET_FREED
ENCIPHERED_RD2
HS_PS_CONNECTED
RM_HS_CONNECTED
RTR_RQ
RTR_RSP
YIELD_SESSION

HS to RM Interprocess Signals

BID
BID_RSP
BIS_RQ
BIS_REPLY
FREE_SESSION
MU(FMH-5 or FMH-12)
RTR_RQ
RTR_RSP

RM-SM Protocol Boundary

RM to SM Interprocess Signals

ABEND_NOTIFICATION
ACTIVATE_SESSION
DEACTIVATE_SESSION

SM to RM Interprocess Signals

ACTIVATE_SESSION_RSP
SESSION_ACTIVATED
SESSION_DEACTIVATED

RM-Initiator Process Protocol Boundary

RM to Initiator Process Interprocess Signal

START_TP_REPLY

Initiator Process to RM Interprocess Signals

SEND_RTR
START_TP

RM-BM Protocol Boundary

RM-BM Calls

FREE_BUFFER

The following figures illustrate both the internal protocol-boundary sequence flows among LU components and the external flows between two LUs that result from basic-conversation verb issuances.

Each flow is illustrated by a pair of figures on facing pages. Each separate figure represents the complete flow as seen by a single LU. The figure labeled local LU represents the LU that initiates the sequence being illustrated; the figure labeled remote LU represents the partner LU. For cases illustrating a race between two LUs, the LUs are distinguished as first speaker (FSP) and bidder. The flows through the path control network are shown in the column nearest the center margin, and are replicated in each figure; numerals in parentheses in the margins between facing parts of the same flow correlate corresponding steps in the facing figures. When flows cross in the path-control network, the crossing is illustrated on the sending side of the delayed flow.

NOTATION

For the interpretation of labels on the arrows, see the following (which, in some cases have been abbreviated):

- For verb and verb-parameter names (TP-PS), SNA Transaction Programmer's Reference Manual for LU Type 6.2
- For protocol-boundary records and message units (TP-PS, PS-RM, RM-SM), "Protocol Boundary Summary" on page 2-46
- For RU names (SM-SM, HS-HS), SNA Formats
- For RH indicators (SM-SM, HS-HS), SNA Formats

  The following abbreviations for chaining indicators are also used:

  - FIC (first in chain) = (BC,¬EC)
  - MIC (middle in chain) = (¬BC,¬EC)
  - LIC (last in chain) = (¬BC, EC)
  - OIC (only in chain) = (BC, EC)
- For data elements of RUs (SM-SM, HS-HS), SNA Formats

In cases where a component returns parameters in a verb, the parameters (e.g., RC), but not the verb, are named on the flow arrow.

The following conventions and abbreviations apply to all sequence flows in the book.

o————>o	asynchronous (send/receive logic) intercomponent flow
o——o——>o	asynchronous (send/receive logic) intercomponent flow with intermediate-component processing
o- - - ->o	creation or destruction of a process (action shown in parenthesis) or synchronous (call) invocation of another component (e.g., the buffer manager)
{ }	Braces surrounding alternatives indicate inclusion required.
[ ]	Brackets surrounding alternatives indicate inclusion optional.
... or :	Ellipses indicate possible repetitions or unshown continuations.
ASM	CP address space manager
BM	buffer manager
CP	control point
HS	half session
IP	initiator process
LU	logical unit
NOF	node operator facility
PC	path control
RM	LU resources manager
SM	LU session manager
SS	CP session services

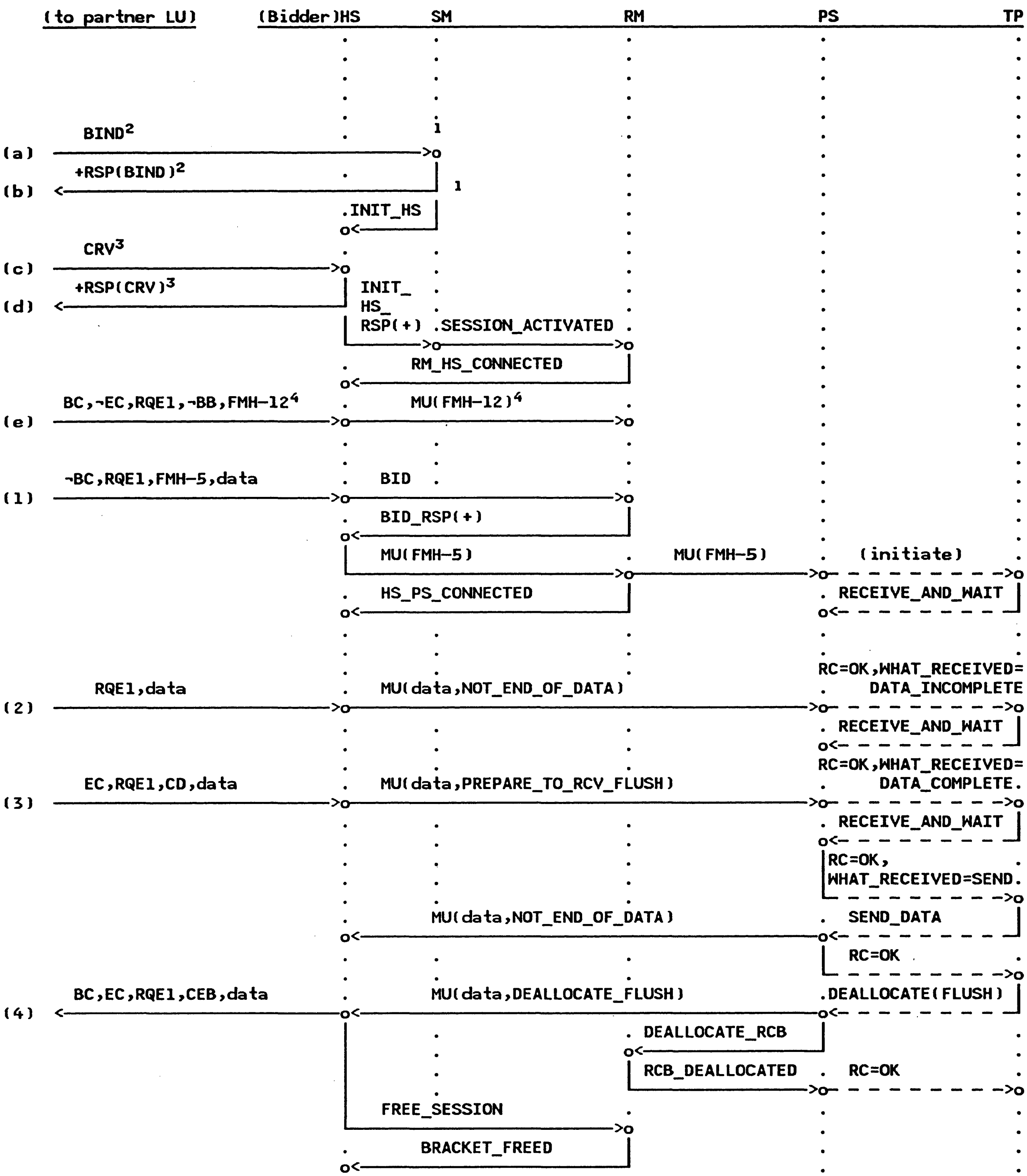
Numbers to the left of the flows correspond to enumerated annotations in the text outside (usually following) the figures. Footnotes appear in some figures to relate minor points such as signal omissions or simplification.

This page intentionally left blank

```
TP                   PS                  RM                  SM        HS(FSP)              (to partner LU)
ALLOCATE(WHEN_         .                   .                   .         .
 SESSION_ALLOCATED).  ALLOCATE_RCB         .                   .         .
o- - - - - - - - - ->o-------------------->o                   .         .
.                    .RCB_ALLOCATED(OK) |                      .         .
.                    o<-------------------+                    .         .
.                    |  GET_SESSION      .ACTIVATE_SESSION     [1]       .  BIND[2]
.                    +------------------>o------------------->o------------------------------------> (a)
.                    .                   .                   [1]         .  +RSP(BIND)[2]
.                    .                   .                   o<------------------------------------ (b)
.                    .                   .                   |INIT_HS  .
.                    .                   .                   +-------->o
.                    .                   . ACTIVATE_         . INIT_   |   CRV[3]
.                    .                   . SESSION_          . HS_     +---------------------------> (c)
. RC=OK             SESSION_ALLOCATED(OK) RSP(+)              . RSP(+)  .   +RSP(CRV)[3]
o<- - - - - - - - - -o<------------------o<------------------o<--------o<-------------------------- (d)
|                    .                   |                   .         .
|                    .                   | RM_HS_CONNECTED   .         .
|                    .                   +---------------------------->o
|                    .                   |                   .         .
|                    .                   | ENCIPHERED_RD2[4] .         .  BC,¬EC,RQE1,¬BB,FMH-12[4]
|                    .                   +---------------------------->o---------------------------> (e)
|                    .                   | HS_PS_CONNECTED   .         .
|                    .                   +---------------------------->o
| SEND_DATA          .       MU(FMH-5,data,NOT_END_OF_DATA).           .     ¬BC,RQE1,FMH-5,data
+- - - - - - - - - ->o-------------------------------------------------->o-------------------------> (1)
. RC=OK              |                   .                   .         .
o<- - - - - - - - - -+                   .                   .         .
|                    .                   .                   .         .
|                    .                   .                   .         .
|                    .                   .                   .         .
|                    .                   .                   .         .
|                    .                   .                   .         .
|                    .                   .                   .         .
|   SEND_DATA        .        MU(data,NOT_END_OF_DATA)       .         .        RQE1,data
+- - - - - - - - - ->o-------------------------------------------------->o-------------------------> (2)
.   RC=OK            |                   .                   .         .
o<- - - - - - - - - -+                   .                   .         .
|                    .                   .                   .         .
| RECEIVE_AND_WAIT   .       MU(data,PREPARE_TO_RCV_FLUSH)   .         .      EC,RQE1,CD,data
+- - - - - - - - - ->o-------------------------------------------------->o-------------------------> (3)
.                    .                   .                   .         .
.                    .                   .                   .         .
.                    .                   .                   .         .
.                    .                   .                   .         .
.                    .                   .                   .         .
.                    .                   .                   .         .
.RC=OK,              .                   .                   .         .
.WHAT_RECEIVED=      .                   .                   .         .
.     DATA_COMPLETE  .        MU(data,DEALLOCATE_FLUSH)      .         .    BC,EC,RQE1,CEB,data
o<- - - - - - - - - -o<--------------------------------------------------o<------------------------ (4)
|RECEIVE_AND_WAIT    .                   .      FREE_SESSION .         |
+- - - - - - - - - ->o                   o<------------------------------+
.RC=DEALLOCATE_NORMAL                    .                   .         .
o<- - - - - - - - - -+                   .                   .         .
|DEALLOCATE(LOCAL)   . DEALLOCATE_RCB    .                   .         .
+- - - - - - - - - ->o------------------>o                   .         .
.   RC=OK            . RCB_DEALLOCATED   |      BRACKET_FREED          .
o<- - - - - - - - - -o<------------------+------------------------------>o
```

Notes:
[1] Session-activation flows to CP and path control have been omitted;
see "Chapter 4. LU Session Manager" for details. Buffer manager calls have been omitted.
[2] BIND/RSP(BIND) flows through the CP (not shown).
[3] CRV/RSP(CRV) flows only when session-level cryptography is being used.
[4] Flows only when LU-LU verification is being used.

Figure 2-33. Complete Conversation Example--Local LU



Notes:
[1] Session-activation flows to CP and path control have been omitted.
[2] BIND/RSP(BIND) flows through the CP (not shown).
[3] CRV/RSP(CRV) flows only when session-level cryptography is being used.
[4] Flows only when LU-LU verification is being used.

Figure 2-34. Complete Conversation Example--Remote LU

```
COPR TP              PS                  RM                  SM        HS(FSP)          (to partner LU)
.                    .                   .                   .
RESET_SESSION_LIMIT¹                     .                   .
o- - - - - - - - - ->o                   .                   .
                     . (if parallel session, CNOS exchange occurs here)
                     o<-------------------------------------------------------------------------> (*)
                     .                   .                   .
                     . CHANGE_SESSIONS²  .                   .
                     o------------------>o                   .
                  ┌                      .                   .
                  |            (drain action)³               .
                  |                      .          BIS_RQ         BIS,RQ,BC,EC,RQE1,¬BB,¬CEB
  Repeat for      |                      o------------------>o--------------------------------> (1)
  each session   <                       .                   .
  for the         |                      .          BIS_REPLY      BIS,RQ,BC,EC,RQE3,¬BB,¬CEB
  specified       |                      o<------------------o<-------------------------------- (2)
  mode name.      |                      |                   .
                  |                      |DEACTIVATE_SESSION  ⁴     UNBIND⁵
                  |                      |------------------>o--------------------------------> (a)
                  |                      .                   .      +RSP(UNBIND)⁵
                  └                      .                 ⁴ o<-------------------------------- (b)
```

Notes:

[1] For specific-session deactivation, substitute DEACTIVATE_SESSION and eliminate the CNOS exchange.
[2] For specific-session deactivation, substitute RM_DEACTIVATE_SESSION and eliminate the drain action
[3] Drain action: wait until no allocation requests, allowed by drain state, are pending, then wait until session is in between-brackets state, i.e., +RSP(CEB) is sent or received.
[4] Session-deactivation flows to CP have been omitted.
[5] UNBIND/RSP(UNBIND) flows through the CP (not shown)

Figure 2-35. Session Deactivation--Local LU

```
    (to partner LU)      (Bidder)HS      SM           RM            PS        CNOS TP
                              .           .            .             .           .
                              .           .            .             .           .
                              .           .            .             .           .
      (if parallel session, CNOS exchange occurs here) .             .           .
(*)  <-------------------------------------------------------------------------->o
                              .           .            .             .
                              .           .            .             .
                              .           .            .             .
                              .           .            .           --+
                              .           .            .             |
     BIS,RQ,BC,EC,RQE1,¬BB,¬CEB.           BIS_RQ      .             |
(1)  ------------------------>o------------------------>o            |
                              .           .     (drain action)[3]    |
     BIS,RQ,BC,EC,RQE3,¬BB,¬CEB.           BIS_REPLY   .             |  Repeat for
(2)  <------------------------o<-----------------------o             > each session
                              .           .            .             |  in the mode.
      UNBIND[5]               .           .SESSION_DEACTIVATED       |
(a)  ------------------------------------>o------------>o            |
     +RSP(UNBIND)[5]          .           |            .             |
(b)  <------------------------------------+  4         .             |
                                                                   --+
```

Notes:

[3] Drain action: wait until no allocation requests allowed by drain state are pending, then wait until session is in between-brackets state, i.e., +RSP(CEB) is sent or received.
[4] Session-activation flows to PU and CP have been omitted.
[5] UNBIND/RSP(UNBIND) flows through the CP (not shown).

Figure 2-36. Session Deactivation--Remote LU

```
TP                      PS                     RM                     HS(FSP)            (to partner LU)
.                       .                      .                      .
.  ALLOCATE             .   ALLOCATE_RCB       .                      .
o- - - - - - - - - - ->o------------------->o                         .
.                       .RCB_ALLOCATED(OK) |                          .
.                       o<-----------------'                          .
.                       |   GET_SESSION     . HS_PS_CONNECTED .
.                       '------------------->o--------------------->o
. RC=OK               SESSION_ALLOCATED(OK)|                          .
o<- - - - - - - - - - -o<------------------'                          .
| SEND_DATA             .                      .                      .
'- - - - - - - - - - ->o                       .                      .
. RC=OK                 |                      .                      .
o<- - - - - - - - - - --'                      .                      .
| CONFIRM               .      MU(Attach,data,CONFIRM)               .OIC,BB,RQD2|3,FMH-5,data
'- - - - - - - - - - ->o------------------------------------------->o------------------------> (1)
.                       .                      .                      .
.                       .                      .                      .
.                       .                      .                      .
. RC=OK                 .       CONFIRMED      .                      .            +RSP
o<- - - - - - - - - - -o<--------------------------------------------o<------------------------ (2)
```

Figure 2-37. ALLOCATE(RETURN_CONTROL=WHEN_SESSION_ALLOCATED), CONFIRM (by First Speaker)--Local LU

```
           (to partner LU)              HS(Bidder)       RM                PS                TP
                                          .               .                 .                 .
                                          .               .                 .                 .
                                          .               .                 .                 .
                                          .               .                 .                 .
                                          .               .                 .                 .
      OIC,BB,RQD2|3,FMH-5¹,data           .     BID       .                 .                 .
(1)  ---------------------------------->o--------------->o                 .                 .
                                          .  BID_RSP(+)   |                 .                 .
                                         o<---------------                  .                 .
                                         | MU(Attach¹,data). MU(Attach,data) .   (initiate)   .
                                          --------------->o---------------->o- - - - - - - - ->o
                                          . HS_PS_CONNECTED |               . RECEIVE_AND_WAIT |
                                         o<-----------------                o<- - - - - - - - -
                                         |                .                 RC=OK,WHAT_RECEIVED=
                                         |          MU(data,CONFIRM)        .     DATA_*COMPLETE
                                          --------------------------------->o- - - - - - - - ->o
                                          .               .                 . RECEIVE_AND_WAIT |
                                          .               .                 o<- - - - - - - - -
                                          .               .                 RC=OK,WHAT_RECEIVED=
                                          .               .                 |          CONFIRM
                                          .               .                  - - - - - - - - ->o
                    +RSP                  .           CONFIRMED             .    CONFIRMED     |
(2)  <-----------------------------------o<---------------------------------o<- - - - - - - - -
                                                                            |    RC=OK         .
                                                                             - - - - - - - - ->o
```

Note:

[1] The FMH-5 contains the Attach

Figure 2-38. ALLOCATE(RETURN_CONTROL=WHEN_SESSION_ALLOCATED), CONFIRM (by First Speaker)--Remote LU

```
TP                 PS                 RM                 HS(Bidder)        (to partner LU)
.                  .                  .                  .
.  ALLOCATE        .   ALLOCATE_RCB   .                  .
o- - - - - - - - ->o----------------->o                  .
.                  RCB_ALLOCATED(OK) |                   .
.                  o<----------------┘                   .
.                  | GET_SESSION      BID_WITHOUT_ATTACH.   LUSTAT,BB,RQD1
.                  └----------------->o----------------->o------------------------------> (1)
.  RC=OK           SESSION_ALLOCATED(OK)  BID_RSP(+)     .              +RSP
o<- - - - - - - - -o<-----------------o<-----------------o<------------------------------ (2)
| SEND_DATA        .                  | HS_PS_CONNECTED .
└- - - - - - - - ->o                  └----------------->o
.  RC=OK           |                  .                  .
o<- - - - - - - - -┘                  .                  .
|RECEIVE_AND_WAIT  .MU(Attach,data,PREPARE_TO_RCV_FLUSH).   OIC,RQE1,CD,FMH-5,data
└- - - - - - - - ->o------------------------------------>o------------------------------> (3)
```

Figure 2-39. ALLOCATE(RETURN_CONTROL=WHEN_SESSION_ALLOCATED), RECEIVE_AND_WAIT (by Bidder)--Local LU

```
     (to partner LU)              HS(FSP)        RM               PS              TP
                                     .            .                .               .
                                     .            .                .               .
                                     .            .                .               .
                                     .            .                .               .
                                     .            .                .               .
         LUSTAT,BB,RQD1              .    BID     .                .               .
(1)  ------------------------------->o----------->o                .               .
               +RSP                  .  BID_RSP(+) |               .               .
(2)  <-------------------------------o<------------'               .               .
                                     .            .                .               .
                                     .            .                .               .
                                     .            .                .               .
                                     .            .                .               .
       OIC,RQE1,CD,FMH-5,data        . MU(Attach,data) . MU(Attach,data) .  (initiate)  .
(3)  ------------------------------->o----------->o--------------->o- - - - - - - - ->o
                                     . HS_PS_CONNECTED |           . RECEIVE_AND_WAIT |
                                     o<----------------'           o<- - - - - - - - -'
                                     |            .                RC=OK,WHAT_RECEIVED=
                                     |  MU(data,PREPARE_TO_RCV_FLUSH)  .   DATA_COMPLETE
                                     '---------------------------->o- - - - - - - - ->o
                                     .            .                . RECEIVE_AND_WAIT |
                                     .            .                o<- - - - - - - - -'
                                     .            .                | RC=OK,
                                     .            .                |WHAT_RECEIVED=SEND
                                     .            .                '- - - - - - - - ->o
```

Figure 2-40. ALLOCATE(RETURN_CONTROL=WHEN_SESSION_ALLOCATED), RECEIVE_AND_WAIT (by Bidder)--Remote LU

```
TP                   PS                  RM                  HS(Bidder)      (to partner LU)
.                    .                   .                   .
.  ALLOCATE          .  ALLOCATE_RCB     .                   .
o- - - - - - - - - ->o------------------>o                   .
.                    RCB_ALLOCATED(OK) |                     .
.                    o<----------------+                     .
.                    | GET_SESSION      BID_WITHOUT_ATTACH.     LUSTAT,BB,RQD1
.                    +------------------>o------------------>o------------------------------> (1)
.  RC=OK            SESSION_ALLOCATED(OK)   BID_RSP(+)       .               +RSP
o<- - - - - - - - - -o<------------------o<------------------o<------------------------------ (2)
| SEND_DATA          .                   | HS_PS_CONNECTED   .
+- - - - - - - - - ->o                   +------------------>o
.  RC=OK             |                   .                   .
o<- - - - - - - - - -+                   .                   .
| CONFIRM            .      MU(Attach,data,CONFIRM)          .   OIC,RQD2|3,FMH-5,data
+- - - - - - - - - ->o-------------------------------------->o------------------------------> (3)
.                    .                   .                   .
.                    .                   .                   .
.                    .                   .                   .
.                    .                   .                   .
.                    .         RECEIVED_ERROR                .          -RSP(0846)
                     o<--------------------------------------o<------------------------------ (4)
.                    .                   .                   .
.                    .                   .                   .
.                    .                   .                   .
.                    .                   .                   .
RC=ALLOCATION_ERROR.    MU(FMH-7,data¹,DEALLOCATE_FLUSH)     .   OIC,CEB,RQE1,FMH-7,data¹
o<- - - - - - - - - -o<--------------------------------------o<------------------------------ (5)
|DEALLOCATE(LOCAL)    DEALLOCATE_RCB       FREE_SESSION      |
+- - - - - - - - - ->o------------------>o<------------------+
.  RC=OK             . RCB_DEALLOCATED   | BRACKET_FREED     .
o<- - - - - - - - - -o<------------------+------------------>o
```

Note:
[1] Optional log data

Figure 2-41. ALLOCATE(RETURN_CONTROL=WHEN_SESSION_ALLOCATED), CONFIRM (by Bidder), Attach Error --Local LU

```
  (to partner LU)                    HS(FSP)         RM               PS             TP
                                        .             .                .              .
                                        .             .                .              .
     LUSTAT,BB,RQD1                     .    BID      .                .              .
(1) ---------------------------------->o------------->o                .              .
               +RSP                     .  BID_RSP(+) |                .              .
(2) <----------------------------------o<-------------+                .              .
                                        .             .                .              .
                                        .             . MU(Attach,data,                .
     OIC,RQD2|3,FMH-5,data              . MU(Attach,data) . error sense code).        .
(3) ---------------------------------->o------------->o---------------->o             .
                                        . HS_PS_CONNECTED|               |             .
                                       o<-------------+                |             .
                                        | MU(data,CONFIRM).              |             .
                                        +------------------+             |             .
                    -RSP(0846)          .        SEND_ERROR|             |             .
(4) <----------------------------------o<------------------ ------------+             .
                                        .             .    |           .              .
                                        .             .    +----------->o             .
                                        .             .                |              .
        OIC,CEB,RQE1,FMH-7,data1.  MU(FMH-7,data1,DEALLOCATE_FLUSH)      |              .
(5) <----------------------------------o<-------------------------------+             .
                                        | FREE_SESSION  . DEALLOCATE_RCB |             .
                                        +------------->o<---------------+             .
                                        . BRACKET_FREED | RCB_DEALLOCATED .             .
                                       o<-------------+---------------->o             .
```

Note:
[1] Optional log data

Figure 2-42. ALLOCATE(RETURN_CONTROL=WHEN_SESSION_ALLOCATED), CONFIRM (by Bidder), Attach Error--Remote LU

```
TP                  PS                  RM                  HS(FSP)            (to partner LU)
.                   .                   .                   .
. ALLOCATE          ALLOCATE_RCB(immediate)                 .
o- - - - - - - - - ->o------------------>o                  .
.                   .          FSP session available        .
. RC=OK             .RCB_ALLOCATED(OK) |                    .
o<- - - - - - - - - -o<----------------+                    .
.                   .                  |  HS_PS_CONNECTED   .
.                   .                  +------------------->o
                             •
                             •
                             •
```

[The flow continues as in the ALLOCATE(RETURN_CONTROL=WHEN_SESSION_ALLOCATED) case.]

Figure 2-43. ALLOCATE(RETURN_CONTROL=IMMEDIATE), Successful--Local LU (to partner LU)   HS   RM   PS   TP

(no activity at remote LU)

from here on just like ALLOCATE(RETURN_CONTROL=WHEN_SESSION_ALLOCATED)

Figure 2-44. ALLOCATE(RETURN_CONTROL=IMMEDIATE), Successful--Remote LU

```
TP                   PS                  RM                 HS                  (to partner LU)
.                    .                   .                  .
.  ALLOCATE          . ALLOCATE_RCB(immediate)              .
o- - - - - - - - - ->o------------------>o                  .
.                    .         (no first-speaker            .
.                    .          session available)          .
.                    .RCB_ALLOCATED     |                   .
. RC=UNSUCCESSFUL    .  (unsuccessful)  |                   .
o<- - - - - - - - - -o<-----------------+                   .
```

Figure 2-45. ALLOCATE(RETURN_CONTROL=IMMEDIATE), Unsuccessful--Local LU

(to partner LU)	HS	RM	PS	TP
		(no activity at remote LU)		

Figure 2-46. ALLOCATE(RETURN_CONTROL=IMMEDIATE), Unsuccessful--Remote LU

```
TP                   PS                   RM                   HS                 (to partner LU)
.                    .                    .                    .
.                    .                    .                    .
.                    .                    .                    .
.DEALLOCATE(FLUSH)   .    MU(DEALLOCATE_FLUSH)                 .       LIC,CEB,RQE1
o- - - - - - - - - ->o------------------------------------------>o-------------------------------> (1)
.                    |                    .    FREE_SESSION    |
.                    |                    o<-------------------|
.                    |  DEALLOCATE_RCB    .                    .
.                    |------------------->o                    .
.  RC=OK             . RCB_DEALLOCATED    |  BRACKET_FREED     .
o<- - - - - - - - - -o<-------------------|------------------->o
```

Figure 2-47. DEALLOCATE(TYPE=FLUSH) (RQE1)--Local LU

```
    (to partner LU)             HS                  RM                  PS                 TP
                                .                   .                   .                  .
                                .                   .                   . RECEIVE_AND_WAIT .
                                .                   .                   o<- - - - - - - - -o
        LIC,CEB,RQE1            .  RCVD_DATA(DEALLOCATE_FLUSH)          RC=DEALLOCATE_NORMAL
(1) ---------------------------->o------------------------------------->o- - - - - - - - ->o
                                | FREE_SESSION     . DEALLOCATE_RCB     DEALLOCATE_LOCAL   |
                                 ------------------>o<------------------o<- - - - - - - - --
                                .                   |                   .                  .
                                .                   |                   .                  .
                                . BRACKET_FREED     | RCB_DEALLOCATED   .     RC=OK        .
                                o<------------------------------------->o- - - - - - - - ->o
```

Figure 2-48. DEALLOCATE(TYPE=FLUSH) (RQE1)--Remote LU

```
TP                  PS                  RM                  HS               (to partner LU)
.                   .                   .                   .
.                   .                   .                   .
.                   .                   .   (sequence number wrap)
.DEALLOCATE(FLUSH) .        MU(DEALLOCATE_FLUSH)            .        LIC,CEB,RQD1¹
o- - - - - - - - - >o-------------------------------------->o---------------------------> (1)
.                   |                   .  FREE_SESSION     .             +RSP
.                   |                   o<------------------o<-------------------------- (2)
.                   |  DEALLOCATE_RCB   .                   .
.                   |------------------>o                   .
.  RC=OK            .  RCB_DEALLOCATED  |  BRACKET_FREED.   .
o<- - - - - - - - - -o<-----------------|------------------>o
```

NOTES:
[1] RQD1 is required under certain sequence number wrap conditions.

Figure 2-49. DEALLOCATE(TYPE=FLUSH) (RQD1)--Local LU

```
    (to partner LU)              HS                 RM                  PS                    TP
                                 .                  .                   .                     .
                                 .                  .                   . RECEIVE_AND_WAIT .
                                 .                  .                   o<- - - - - - - - - -o
      LIC,CEB,RQD1               .          MU(DEALLOCATE_FLUSH)      RC=DEALLOCATE_NORMAL.
(1) ----------------------------->o------------------------------------->o- - - - - - - - - ->o
                   +RSP          |                  . DEALLOCATE_RCB    DEALLOCATE_LOCAL    |
(2) <----------------------------                   o<-------------------o<- - - - - - - - - -
                                 | FREE_SESSION     .                   .                     .
                                 ------------------>o                   .                     .
                                 . BRACKET_FREED    | RCB_DEALLOCATED   .     RC=OK           .
                                 o<------------------------------------->o- - - - - - - - - ->o
```

Figure 2-50. DEALLOCATE(TYPE=FLUSH) (RQD1)--Remote LU

```
TP                  PS                  RM                  HS                (to partner LU)
.                   .                   .                   .
.                   .                   .                   .
.                   .                   .                   .
.  SEND_DATA        .     MU(data,NOT_END_OF_DATA)          . FIC,data
o- - - - - - - - - >o------------------------------------->o---------------------------> (1)
.  RC=OK            |                   .                   .
o<- - - - - - - - - -                   .                   .
| DEALLOCATE(FLUSH).      MU(data,DEALLOCATE_FLUSH)         . LIC,CEB,RQE1,data
L- - - - - - - - - >o------------------------------------->o-----------------
.                   |                   . FREE_SESSION      |                |
.                   |                   o<------------------                 |
.                   | DEALLOCATE_RCB    .                   .       -RSP(0846)|
.                   ------------------->o                   o<----------------|------------ (2)
.                   .                   |     (This stray response            |
.                   .                   |        is discarded)                ------------> (3)
.                   .                   |                   .
.                   .                   |                   .
.  RC=OK            . RCB_DEALLOCATED   |   BRACKET_FREED   .
o<- - - - - - - - - -o<----------------------------------->o
```

Figure 2-51. DEALLOCATE(TYPE=FLUSH) (RQE1), SEND_ERROR, -RSP Sent--Local LU

```
(to partner LU)                HS                 RM                  PS                TP
                               .                  .                   .                  .
                               .                  .                   . RECEIVE_AND_WAIT .
                               .                  .                   o<- - - - - - - - -o
    FIC,data                   .     MU(data,NOT_END_OF_DATA)         . RC=OK,           .
(1) ---------------------------->o----------------------------------->o WHAT_RECEIVED=   .
                               .                  .                   | DATA_INCOMPLETE  .
                               .                  .                   L- - - - - - - - ->o
                               .                  .                   .                  |
                               .                  .                   .                  |
                               .                  .                   .                  |
                               .                  .                   .                  |
         -RSP(0846)            .        SEND_ERROR                    .  SEND_ERROR      |
(2) <----------------------------o<-----------------------------------o<- - - - - - - - -
    LIC,CEB,RQE1,data          .     MU(data,DEALLOCATE_FLUSH)        RC=DEALLOCATE_NORMAL
(3) ---------------------------->o----------------------------------->o- - - - - - - - ->o
                               |                          (purge data)                   |
                               | FREE_SESSION     . DEALLOCATE_RCB    . DEALLOCATE(LOCAL)|
                               L----------------->o<------------------o<- - - - - - - - -
                               .  BRACKET_FREED   | RCB_DEALLOCATED   .   RC=OK          .
                               o<-----------------+------------------>o- - - - - - - - ->o
```

Figure 2-52. DEALLOCATE(TYPE=FLUSH) (RQE1), SEND_ERROR, -RSP Sent--Remote LU

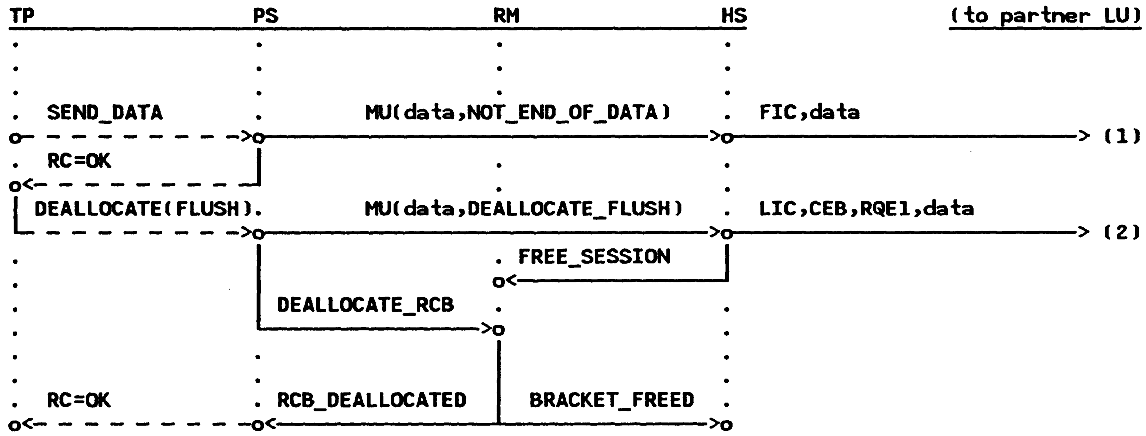


Figure 2-53. DEALLOCATE(TYPE=FLUSH) (RQE1), SEND_ERROR, -RSP not Sent--Local LU

```
    (to partner LU)                HS               RM                   PS                TP
                                   .                .                    .                 .
                                   .                .                    . RECEIVE_AND_WAIT .
                                   .                .                    o<- - - - - - - - -o
     FIC,data                      .   MU(data,NOT_END_OF_DATA)          . RC=OK,          .
(1)  ----------------------------->o------------------------------------>o WHAT_RCVD=      .
                                   .                .                    | DATA_*COMPLETE  .
                                   .                .                    └ - - - - - - - - ->o
     LIC,CEB,RQE1,data             .   MU(data,DEALLOCATE_FLUSH)         .                 |
(2)  ----------------------------->o------------------------------------>o                 |
                                   | FREE_SESSION   .                    .   SEND_ERROR    |
                                   └--------------->o                    o<- - - - - - - --┘
                                   .                .         (purge data)                 .
                                   .                .                    |RC=DEALLOCATE_NORMAL
                                   .                .                    └ - - - - - - - - ->o
                                   .                . DEALLOCATE_RCB     .DEALLOCATE(LOCAL) |
                                   .                o<-------------------o<- - - - - - - --┘
                                   . BRACKET_FREED  | RCB_DEALLOCATED    .   RC=OK         .
                                   o<---------------┴------------------->o- - - - - - - - ->o
```

Figure 2-54. DEALLOCATE(TYPE=FLUSH) (RQE1), SEND_ERROR, -RSP not Sent--Remote LU

```
TP                   PS                   RM                   HS                (to partner LU)
.                    .                    .                    .
.                    .                    .                    .
.                    .                    .                    .
.                    .                    .                    .
DEALLOCATE(CONFIRM). MU(DEALLOCATE_CONFIRM)                    .     EC,CEB,RQD2|3
o- - - - - - - - - ->o---------------------------------------->o----------------------------> (1)
.                    .             CONFIRMED                   .          +RSP
.                    o<----------------------------------------o<---------------------------- (2)
.                    | DEALLOCATE_RCB     . FREE_SESSION       |
.                    |------------------->o<-------------------|
.                    .                    |                    .
.                    .                    |                    .
.                    .                    |                    .
.                    .                    |                    .
.                    .                    |                    .
.                    .                    |                    .
.                    .                    |                    .
.  RC=OK             . RCB_DEALLOCATED    |  BRACKET_FREED     .
o<- - - - - - - - - -o<-------------------------------------->o
```

Figure 2-55. DEALLOCATE(TYPE=CONFIRM) (RQD2|3)--Local LU

```
(to partner LU)           HS                    RM                    PS                   TP
                          .                     .                     .
                          .                     .                     . RECEIVE_AND_WAIT
                          .                     .                     o<- - - - - - - - - -o
                          .                     .                     RC=OK,WHAT_RECEIVED=
     EC,CEB,RQD2|3        .     MU(DEALLOCATE_CONFIRM)                . CONFIRM_DEALLOCATE
(1) ---------------------->o------------------------------------------>o- - - - - - - - - ->o
                +RSP      .            CONFIRMED                      .   CONFIRMED        |
(2) <----------------------o<------------------------------------------o<- - - - - - - - - -
                          | FREE_SESSION        .                     |   RC=OK
                          ---------------------->o                     - - - - - - - - - - ->o
                          .                     . DEALLOCATE_RCB      .DEALLOCATE(LOCAL)   |
                          .                     o<---------------------o<- - - - - - - - - -
                          .  BRACKET_FREED      | RCB_DEALLOCATED     .   RC=OK
                          o<----------------------------------------->o- - - - - - - - - ->o
```

Figure 2-56. DEALLOCATE(TYPE=CONFIRM) (RQD2|3)--Remote LU

```
TP                   PS                  RM                  HS                 (to partner LU)
.                    .                   .                   .
.                    .                   .                   .
.                    .                   .                   .
. DEALLOCATE         .                   .                   .
. (ABEND_PROG)       .  MU(FMH-7,data[1],DEALLOCATE_FLUSH)  .OIC,CEB,RQD1,FMH-7(0864),data[1]
o- - - - - - - - - ->o-------------------------------------->o-------------------------------> (1)
.                    | DEALLOCATE_RCB   . FREE_SESSION       .              +RSP
.                    |------------------>o<------------------o<------------------------------- (2)
.   RC=OK            . RCB_DEALLOCATED  |  BRACKET_FREED     .
o<- - - - - - - - - -o<------------------------------------->o
```

Note:
[1] Optional log data

Figure 2-57. DEALLOCATE(TYPE=ABEND_PROG) Issued in SEND_STATE, Between-Chain State--Local LU

```
    (to partner LU)              HS                 RM                 PS                  TP
                                 .                  .                  .                   .
                                 .                  .                  . RECEIVE_AND_WAIT  .
                                 .                  .                  o<- - - - - - - - - o
                                 .                  .                  .  RC=DEALLOCATE_   .
    OIC,CEB,RQD1,FMH7(0864),data1. MU(FMH-7,data1,DEALLOCATE_FLUSH)    .    ABEND_PROG     .
(1)  ---------------------------->o-------------------------------------->o- - - - - - - - - ->o
                   +RSP          |                  .DEALLOCATE_RCB    . DEALLOCATE(LOCAL)|
(2)  <---------------------------                   o<-----------------o<- - - - - - - - - -
                                 |                  | RCB_DEALLOCATED  .   RC=OK           .
                                 |                   ----------------->o- - - - - - - - - ->o
                                 |  FREE_SESSION    .                  .                   .
                                  ----------------->o                  .                   .
                                 .  BRACKET_FREED   |                  .                   .
                                 .<-----------------                   .                   .
```

Note:
[1] Optional log data

Figure 2-58. DEALLOCATE(TYPE=ABEND_PROG) Issued in SEND_STATE, Between-Chain State--Remote LU

```
TP                       PS                       RM                     HS                  (to partner LU)
.                        .                        .                      .
.                        .                        .                      .
.                        .                        .                      .
.   SEND_DATA            .     MU(data,NOT_END_OF_DATA)                  .  FIC,data
o- - - - - - - - - ->o----------------------------------------------->o------------------------------> (1)
.   RC=OK                |                        .                      .
o<- - - - - - - - - -----                         .                      .
|                        .                        .                      .
| DEALLOCATE             .                        .                      .
| (ABEND_PROG)           .            MU(data, FLUSH)                    .  RQE1,data
|_ - - - - - - - - ->o----------------------------------------------->o------------------------------> (2)
.                        |                        .                      .
.                        |                        .                      .
.                        |                        .                      .
.                        |    MU(FMH-7,DEALLOCATE_FLUSH)                 .  LIC,CEB,RQD1,FMH-7(0864)
.                        |----------------------------------------------->o------------------------------> (3)
.                        |  DEALLOCATE_RCB      .   FREE_SESSION         .              +RSP
.                        |------------------->o<---------------------o<------------------------------ (4)
.   RC=OK                .  RCB_DEALLOCATED     |  BRACKET_FREED         .
o<- - - - - - - - - -o<---------------------------------------------->o
```

Figure 2-59. DEALLOCATE(TYPE=ABEND_PROG) Issued in SEND_STATE, In-Chain State--Local LU

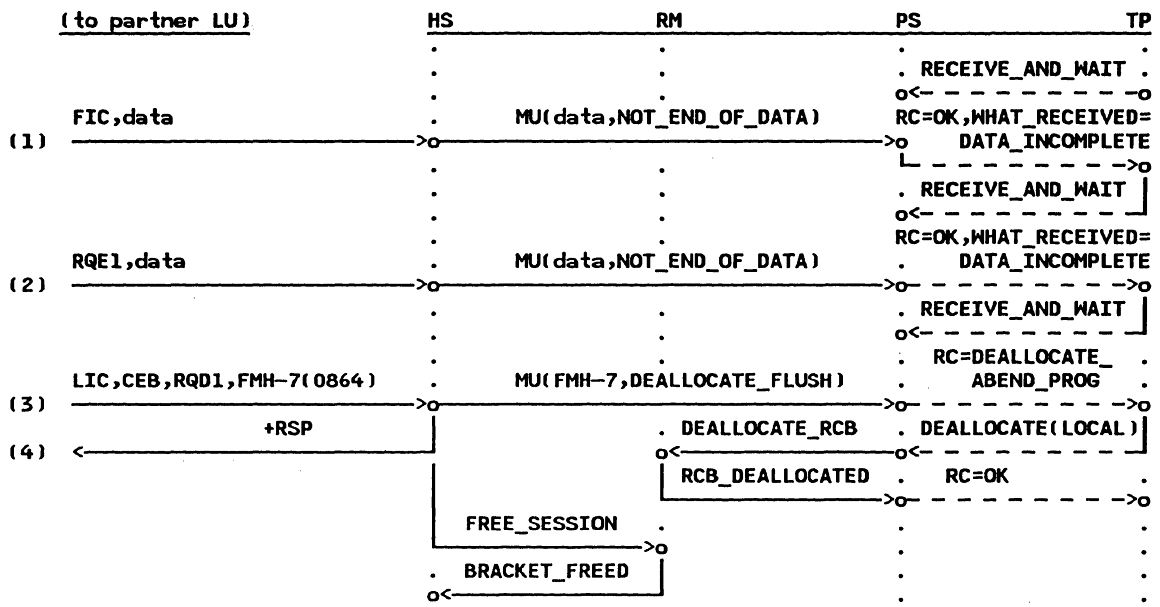


Figure 2-60. DEALLOCATE(TYPE=ABEND_PROG) Issued in SEND_STATE, In-Chain State--Remote LU

```
TP                 PS                 RM                 HS                 (to partner LU)
.                  .                  .                  .
.  SEND_DATA       .                  .                  .
o- - - - - - - - ->o                  .                  .
.  RC=OK           |                  .                  .
o<- - - - - - - - -┘                  .                  .
.    FLUSH         .      MU(data,NOT_END_OF_DATA)       .  FIC,data
o- - - - - - - - ->o------------------------------------>o------------------------------> (1)
.                  |                  .                  .
o<- - - - - - - - -┘                  .                  .
|                  .      MU(RECEIVE_ERROR)              .                  -RSP(0846)
|                  o<------------------------------------o<------------------------------ (2)
| DEALLOCATE       .                  .                  .
| (ABEND_PROG)     .  MU(FMH-7,data[1],DEALLOCATE_FLUSH) .LIC,CEB,RQD1,FMH-7(0864),data[1]
└ - - - - - - - - >o------------------------------------>o------------------------------> (3)
.                  | DEALLOCATE_RCB   . FREE_SESSION     |
.                  └----------------->o<-----------------┘
.  RC=OK           . RCB_DEALLOCATED  | BRACKET_FREED    .
o<- - - - - - - - -o<-----------------┴----------------->o
```

Note:
[1] Optional log data

Figure 2-61. DEALLOCATE(TYPE=ABEND_PROG) Issued in SEND_STATE, -RSP Received State--Local LU

```
     (to partner LU)              HS                 RM                  PS                TP
                                  .                  .                   .                  .
                                  .                  .                   .                  .
                                  .                  .                   . RECEIVE_AND_WAIT .
                                  .                  .                   o<- - - - - - - - -o
                                  .                  .                   RC=OK,WHAT_RECEIVED=
       FIC,data                   .      MU(data,NOT_END_OF_DATA)        . DATA_INCOMPLETE  .
(1)  ---------------------------->o------------------------------------->o- - - - - - - - ->o
                                  .                  .                   .                  |
                                  .                  .                   .                  |
                   -RSP(0846)     .             SEND_ERROR               .    SEND_ERROR    |
(2)  <----------------------------o<-------------------------------------o<- - - - - - - - -'
                                  .                  .                   .                  .
       LIC,CEB,RQD1,FMH-7(0864)   .  MU(FMH-7,data[1],DEALLOCATE_FLUSH)  RC=DEALLOCATE_NORMAL
(3)  ---------------------------->o------------------------------------->o- - - - - - - - ->o
                                  |                  .               (purge)                |
                                  |  FREE_SESSION    . DEALLOCATE_RCB    .    DEALLOCATE    |
                                  '----------------->o<------------------o<- - - - - - - - -'
                                  . BRACKET_FREED    | RCB_DEALLOCATED   .      RC=OK       .
                                  o<-----------------'------------------>o- - - - - - - - ->o
```

Notes:

[1] Optional log data

This TP gets no indication that the DEALLOCATE is of type ABEND because everything (including FM headers) is discarded when purging.

Figure 2-62. DEALLOCATE(TYPE=ABEND_PROG) Issued in SEND_STATE, -RSP Received State--Remote LU

```
TP                  PS                  RM                  HS              (to partner LU)
.                   .                   .                   .
.                   .                   .                   .
.                   .                   .                   .
.   SEND_DATA       .     MU(data,NOT_END_OF_DATA)          .  FIC,data
o- - - - - - - - - >o------------------------------------->o-------------------------------> (1)
.   RC=OK           |                   .                   .
o<- - - - - - - - - -                   .                   .
|                   .                   .                   .
|  DEALLOCATE       .                   .                   .
|  (ABEND_PROG)     .        MU(data, FLUSH)                .  RQE1, data
L- - - - - - - - - >o------------------------------------->o-------------------------------> (2)
.                   |                                       .
.                   |   MU(FMH-7,DEALLOCATE_FLUSH)          .  LIC,CEB,RQD1,FMH-7(0864)
.                   L------------------------------------->o-------------------------------> (3)
.                   |                   .                   .
.                   |  DEALLOCATE_RCB   .  FREE_SESSION     .                     -RSP(0846)
.                   L----------------->o<-------------------o<------------------------------ (4)
.  RC=OK            . RCB_DEALLOCATED   |  BRACKET_FREED    .
o<- - - - - - - - - o<-------------------------------------->o
```

Figure 2-63. DEALLOCATE(TYPE=ABEND_PROG) Issued in SEND_STATE Crossing SEND_ERROR--Local LU

```
    (to partner LU)              HS                 RM                 PS                   TP
                                 .                  .                  .                    .
                                 .                  .                  . RECEIVE_AND_WAIT   .
                                 .                  .                  o<- - - - - - - - - -o
     FIC,data                    .    MU(data,NOT_END_OF_DATA)         RC=OK,WHAT_RECEIVED=
(1)  --------------------------->o------------------------------------>o     DATA_INCOMPLETE
                                                    .                  L- - - - - - - - - ->o
                                 .                  .                  .                    |
                     -RSP(0846). .          SEND_ERROR                 . SEND_ERROR         |
                  +--------------o<------------------------------------o<- - - - - - - - - -+
        RQE1,     | data         .    MU(data,NOT_END_OF_DATA)         .                    .
(2)  -------------|------------->o------------------------------------>o                    .
                  |              .                               (purge data)               .
     LIC,CEB,     | RQD1,FMH-7(0864) . MU(FMH-7,DEALLOCATE_FLUSH)      RC=DEALLOCATE_NORMAL
(3)  -------------|------------->o------------------------------------>o- - - - - - - - - ->o
                  |              .                  .            (purge FMH-7)              |
                  |              | FREE_SESSION     . DEALLOCATE_RCB   . DEALLOCATE(LOCAL)  |
(4)  <------------+              +---------------->o<------------------o<- - - - - - - - - -+
                                 . BRACKET_FREED    | RCB_DEALLOCATED  .   RC=OK            .
                                 o<-----------------+----------------->o- - - - - - - - - ->o
```

NOTE: TPN on right gets no indication that DEALLOCATE_ABEND occurred because everything (including FMHs) are discarded when in purge state.

Figure 2-64. DEALLOCATE(TYPE=ABEND_PROG) Issued in SEND_STATE Crossing SEND_ERROR--Remote LU

```
TP                  PS                  RM                 HS              (to partner LU)
.                   .                   .                  .
.              in RCV state             .                  .
.DEALLOCATE         .                   .                  .
.(ABEND_PROG)       .       SEND_ERROR  .                  .
o- - - - - - - - - >o------------------------------------->o
.                   .    MU(data,NOT_END_OF_DATA)          .  FIC,data
.                   o<-------------------------------------o<----------------------------- (1)
.             (purge data)              .                  |
.                   .                   .                  |
.                   .                   .                  |  -RSP(0846)
.                   .                   .                  |---------------------------> (2)
.                   .    MU(PREPARE_TO_RCV_FLUSH)          .  LIC,RQE1,CD,no data
.                   o<-------------------------------------o<----------------------------- (3)
.                   |                   .                  .
.                   |                   .                  .
.                   |    MU(FMH-7,DEALLOCATE_FLUSH)        .  OIC,CEB,RQD1,FMH-7(0864)
.                   |------------------------------------->o---------------------------> (4)
.                   | DEALLOCATE_RCB    . FREE_SESSION     .             +RSP
.                   |------------------>o<-----------------o<----------------------------- (5)
.  RC=OK            . RCB_DEALLOCATED   | BRACKET_FREED    .
o<- - - - - - - - - o<------------------|----------------->o
```

Figure 2-65. DEALLOCATE(TYPE=ABEND_PROG) Issued in RCV_STATE, Between-Chain State--Local LU

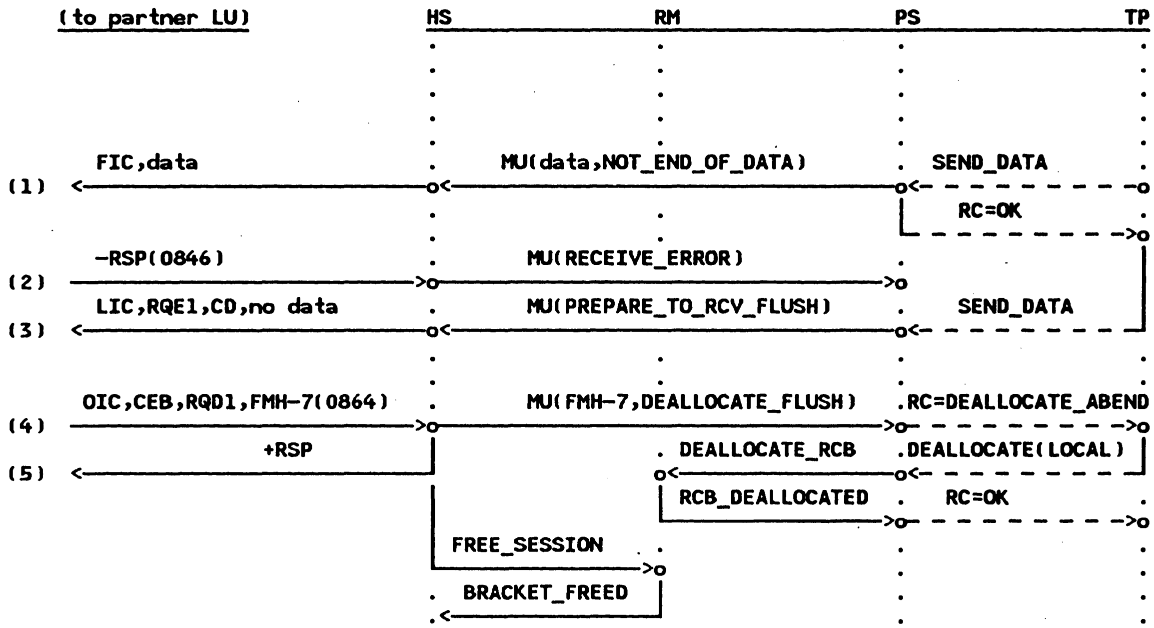


Figure 2-66. DEALLOCATE(TYPE=ABEND_PROG) Issued in RCV_STATE, Between-Chain State--Remote LU

```
TP                  PS                  RM                  HS                  (to partner LU)
.                   .                   .                   .
.                   .                   .                   .
.RECEIVE_AND_WAIT   .                   .                   .
o- - - - - - - - - ->o                  .                   .
RC=OK,WHAT_RECEIVED=                    .                   .
.    DATA_INCOMPLETE       MU(data,NOT_END_OF_DATA)         .  FIC,data
o<- - - - - - - - - -o<-------------------------------------o<--------------------------- (1)
|                   .                   .                   .
| DEALLOCATE        .                   .                   .
| (ABEND_PROG)      .      SEND_ERROR   .                   .  -RSP(0846)
L- - - - - - - - - ->o------------------------------------->o---------------------------> (2)
.                   .      MU(PREPARE_TO_RCV_FLUSH)         . LIC,RQE1,CD,no data
.                   o<--------------------------------------o<--------------------------- (3)
.                   |                   .                   .
.                   |                   .                   .
.                   |       MU(FMH-7,DEALLOCATE_FLUSH)      . OIC,CEB,RQD1,FMH-7(0864)
.                   L-------------------------------------->o---------------------------> (4)
.                   | DEALLOCATE_RCB    .  FREE_SESSION     .                +RSP
.                   L------------------>o<------------------o<--------------------------- (5)
.  RC=OK            . RCB_DEALLOCATED   |  BRACKET_FREED    .
o<- - - - - - - - - -o<-----------------L------------------>o
```

Figure 2-67. DEALLOCATE(TYPE=ABEND_PROG) Issued in RCV_STATE, In-Chain State--Local LU

```
    (to partner LU)           HS                   RM                    PS                 TP
                               .                    .                     .                  .
                               .                    .                     .                  .
                               .                    .                     .                  .
                               .                    .                     .                  .
        FIC,data               .      MU(data,NOT_END_OF_DATA)            .    SEND_DATA     .
(1)  <-------------------------o<-----------------------------------------o<- - - - - - - -o
                               .                    .                     |    RC=OK         .
                               .                    .                     L- - - - - - - - ->o
      -RSP(0846)               .      MU(RECEIVE_ERROR)                   .                  |
(2)  ------------------------->o----------------------------------------->o                  |
      LIC,RQE1,CD,no data      .      MU(PREPARE_TO_RCV_FLUSH)            .     SEND_DATA    |
(3)  <-------------------------o<-----------------------------------------o<- - - - - - - - -
                               .                    .                     .                  .
                               .                    .                     . RC=DEALLOCATE_   .
     OIC,CEB,RQD1,FMH7(0864)   .      MU(FMH-7,DEALLOCATE_FLUSH)          .    ABEND_PROG    .
(4)  ------------------------->o----------------------------------------->o- - - - - - - - ->o
                     +RSP      |                    . DEALLOCATE_RCB      .DEALLOCATE(LOCAL) |
(5)  <-------------------------                     o<--------------------o<- - - - - - - - -
                               |                    | RCB_DEALLOCATED     .   RC=OK          .
                               |                    L-------------------->o- - - - - - - - ->o
                               |   FREE_SESSION     .                     .                  .
                               L------------------->o                     .                  .
                               . BRACKET_FREED      |                     .                  .
                               o<-------------------                      .                  .
```

Figure 2-68. DEALLOCATE(TYPE=ABEND_PROG) Issued in RCV_STATE, In-Chain State--Remote LU

```
TP                    PS                    RM                   HS                (to partner LU)

    SEND_DATA
o- - - - - - - - - ->o
    RC=OK            |
o<- - - - - - - - - -┘
    CONFIRM                  MU(data,CONFIRM)           OIC,RQD2|3,data
o- - - - - - - - - ->o------------------------------->o-------------------------------> (1)


    RC=OK                      CONFIRMED                                  +RSP
o<- - - - - - - - - -o<-------------------------------o<------------------------------- (2)
```

Figure 2-69. CONFIRM (RQD2|3)--Local LU

```
     (to partner LU)            HS            RM             PS             TP

                                                           RECEIVE_AND_WAIT
                                                          o<- - - - - - - - -o
                                                         RC=OK,WHAT_RECEIVED=
    OIC,RQD2|3,data                     MU(data,CONFIRM)       DATA_*COMPLETE
(1) --------------------------->o------------------------->o- - - - - - - - ->o
                                                           RECEIVE_AND_WAIT |
                                                          o<- - - - - - - - -
                                                          |RC=OK,
                                                          |WHAT_RECEIVED=CONFIRM
                                                          L- - - - - - - - ->o
              +RSP                    CONFIRMED                CONFIRMED    |
(2) <---------------------------o<------------------------o<- - - - - - - - -
                                                          |    RC=OK
                                                          L- - - - - - - - ->o
```

Figure 2-70. CONFIRM (RQD2|3)--Remote LU

```
TP                   PS                   RM                   HS              (to partner LU)
.                    .                    .                    .
. SEND_DATA          .                    .                    .
o- - - - - - - - - ->o                    .                    .
. RC=OK              |                    .                    .
o<- - - - - - - - - -┘                    .                    .
|PREPARE_TO_RECEIVE.                      .                    .
|(TYPE=CONFIRM,      .                    .                    .
| LOCKS=LONG)        MU(data,PREPARE_TO_RCV_CONFIRM_LONG) . OIC,RQE2|3,CD,data
└- - - - - - - - - ->o------------------------------------->o------------------------------> (1)
.                    .                    .                    .
.                    .                    .                    .
.                    .                    .                    .
.                    .                    .                    .
.                    .                    .                    .
.                    .                    .                    .
.                    .                    .                    .
.                    .                    .                    .
.                    .                    .                    .
.  RC=OK             .         CONFIRMED                       .          FIC,data
o<- - - - - - - - - -o<-----------------------------------------o<----------------------------- (2)
| RECEIVE_AND_WAIT .                      .                    |
└- - - - - - - - - ->o                    .                    |
.RC=OK,WHAT_RECEIVED=                     .                    |
.   DATA_INCOMPLETE       MU(data,NOT_END_OF_DATA)             |
o<- - - - - - - - - -o<-----------------------------------------┘
```

Figure 2-71. CONFIRM (RQE2|3)--Local LU

```
     (to partner LU)         HS               RM                 PS            TP
                             .                .                  .              .
                             .                .                  .              .
                             .                .                  .              .
                             .                .                  .              .
                             .                .                  . RECEIVE_AND_WAIT .
                             .                .                  o<- - - - - - - - -o
                             .                .                  RC=OK,WHAT_RECEIVED=
      OIC,RQE2|3,CD,data     .  MU(data,PREPARE_TO_RCV_CONFIRM)  .     DATA_COMPLETE
 (1)  ---------------------->o---------------------------------->o- - - - - - - - ->o
                             .                .                  . RECEIVE_AND_WAIT |
                             .                .                  o<- - - - - - - - -┘
                             .                .                  RC=OK,WHAT_RECEIVED=
                             .                .                  |     CONFIRM_SEND .
                             .                .                  └- - - - - - - - ->o
                             .            CONFIRMED              .  CONFIRMED       |
                             o<----------------------------------o<- - - - - - - - -┘
                             .                .                  |  RC=OK           .
                             .                .                  └- - - - - - - - ->o
           FIC,data          .     MU(data,NOT_END_OF_DATA)      .  SEND_DATA       |
 (2)  <----------------------o<----------------------------------o<- - - - - - - - -┘
                             .                .                  |  RC=OK           .
                             .                .                  └- - - - - - - - ->o
```

Figure 2-72. CONFIRM (RQE2|3)--Remote LU

```
TP                  PS                  RM                 HS              (to partner LU)
.                   .                   .                  .
. SEND_DATA         .                   .                  .
o- - - - - - - - - >o                   .                  .
. RC=OK             |                   .                  .
o<- - - - - - - - - -                   .                  .
|PREPARE_TO_RECEIVE                     .                  .
|(TYPE=CONFIRM      .                   .                  .
| LOCKS=LONG)       .MU(data,PREPARE_TO_RCV_CONFIRM_LONG). OIC,RQE2|3,CD,data
 - - - - - - - - - >o-------------------------------------->o----------------------------> (1)
.                   .                   .                  .
.                   .                   .                  .
.                   .                   .                  .
.                   .                   .                  .
.                   .                   .                  .
.                   .         MU(RECEIVE_ERROR)            .          -RSP(0846)
.                   o<--------------------------------------o<---------------------------- (2)
.                   .                   .                  .
. RC='derived       .                   .                  .
.      from FMH-7'  .  MU(FMH-7,data,NOT_END_OF_DATA)      .      FIC,FMH-7,data
o<- - - - - - - - - -o<-------------------------------------o<---------------------------- (3)
```

Figure 2-73. CONFIRM (RQE2|3), SEND_ERROR--Local LU

```
       (to partner LU)              HS          RM           PS          TP
                                    .           .            .            .
                                    .           .            .            .
                                    .           .            .            .
                                    .           .            .            .
                                    .           .            . RECEIVE_AND_WAIT .
                                    .           .            o<- - - - - - - - -o
                                    .           .            RC=OK,WHAT_RECEIVED=
        OIC,RQE2|3,CD,data          .  MU(data,PREPARE_TO_RCV_CONFIRM)  .    DATA_COMPLETE
(1)   ------------------------------>o---------------------------->o- - - - - - - - ->o
                                    .           .            . RECEIVE_AND_WAIT |
                                    .           .            o<- - - - - - - - -┘
                                    .           .            RC=OK,WHAT_RECEIVED=
                                    .           .            |       CONFIRM_SEND
                                    .           .            └- - - - - - - - ->o
                 -RSP(0846)         .      SEND_ERROR        .    SEND_ERROR    |
(2)   <------------------------------o<----------------------------o<- - - - - - - -┘
                                    .           .            |   RC=OK          .
                                    .           .            └- - - - - - - - ->o
             FIC,FMH-7,data[1]      .  MU(FMH-7,data[1],NOT_END_OF_DATA)  .  SEND_DATA  |
(3)   <------------------------------o<----------------------------o<- - - - - - - -┘
                                    .           .            |   RC=OK          .
                                    .           .            └- - - - - - - - ->o
```

Note:
[1] The data consists of optional log data from the SEND_ERROR verb and the TP data from the SEND_DATA verb or the data from the SEND_DATA verb alone.

Figure 2-74. CONFIRM (RQE2|3), SEND_ERROR--Remote LU

```
TP                   PS                   RM                   HS             (to partner LU)
.                    .                    .                    .
. SEND_DATA          .                    .                    .
o- - - - - - - - - ->o                    .                    .
. RC=OK              |                    .                    .
o<- - - - - - - - - -┘                    .                    .
| SEND_DATA          .                    .                    .
└ - - - - - - - - - >o                    .                    .
. RC=OK              |                    .                    .
o<- - - - - - - - - -┘                    .                    .
| CONFIRM            .         MU(data,CONFIRM)                .  OIC,RQD2|3,¬CD,data
└ - - - - - - - - - >o-----------------------------------------> o---------------------------> (1)
.                    .                    .                    .
.                    .                    .                    .
.                    .                    .                    .
.                    .                    .                    .
.                    .                    .                    .
.                    .         MU(RECEIVE_ERROR)               .            -RSP(0846)
.                    o<----------------------------------------o<---------------------------- (2)
.                    .                    .                    .
. RC='derived        .                    .                    .
.      from FMH-7'   .    MU(FMH-7,data,NOT_END_OF_DATA)       .       FIC,FMH-7,data
o<- - - - - - - - - -o<----------------------------------------o<---------------------------- (3)
```

Figure 2-75. CONFIRM (RQD2|3), SEND_ERROR--Local LU

```
         (to partner LU)                HS                  RM                   PS                TP
                                        .                   .                    .                  .
                                        .                   .                    .                  .
                                        .                   .                    .                  .
                                        .                   .                    .                  .
                                        .                   .                    .                  .
                                        .                   .                    .                  .
                                        .                   .                    . RECEIVE_AND_WAIT .
                                        .                   .                    o<- - - - - - - - -o
                                        .                   .                    RC=OK,WHAT_RECEIVED=
           OIC,RQD2|3,¬CD,data          .          MU(data,CONFIRM)              .    DATA_INCOMPLETE
(1)  ---------------------------------->o------------------------------------->o- - - - - - - - ->o
                                        .                   .                    . RECEIVE_AND_WAIT |
                                        .                   .                    o<- - - - - - - - -┘
                                        .                   .                    |RC=OK,WHAT_RECEIVED=
                                        .                   .                    |            CONFIRM
                                        .                   .                    └- - - - - - - - ->o
                      -RSP(0846)        .             SEND_ERROR                 .    SEND_ERROR    |
(2)  <----------------------------------o<-------------------------------------o<- - - - - - - - -┘
                                        .                   .                    |  RC=OK           .
                                        .                   .                    └- - - - - - - - ->o
                 FIC,FMH-7,data         .  MU(FMH-7,data,NOT_END_OF_DATA)        .    SEND_DATA     |
(3)  <----------------------------------o<-------------------------------------o<- - - - - - - - -┘
                                        .                   .                    |  RC=OK           .
                                        .                   .                    └- - - - - - - - ->o
```

Figure 2-76. CONFIRM (RQD2|3), SEND_ERROR--Remote LU

```
TP                      PS                      RM                      HS                  (to partner LU)
.                       .                       .                       .
. SEND_DATA             .                       .                       .
o- - - - - - - - - - ->o                        .                       .
. RC=OK                 |                       .                       .
o<- - - - - - - - - - --┘                       .                       .
|DEALLOCATE(CONFIRM)     MU(data,DEALLOCATE_CONFIRM)                    .  EC,RQD2|3,CEB,data
└- - - - - - - - - - ->o----------------------------------------------->o-------------------------------> (1)
.                       .                       .                       .
.                       .                       .                       .
.                       .                       .                       .
.                       .                       .                       .
.                       .                       .                       .
.                       .          MU(RECEIVE_ERROR)                    .            -RSP(0846)
.                       o<----------------------------------------------o<------------------------------ (2)
.                       .                       .                       .
. RC='derived           .                       .                       .
.       from FMH-7'     .   MU(FMH-7,data,NOT_END_OF_DATA)              .         FIC,FMH-7,data
o<- - - - - - - - - - -o<-----------------------------------------------o<------------------------------ (3)
```

Figure 2-77. DEALLOCATE(TYPE=CONFIRM), SEND_ERROR--Local LU

```
    (to partner LU)                   HS                 RM                  PS                  TP
                                      .                  .                   .                   .
                                      .                  .                   .                   .
                                      .                  .                   . RECEIVE_AND_WAIT  .
                                      .                  .                   o<- - - - - - - - - -o
                                      .                  .                   RC=OK,WHAT_RECEIVED=
      EC,RQD2|3,CEB,data              .   MU(data,DEALLOCATE_CONFIRM)        .     DATA_COMPLETE
(1) -------------------------------->o---------------------------------->o- - - - - - - - - ->o
                                      .                  .                   . RECEIVE_AND_WAIT |
                                      .                  .                   o<- - - - - - - - --
                                      .                  .                   RC=OK,WHAT_RECEIVED=
                                      .                  .                   | CONFIRM_DEALLOCATE
                                      .                  .                   L- - - - - - - - - ->o
                 -RSP(0846)           .     SEND_ERROR                       .    SEND_ERROR    |
(2) <--------------------------------o<--------------------------------------o<- - - - - - - - --
                                      .                  .                   |    RC=OK          .
                                      .                  .                   L- - - - - - - - - ->o
            FIC,FMH-7,data            .  MU(FMH-7,data,NOT_END_OF_DATA)      .    SEND_DATA     |
(3) <--------------------------------o<--------------------------------------o<- - - - - - - - --
                                      .                  .                   |    RC=OK          .
                                      .                  .                   L- - - - - - - - - ->o
```

Figure 2-78. DEALLOCATE(TYPE=CONFIRM), SEND_ERROR--Remote LU

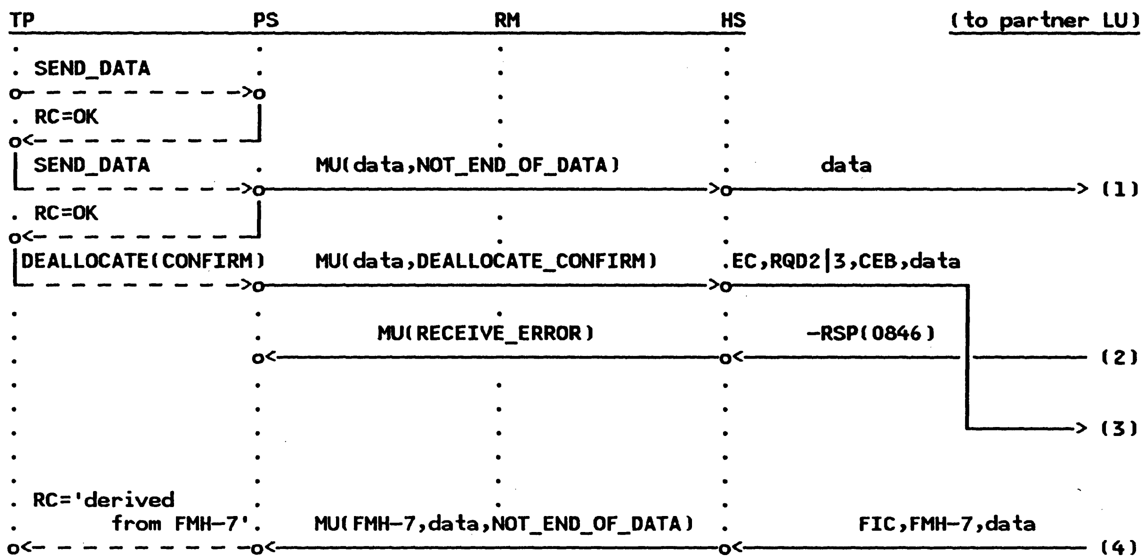


Figure 2-79. DEALLOCATE(TYPE=CONFIRM) Crossing SEND_ERROR--Local LU

```
       (to partner LU)                HS                          RM                        PS                   TP
                                       .                           .                         .                    .
                                       .                           .                         . RECEIVE_AND_WAIT  .
                                       .                           .                         o<- - - - - - - - - -o
                                       .                           .                         .                    .
                                       .                           .                         RC=OK,WHAT_RECEIVED=
               data                    .        MU(data,NOT_END_OF_DATA)                     .   DATA_INCOMPLETE
(1)  --------------------------------->o------------------------------------------------->o- - - - - - - - - ->o
                                       .                           .                         .                    .
                                       .                           .                         .                    .
                                       .                           .                         .                    .
                                       .                           .                         .                    .
                                       .                           .                         .                    .
            -RSP(0846)                 .            SEND_ERROR                               .    SEND_ERROR      .
(2)  <---------------------------------o<------------------------------------------------o<- - - - - - - - - -o
                                       .                           .                         .                    .
        EC,RQD2|3,CEB,data             .     MU(data,DEALLOCATE_CONFIRM)                     .                    .
(3)  --------------------------------->o------------------------------------------------->o                    .
                                       .                           .                (data purged)                 .
                                       .                           .                         |    RC=OK           .
                                       .                           .                         L- - - - - - - - - ->o
            FIC,FMH-7,data             .     MU(FMH-7,data,NOT_END_OF_DATA)                  .    SEND_DATA       |
(4)  <---------------------------------o<------------------------------------------------o<- - - - - - - - - -J
                                       .                           .                         |    RC=OK           .
                                       .                           .                         L- - - - - - - - - ->o
```

Figure 2-80. DEALLOCATE(TYPE=CONFIRM) Crossing SEND_ERROR--Remote LU

```
TP                  PS                   RM                     HS                (to partner LU)
.                   .                    .                      .
.   SEND_DATA       .                    .                      .
o- - - - - - - - - >o                    .                      .
.   RC=OK           |                    .                      .
o<- - - - - - - - --┘                    .                      .
|   SEND_DATA       .     MU(data,NOT_END_OF_DATA)              .          FIC,data
└ - - - - - - - - - >o-------------------------------------------->o-------------------------------> (1)
.   RC=OK           |                    .                      .
o<- - - - - - - - --┘                    .                      .
| RECEIVE_AND_WAIT  .   MU(data,PREPARE_TO_RCV_FLUSH)           .          LIC,CD,RQE1,data
└ - - - - - - - - - >o-------------------------------------------->o-------------------------------> (2)
```

Figure 2-81. RECEIVE_AND_WAIT Causing RQE,CD--Local LU

```
       (to partner LU)          HS              RM                PS                TP
                                .               .                 .                 .
                                .               .                 .                 .
                                .               .                 . RECEIVE_AND_WAIT .
                                .               .                 o<- - - - - - - - -o
                                .               .                 RC=OK,WHAT_RECEIVED=
          FIC,data              .     MU(data,NOT_END_OF_DATA)    .   DATA_INCOMPLETE
(1)  ---------------------------->o-------------------------------->o- - - - - - - - ->o
                                .               .                 .                 |
                                .               .                 .                 |
          LIC,CD,RQE1,data      .     MU(data,PREPARE_TO_RCV_FLUSH) .               |
(2)  ---------------------------->o-------------------------------->o                 |
                                .               .                 . RECEIVE_AND_WAIT |
                                .               .                 o<- - - - - - - - -+
                                .               .                 RC=OK,WHAT_RECEIVED=
                                .               .                 |     DATA_COMPLETE
                                .               .                 +- - - - - - - - ->o
                                .               .                 . RECEIVE_AND_WAIT |
                                .               .                 o<- - - - - - - - -+
                                .               .                 |RC=OK,            .
                                .               .                 |WHAT_RECEIVED=SEND
                                .               .                 +- - - - - - - - ->o
```

Figure 2-82. RECEIVE_AND_WAIT Causing RQE,CD--Remote LU

```
TP                   PS                  RM                  HS              (to partner LU)
 .                    .                   .                   .
 .  SEND_DATA         .                   .                   .
 o- - - - - - - - - ->o                   .                   .
 .  RC=OK             |                   .                   .
 o<- - - - - - - - - -┘                   .                   .
 .  SEND_DATA         .   MU(data,NOT_END_OF_DATA)            .FIC,data
 o- - - - - - - - - ->o-------------------------------------->o-------------------------> (1)
 .  RC=OK             |                   .                   .
 o<- - - - - - - - - -┘                   .                   .
 |                    .                   .                   .
 |                    .                   .                   .
 |  SEND_DATA         .   MU(data,NOT_END_OF_DATA)            .MIC,data
 └ - - - - - - - - - >o-------------------------------------->o----------------┐
 .   RC=OK            |                   .                   .                |
 o<- - - - - - - - - -┘                   .                   .                |
 |                    .   MU(RECEIVE_ERROR)                   .                |-RSP(0846)
 |                    o<--------------------------------------o<---------------|----------- (2)
 |                    .                   .                   .                |
 |                    .                   .                   .                └----------> (3)
 |                    .                   .                   .
 |  SEND_DATA         .   MU(PREPARE_TO_RCV_FLUSH)            .  LIC,CD,RQE1,no data
 └ - - - - - - - - - >o-------------------------------------->o-------------------------> (4)
 .              (discard data)            .                   .
 .                    .                   .                   .
 . RC=PROG_ERROR_     .                   .                   .
 .      PURGING       .  MU(FMH-7,data,NOT_END_OF_DATA)       .         FIC,FMH-7,data
 o<- - - - - - - - - -o<--------------------------------------o<------------------------o (5)
 |RECEIVE_AND_WAIT    .                   .                   .
 └ - - - - - - - - - >o                   .                   .
 RC=OK,WHAT_RECEIVED=                     .                   .
 .  DATA_INCOMPLETE   |                   .                   .
 o<- - - - - - - - - -┘                   .                   .
```

Figure 2-83. SEND_ERROR before SEND_DATA--Remote LU

```
     (to partner LU)                HS                 RM                  PS                  TP
                                    .                  .                   .                   .
                                    .                  .                   .  RECEIVE_AND_WAIT .
                                    .                  .                   o<- - - - - - - - - o
                                    .                  .                   RC=OK,WHAT_RECEIVED=
       FIC,data                     .  MU(data,NOT_END_OF_DATA)            .    DATA_*COMPLETE
(1)    ----------------------------->o------------------------------------>o- - - - - - - - ->o
                                    .                  .                   .                   |
                                    .                  .                   .                   |
                                    .                  .                   .                   |
                                    .                  .                   .                   |
                                    .                  .                   .                   |
                                    .                  .                   .                   |
              -RSP(0846)            .            SEND_ERROR                .   SEND_ERROR      |
(2)    <-----------------------------o<------------------------------------o<- - - - - - - - --
         MIC,data                   .  MU(data,NOT_END_OF_DATA)            .                   .
(3)    ----------------------------->o------------------------------------>o purged            .
                                    .                  .                   .                   .
         LIC,CD,RQE1,no data        .  MU(PREPARE_TO_RCV_FLUSH)            .   RC=OK           .
(4)    ----------------------------->o------------------------------------>o- - - - - - - - ->o
                                    .                  .                   .                   |
                                    .                  .                   .                   |
              FIC,FMH-7,data        .  MU(FMH-7,data,NOT_END_OF_DATA)      .   SEND_DATA       |
(5)    <-----------------------------o<------------------------------------o<- - - - - - - - --
                                    .                  .                   |   RC=OK           .
                                    .                  .                   L- - - - - - - - - >o
```

Figure 2-84. SEND_ERROR before SEND_DATA--Local LU

```
TP                  PS                  RM                  HS              (to partner LU)
.                   .                   .                   .
.   SEND_DATA       .                   .                   .          -RSP(0846)
o- - - - - - - - - >o                   .                   .   ┌──────────────────── (1)
.   RC=OK          ┘                    .                   .   │
o<- - - - - - - - -┘                    .                   .   │
┌PREPARE_TO_RECEIVE                     .                   .   │
│ (TYPE=FLUSH)         MU(data,PREPARE_TO_RCV_FLUSH)        .   │ LIC,RQE1,CD,data
└- - - - - - - - - >o──────────────────────────────────────>o───┼──────────────────> (2)
.   RC=OK          ┐                    .                   .   │
o<- - - - - - - - -┘                    .                   .   │
│   SEND_ERROR      .              SEND_ERROR               .   │
└- - - - - - - - - >o──────────────────────────────────────>o   │
.                   .            MU(RECEIVE_ERROR)          .   │
.                   o<──────────────────────────────────────o<──┘
.               (purged)                .                   .
.                   .             MU(FMH-7,data)            .        FIC,FMH-7,data
.                   o<──────────────────────────────────────o<──────────────────── (3)
.               (purged)                .                   │
.                   .                   .                   │
.                   .                   .                   │ -RSP(0846)
.                   .                   .                   └──────────────────> (4)
.                   .                   .                   .
.   RC=OK           .        MU(PREPARE_TO_RCV_FLUSH)       .     LIC,RQE1,CD,no data
o<- - - - - - - - -o<───────────────────────────────────────o<──────────────────── (5)
│                   .                   .                   .
│                   .                   .                   .
│   SEND_DATA       .    MU(FMH-7,data,NOT_END_OF_DATA)     .  FIC,FMH-7,data
└- - - - - - - - - >o──────────────────────────────────────>o──────────────────> (6)
.   RC=OK          ┐                    .                   .
o<- - - - - - - - -┘                    .                   .
```

Figure 2-85. SEND_ERROR Crossing SEND_ERROR, Both Issued in RCV_STATE--Remote LU

```
        (to partner LU)                   HS                  RM                    PS                 TP
                                           .                   .                    .                   .
           -RSP(0846)                      .              SEND_ERROR                .    SEND_ERROR     .
 (1)   <-----------------------------------o<-----------------------------------------o<- - - - - - - - -o
                                           .                   .                    .                   .
                                           .                   .                    .                   .
                                           .                   .                    .                   .
       LIC,RQE1,CD,data                    .    MU(data,PREPARE_TO_RCV_FLUSH)        .    RC=OK          .
 (2)   ---------------------------------->o------------------------------------------>o- - - - - - - - ->o
                                           .                   .                (purge)                 |
                                           .                   .                    .                   |
                                           .                   .                    .                   |
                                           .                   .                    .                   |
                                           .                   .                    .                   |
                                           .                   .                    .                   |
                                           .                   .                    .                   |
                   FIC,FMH-7,data          .    MU(FMH-7,data,NOT_END_OF_DATA)       .    SEND_DATA      |
 (3)   <-----------------------------------o<-----------------------------------------o<- - - - - - - - -
                                           .                   .                    |    RC=OK          .
                                           .                   .                    L- - - - - - - - - ->o
           -RSP(0846)                      .  MU(RECEIVE_ERROR)                     .                   .
 (4)   ---------------------------------->o------------------------------------------>o                  .
                                           .                   .                    .                   .
              LIC,CD,RQE1,no data          .     MU(PREPARE_TO_RCV_FLUSH)           .    SEND_DATA      .
 (5)   <-----------------------------------o<-----------------------------------------o<- - - - - - - - -o
                                           .                   .            (discard data)              .
                                           .                   .                    .                   .
                                           .                   .                    .RC=                .
          FIC,FMH-7,data                   .  MU(FMH-7,data,NOT_END_OF_DATA)         .PROG_ERROR_PURGING.
 (6)   o---------------------------------->o------------------------------------------>o- - - - - - - - ->o
```

Figure 2-86. SEND_ERROR Crossing SEND_ERROR, Both Issued in RCV_STATE--Local LU

```
TP                    PS                  RM                   HS              (to partner LU)
.                     .                   .                    .
.                     .                   .                    .
.                     .                   .                    .
.   SEND_DATA         .      MU(data,NOT_END_OF_DATA)          .   FIC,data
o- - - - - - - - - ->o----------------------------------------->o---------------------------> (1)
.   RC=OK            |                    .                    .
o<- - - - - - - - - -┘                    .                    .
|                     .                   .                    .
|                     .                   .                    .
|                     .        MU(RECEIVE_ERROR)               .            -RSP(0846)
|                    o<-----------------------------------------o<--------------------------- (2)
|                     .                   .                    .
|   CONFIRM           .      MU(PREPARE_TO_RCV_FLUSH)          .   LIC,CD,RQE1,no data
└- - - - - - - - - ->o----------------------------------------->o---------------------------> (3)
.             (purge data)                .                    .
.                     .                   .                    .
. RC='derived         .                   .                    .
.      from FMH-7'    . MU(FMH-7,data,PREPARE_TO_RCV_FLUSH).      OIC,CD,RQE1,FMH-7
o<- - - - - - - - - -o<-----------------------------------------o<--------------------------- (4)
| RECEIVE_AND_WAIT    .                   .                    .
└- - - - - - - - - ->o                    .                    .
 RC=OK,              |                    .                    .
 WHAT_RECEIVED=SEND  |                    .                    .
o<- - - - - - - - - -┘                    .                    .
```

Figure 2-87. SEND_ERROR before CONFIRM--Remote LU

```
    (to partner LU)                 HS                RM                   PS                TP
                                    .                 .                    .                  .
                                    .                 .                    . RECEIVE_AND_WAIT .
                                    .                 .                    o<- - - - - - - - -o
        FIC,data                    .   MU(data,NOT_END_OF_DATA)           RC=OK,WHAT_RECEIVED=
(1)  ------------------------------>o------------------------------------->┐   DATA_INCOMPLETE
                                    .                 .                    └ - - - - - - - - ->o
                                    .                 .                    .                  |
                                    .                 .                    .                  |
                                    .                 .                    .                  |
                    -RSP(0846)      .            SEND_ERROR                .   SEND_ERROR     |
(2)  <------------------------------o<-------------------------------------o<- - - - - - - - -┘
                                    .                 .                    .                  .
        LIC,CD,RQE1,no data         .      MU(PREPARE_TO_RCV_FLUSH)        .   RC=OK          .
(3)  ------------------------------>o------------------------------------->o- - - - - - - - ->o
                                    .                 .                    .                  |
                                    .                 .                    .                  |
                                    .                 .                    .                  |
              OIC,CD,RQE1,FMH-7     . MU(FMH-7,data,PREPARE_TO_RCV_FLUSH). RECEIVE_AND_WAIT   |
(4)  <------------------------------o<-------------------------------------o<- - - - - - - - -┘
```

Figure 2-88. SEND_ERROR before CONFIRM--Local LU

```
TP                 PS                  RM                 HS              (to partner LU)
.                  .                   .                  .
.                  .                   .                  .
.                  .                   .                  .
.   SEND_DATA      .    MU(data,NOT_END_OF_DATA)          .   FIC,data
o- - - - - - - - ->o------------------------------------->o---------------------------> (1)
.   RC=OK          |                   .                  .
o<- - - - - - - - -┘                   .                  .
|                  .                   .                  .
|                  .                   .                  .
|                  .       MU(RECEIVE_ERROR)              .            -RSP(0846)
|                  o<-------------------------------------o<-------------------------- (2)
|                  .                   .                  .
|DEALLOCATE(CONFIRM)    MU(PREPARE_TO_RCV_FLUSH)          .    LIC,CD,RQE1,no data
└- - - - - - - - ->o------------------------------------->o---------------------------> (3)
.            (purge data)              .                  .
.                  .                   .                  .
. RC='derived      .                   .                  .
.      from FMH-7' .  MU(FMH-7,PREPARE_TO_RCV_FLUSH)      .       OIC,CD,RQE1,FMH-7
o<- - - - - - - - -o<-------------------------------------o<-------------------------- (4)
| RECEIVE_AND_WAIT .                   .                  .
└- - - - - - - - ->o                   .                  .
 RC=OK,            |                   .                  .
 WHAT_RECEIVED=SEND|                   .                  .
o<- - - - - - - - -┘                   .                  .
```

Figure 2-89. SEND_ERROR Before DEALLOCATE(TYPE=CONFIRM)--Remote LU

```
      (to partner LU)                 HS                RM                  PS                 TP
                                      .                  .                  .                  .
                                      .                  .                  . RECEIVE_AND_WAIT .
                                      .                  .                  o<- - - - - - - - -o
        FIC,data                      .   MU(data,NOT_END_OF_DATA)          RC=OK,WHAT_RECEIVED=
(1)   ------------------------------->o------------------------------------>┐  DATA_INCOMPLETE
                                      .                  .                  └ - - - - - - - - ->o
                                      .                  .                  .                   |
                                      .                  .                  .                   |
                                      .                  .                  .                   |
                     -RSP(0846)       .       SEND_ERROR                    .  SEND_ERROR       |
(2)   <-------------------------------o<------------------------------------o<- - - - - - - - --┘
                                      .                  .                  .                   .
        LIC,CD,RQE1,no data           .   MU(PREPARE_TO_RCV_FLUSH)          .   RC=OK           .
(3)   ------------------------------->o------------------------------------>o- - - - - - - - - ->o
                                      .                  .                  .                   |
                                      .                  .                  .                   |
                                      .                  .                  .                   |
              OIC,CD,RQE1,FMH-7       . MU(FMH-7,PREPARE_TO_RCV_FLUSH)      . RECEIVE_AND_WAIT  |
(4)   <-------------------------------o<------------------------------------o<- - - - - - - - --┘
```

Figure 2-90. SEND_ERROR Before DEALLOCATE(TYPE=CONFIRM)--Local LU

```
TP                  PS                  RM                  HS                  (to partner LU)
.                   .                   .                   .
.  SEND_DATA        .                   .                   .
o- - - - - - - - - ->o                  .                   .
.  RC=OK            |                   .                   .
o<- - - - - - - - - -┘                  .                   .
| RECEIVE_AND_WAIT .  MU(data,PREPARE_TO_RCV_FLUSH)     . OIC,RQE1,CD,data
└- - - - - - - - - ->o-------------------------------------->o------------------------------> (1)
.                   .         MU(RECEIVE_ERROR)          .              -RSP(0846)
.                   o<--------------------------------------o<------------------------------ (2)
.                   .                   .                   .
.RC='derived        .                   .                   .
.      from FMH-7'  .  MU(FMH-7,data,NOT_END_OF_DATA)    .          FIC,FMH-7,data
o<- - - - - - - - - -o<--------------------------------------o<------------------------------ (3)
```

Figure 2-91. SEND_ERROR at End-of-Chain--Remote LU

```
        (to partner LU)              HS           RM            PS          TP
                                     .            .             .            .
                                     .            .             .            .
                                     .            .             . RECEIVE_AND_WAIT .
                                     .            .             o<- - - - - - - - -o
                                     .            .           RC=OK,WHAT_RECEIVED=
         OIC,RQE1,CD,data            .   MU(data,PREPARE_TO_RCV_FLUSH) .   DATA_COMPLETE
(1)      --------------------------->o-------------------------->o- - - - - - - - ->o
                    -RSP(0846)       .        SEND_ERROR         .  SEND_ERROR      |
(2)      <---------------------------o<--------------------------o<- - - - - - - - -
                                     .            .             |  RC=OK             .
                                     .            .             L- - - - - - - - - ->o
                  FIC,FMH-7,data     .   MU(FMH-7,data,NOT_END_OF_DATA) .  SEND_DATA |
(3)      <---------------------------o<--------------------------o<- - - - - - - - -
                                     .            .             |  RC=OK             .
                                     .            .             L- - - - - - - - - ->o
```

Figure 2-92. SEND_ERROR at End-of-Chain--Local LU

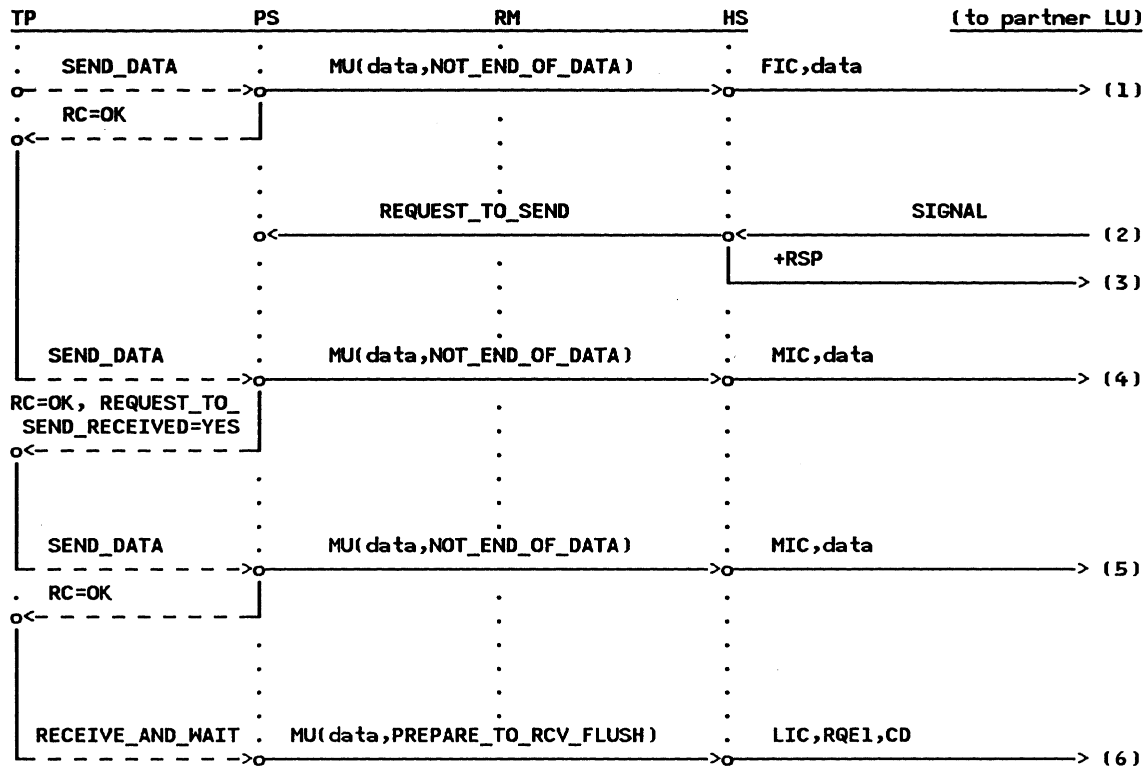


Figure 2-93. REQUEST_TO_SEND, Received in SEND_STATE--Remote LU

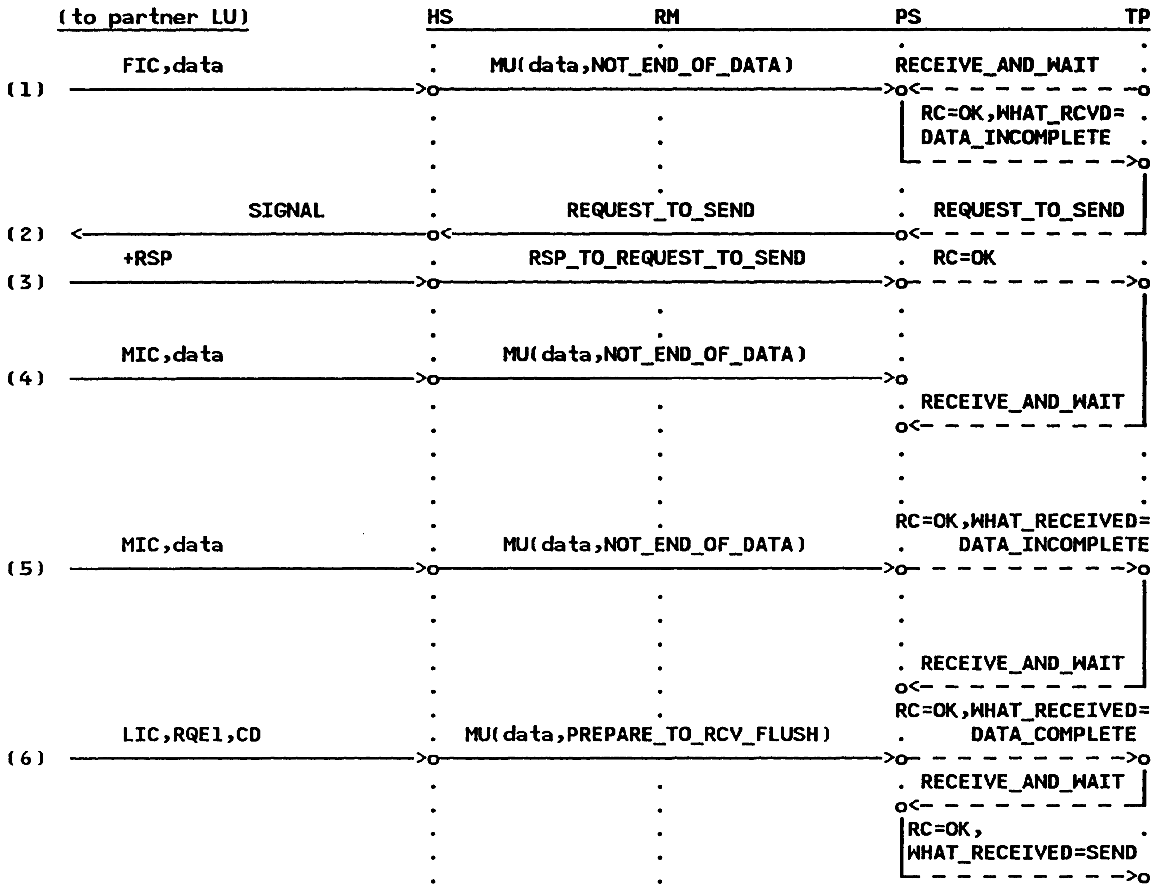


Figure 2-94. REQUEST_TO_SEND, Received in SEND_STATE--Local LU

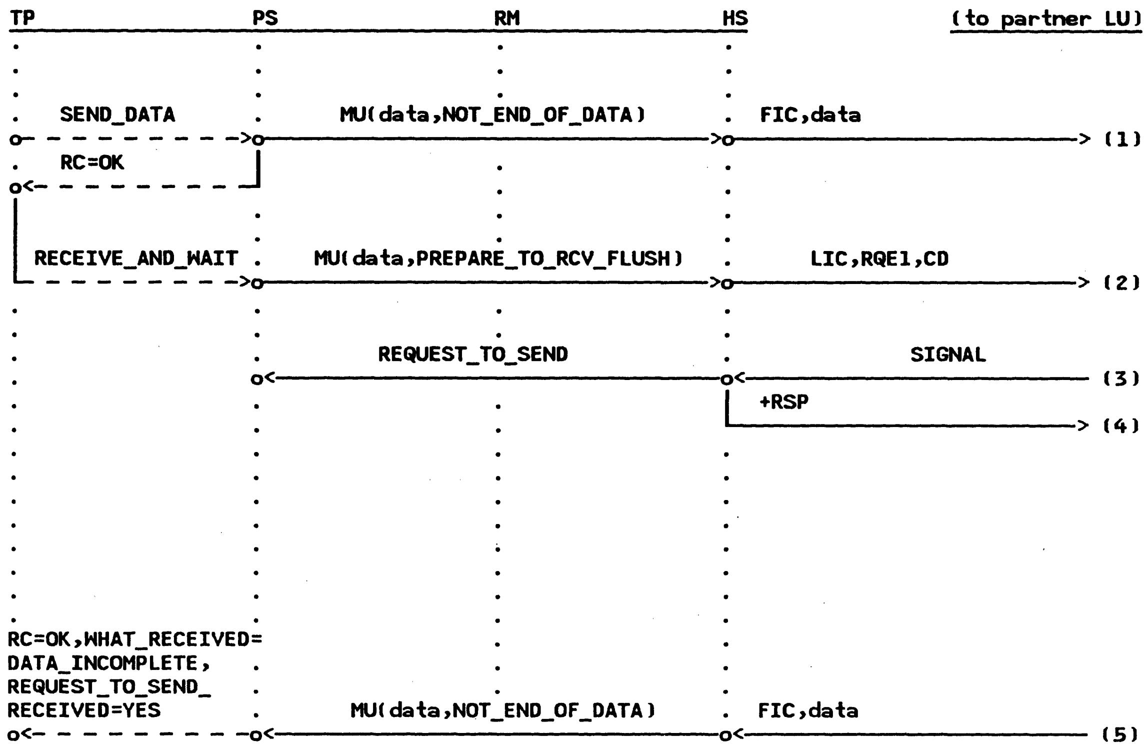


Figure 2-95. REQUEST_TO_SEND, Received in RCV_STATE--Remote LU

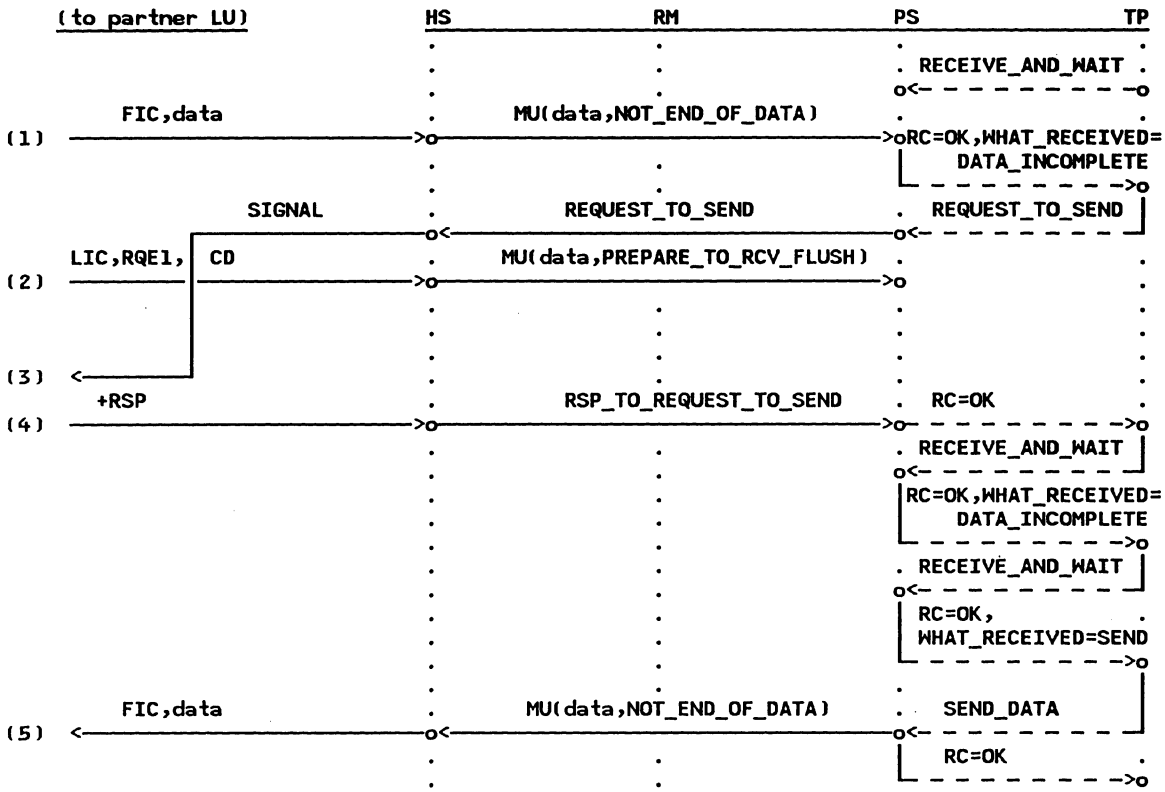


Figure 2-96. REQUEST_TO_SEND, Received in RCV_STATE--Local LU

This page intentionally left blank

CHAPTER 3. LU RESOURCES MANAGER

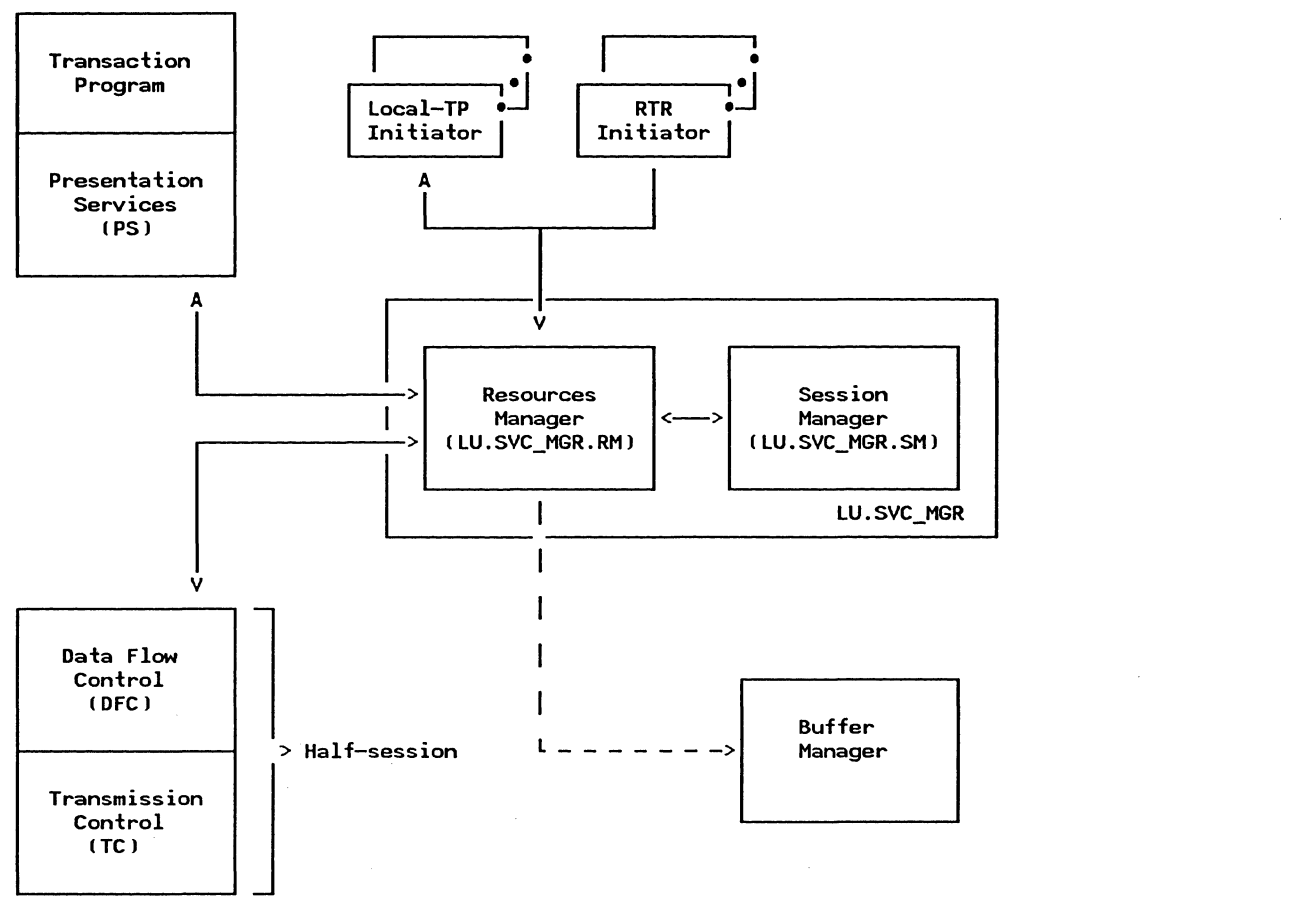


Figure 3-1. Overview of Component Interactions Involving the Resources Manager

GENERAL DESCRIPTION

When one transaction program wishes to communicate with another, the LU may activate, manage, and later deactivate a conversation. This chapter describes the management of conversation resources (or simply "conversations").

An LU contains a services manager, which in turn contains a resources manager, RM. The resources manager stores information about active transaction programs, conversations, and LU-LU sessions in control blocks, some of which are the TCB, RCB, and SCB (see "Resources Manager Data Base" on page 3-4 for additional information).

The resources manager interacts with other components in the node. These components are shown in Figure 3-1. They are PS ("Chapter 5.0. Overview of Presentation Services" and "Chapter 5.1. Presentation Services--Conversation Verbs"), SM ("Chapter 4. LU Session Manager"), HS ("Chapter 6.0. Half-Session"), BM ("Appendix B. Buffer Manager"), transaction-program-initiating process and Ready-to-Receive initiating process.

RESOURCES MANAGER FUNCTIONS

The resources manager (RM) coordinates the following functions:

- Creating new instances, and destroying existing instances, of presentation services
- Attaching new instances, and destroying existing instances, of transaction programs
- Activating and deactivating conversations
- Choosing sessions to be used by a conversation and, if necessary, requesting (bidding for) use of the session
- Requesting the session manager (SM) to activate a new session or to deactivate an existing session
- Coordinating normal cessation of conversation assignments to a particular session targeted for deactivation (using BRACKET INITIATION STOPPED--BIS protocols)
- Completing LU-LU verification (FMH-12 processing)
- Replying to requests (bids) for use of a session that are received from remote resources managers
- Providing services for support of the sync point log (the content and use of which is described in "Chapter 5.3. Presentation Services--Sync Point Services Verbs")--these services are not formally defined in this book
- Coordinating and managing conversation-level security

LU COMPONENT INTERACTIONS

Other components in the LU with which the resources manager interacts are the presentation services (PS) component associated with each transaction program instance attached to the LU, each half-session (HS) that is available for use by the resources manager, and the session manager (SM). Examples of the type of interactions that take place are given below.

When presentation services is requested by its transaction program (TP) to initiate a conversation with another TP, it requests the resources manager to assist in the request. The resources manager is responsible for such tasks as choosing a session on which to initiate the conversation; checking that the synchronization level and security level on the request correspond to what the target LU supports for this LU; and performing other functions necessary for acquiring a session for use by the requested conversation, such as creating the appropriate control blocks (see "Resources Manager Data Base" on page 3-4 for more on control blocks). After the resources manager has completed processing of the request that it received from presentation services, it sends a reply to PS informing it of the outcome of the request.

One type of unsolicited information that the resources manager sends to presentation services is an Attach FM header (FMH-5). When the resources manager receives an Attach from a remote LU via one of its half-sessions, it checks certain fields, including all security fields, carried in the Attach. Depending upon the installation-defined limit on the number of TP instances for the target transaction program (instance limit, see TRANSACTION_PROGRAM on page A-5), RM does one of two things: If the number of instances of the target transaction program has not yet reached its limit, RM creates a new instance of presentation services and sends the Attach, along with other information, to the new PS ("Attaching a Transaction Program" on page 3-10 and "Creation and Termination of Presentation Services" on page 3-18 provide additional details). If the instance limit has been reached, RM queues the Attach request. The Attach remains queued until a target TP-PS instance sends RM notification, via a TERMINATE_PS record, of its readiness to accept another Attach request (or, if none is queued, to be destroyed).

Data that the resources manager wishes to send to another resources manager in the network is first sent to the local HS component of one of the sessions connecting the two LUs. Likewise, the resources manager receives from HS all data destined for the resources manager that comes in over a session. Examples of the kind of data that flows between the resources manager and HS are bids for the use of a session, replies to bid requests, and Attach FM headers.

When the resources manager receives a request from presentation services for a session and finds that no free sessions have the required characteristics, the resources manager sends a request to SM asking it to activate a new session. Similarly, the resources manager sends to the session manager a request that a session be deactivated upon notification by PC.COPR ("Chapter 5.4. Presentation Services--Control-Operator Verbs") that too many sessions are active. SM replies to the resources manager after it has carried out the requested function. See "Activating a New Session" on page 3-15 and "Changing the Maximum Session Limit" on page 3-16 for more details on session activation and deactivation.

Other components in the node, outside of the LU, with which the resources manager interacts are the buffer manager, local-TP initiator, and RTR initiator.

The primary objective of node buffer management is to provide storage, allocation, and management for session-level pacing and to avoid unnecessary data movement from one buffer to another.

For most of its work, RM uses transient storage, not managed by the node buffer manager, that is used for records that are local to the node and not sent outside the node. This transient storage is short-lived storage that is implicitly allocated by the creation of local records and freed when the records are destroyed. Node buffer management does not manage such transient storage.

Incoming message units that may be queued for extended periods of time before being processed use storage managed by the buffer manager. FMH-5 records may be queued for an instance-limited TP for an indefinite period of time. (For more information on instance-limited TPs refer to "Attaching a Transaction Program" on page 3-10.) FMH-7 records may be queued for a TP that is not receiving. Storage for FMH records is managed by the buffer manager.

```
                           Resources                         Buffer
HS                          Manager                          Manager
.                              .                                .
.                              .                                .
.         MU(FMH-5)            .                                .
o---------------------------->o                                 .
.                              .                                .
.                     (Format or protocol                       .
.                       error detected)                         .
.                              .                                .
.                              .   FREE_BUFFER(MU(FMH-5))       .
.                              o- - - - - - - - - - - - - - - ->o
```

Figure 3-2. Buffer Management for FMH-5 MU

```
                           Resources                         Buffer
HS                          Manager                          Manager
.                              .                                .
.                              .                                .
.         MU(FMH-12)           .                                .
o---------------------------->o                                 .
.                              .                                .
.                     (Process FMH-12.)                         .
.                              .                                .
.                              .   FREE_BUFFER(MU(FMH-12))      .
.                              o- - - - - - - - - - - - - - - ->o
```

Figure 3-3. Buffer Management for FMH-12 MU

When RM receives an FMH-5 record, it is contained in an MU. Normally, RM sends the FMH-5 MU to PS for further processing, but if RM detects a format or protocol error in the FMH-5 record, it discards the record by specifying the FMH-5 MU in a FREE_BUFFER call to the buffer manager (see Figure 3-2). The FREE_BUFFER call informs the buffer manager that the storage for the discarded FMH MU is available. In the same way, when RM finishes processing FMH-12 MUs, it informs the buffer manager using FREE_BUFFER (see Figure 3-3).

Certain independent processes, called <u>initiating processes</u>, interact with RM for the purpose of starting an initial transaction program, i.e., the originator of a distributed transaction, or sending an RTR request to a partner LU, which allows the partner LU to initiate a conversation via a bid. These initiating processes include examples such as an application, a combined TP-PS process, a control-point process, the node operator facility (NOF), and RM itself. An initiating process is normally a privileged process.

The resources manager needs information about such things as the transaction programs currently attached to the LU, the conversations associated with each transaction program, and the sessions available for use by a conversation between transaction programs. This information is stored in a group of control blocks found in the LU (see "Appendix A. Node Data Structures" for the control block definitions). The resources manager initializes entries in some control blocks, while it only accesses or updates information in entries already existing in other control blocks.

CONTROL BLOCKS MAINTAINED BY THE RESOURCES MANAGER

Information about transaction programs is contained in the transaction control block (TCB). One TCB exists for each active TP-PS process associated with the LU. Each TCB contains a TCB identifier (TCB_ID), which uniquely identifies the transaction program being represented by the TCB. The TCB_ID is also used in all communication between the resources manager and presentation services servicing the transaction program. For example, when presentation services sends a record to the resources manager, it provides its TCB_ID so that the resources manager will know, of all the TP-PS processes it manages, which presentation services to send a reply to. Presentation services is informed of its TCB_ID when the TP-PS process is created by the resources manager. When the resources manager receives an Attach header (FMH-5) from a remote resources manager, it creates a new TCB, creates a new instance of presentation services to be associated with the transaction program being attached, and sends the TCB_ID of the new TCB to presentation services. Thus, attaching a transaction program results in creation of a new TP-PS process for that transaction program, with which a presentation services component is always associated.

Associated with each TCB is a group of resource control blocks (RCBs). One RCB exists in the group for each conversation associated with the transaction program. Besides the RCB_ID, an RCB contains several other pieces of information, such as the TCB_ID of the TP-PS process that is using the conversation; the LU name, mode name, and half-session identifier (HS_ID) of the session on which a conversation is running; and a field in which presentation services stores data that it receives from the transaction program.

The final control block maintained by the resources manager is the session control block (SCB). One SCB exists for each active session between the LU and a partner LU. Information contained in an SCB includes a half-session identifier (HS_ID) and the partner LU name (LU_NAME) and mode name (MODE_NAME) for the session.

CONTROL BLOCKS ACCESSED BY THE RESOURCES MANAGER

In addition to those control blocks managed by the resources manager, other control blocks exist that are managed by another component but are accessed and updated by the resources manager.

One of these control blocks is MODE. There is one MODE control block for each mode name that is defined for the particular LU. The MODE entry contains information that is fixed on a mode name basis such as session counts and session limits.

```
Transaction                        Presentation                      Resources
  Program                            Services                          Manager
     .                                  .                                 .
     .            ALLOCATE              .                                 .
     o--------------------------------->o                                 .
     .                                  .                                 .
     .                                  .          ALLOCATE_RCB           .
     .                                  o-------------------------------->o
     .                                  .                                 .
     .                                  .      RCB_ALLOCATED(RCB_ID)      .
     .                                  o<--------------------------------o
```

Figure 3-4. Allocation of a Resource Control Block (RCB)

When the resources manager receives a message unit (MU) containing an Attach from HS for a TP that has not reached its instance limit, it creates a new TCB (representing the new instance of a TP-PS process) and RCB (representing the transaction program's initial conversation). It passes the IDs of the control blocks to the newly-created presentation services process (see "Attaching a Transaction Program" on page 3-10). Once the transaction program is attached, it can initiate conversations with other transaction programs.

A TP-PS process can also be created as the result of a local request generated by a independent, initiating process running on the same system as RM. To start a transaction program locally, the initiating process creates a START_TP record (refer to page A-19). The START_TP record contains information such as the name of the TP to be started; security tokens, e.g., user ID, password, and profile; and, if a reply to the START_TP request is desired, the identification of the initiating process. The START_TP record is sent to RM via a queue also used to receive SEND_RTR records. RM treats a START_TP much like an Attach (i.e., it creates the PS process and sends it the START_TP record), except that no conversation or RCB is associated with the request, and a reply (see START_TP_REPLY on page A-20) is optionally permitted.

ALLOCATING A NEW CONVERSATION

When the transaction program is ready to start a new conversation, it issues an ALLOCATE verb to presentation services. In general, presentation services separates the ALLOCATE request into two distinct functions, i.e., allocating an RCB and obtaining a session. Presentation services requests the resources manager to create a new RCB via an ALLOCATE_RCB record. The ALLOCATE_RCB contains information about the type of session that will be needed for the conversation. RM stores the session-related information in the new RCB and sends presentation services an RCB_ALLOCATED record, which contains the ID of the RCB. See Figure 3-4 for the flows that take place.

OBTAINING A SESSION

Once presentation services (PS) is informed of the ID of the new RCB, it requests that an LU-LU session be allocated to the conversation. After RM has allocated an LU-LU session to satisfy the request from PS, PS creates an Attach FM header (FMH-5) (in a buffer obtained from the buffer manager) and places its address in the RCB. PS then returns to the transaction program. (see "Chapter 5.1. Presentation Services--Conversation Verbs" for specific details).

Presentation services asks for a session to be allocated by sending a GET_SESSION record to the resources manager. The GET_SESSION contains the RCB_ID of the conversation that is to use the session.

An LU attempts to allocate a session that it considers available. A session that is available is between brackets, is not currently in conversation, and is not in the process of being terminated. A session that is available is referred to as being <u>free</u>. The set of free sessions at an LU is referred to as the <u>free-session pool</u>. The LU removes free sessions from the free-session pool when they are needed for conversations and returns them to the free-session pool when they are available.

The resources manager at either end of a session connecting two LUs may attempt to allocate that session to a conversation. If both resources managers attempt to allocate the same session at the same time, there must be some way to resolve the contention for the session. For this reason, one of the LUs is designated the "first speaker" (or "contention winner") and the other LU is designated the "bidder" (or "contention loser") for the session. The assignment of first-speaker and bidder status is established during session activation and remains in effect for the duration of the session. If more than one session exists between a pair of LUs, one LU may be the first speaker for some sessions and the bidder for the others. If an LU is the first speaker for a particular session, that session is said to be a first-speaker session for the LU.

The resources manager in a bidder LU must request the resources manager in the first-speaker LU for permission to use a session. This is called "bidding" for a session. The first-speaker LU may either grant or deny the request for the session from the bidder LU by sending a positive or negative bid response.

There are two forms of negative bid response associated with a parallel session. They are distinguished by the sense code in the negative bid response. The first form (sense code X'0813') is the rejection of a bid with no restriction on bidding for the same session again. The second form (sense code X'0814') is the rejection of a bid with the restriction that no further bids on the session are permitted until the first-speaker LU sends a Ready-to-Receive (RTR) record. This second form of bid rejection reserves the session for the first-speaker LU's use until it is ready to receive bids again for the session. The first-speaker LU may send RTR on the reserved session whenever the session is between brackets. When the RTR is sent is implementation or installation defined. This book models an initiator interface to RM that may be used to prompt RM to send the RTR. This prompt is modeled as a SEND_RTR record that is created and sent to RM by an RTR initiating process.

When a bid is rejected, the bidding LU may try to bid on the same session (depending upon the sense code in the negative bid response) or another session that is between brackets; if no sessions are between brackets, RM will queue the session allocation request to await the freeing of a session.

If the resources manager in a first-speaker LU wishes to allocate a free session to a conversation, it may do so immediately, without requesting permission from the resources manager in the other LU.

```
                 Presentation        Resources
                   Services           Manager                   HS
                      .                  .                       .
                      .                  .                       .
                      . GET_SESSION(RCB_ID) .                    .
1                     o----------------->o                       .
                      .                  .                       .
                    [ .                  .                       .  ]
                    [ .                  .    HS_PS_CONNECTED    .  ]
4          First-   [ .                  o---------------------->o  ]
           Speaker  [ .                  .                       .  ]
           Flows    [ .  SESSION_ALLOCATED .                     .  ]
5                   [ o<-----------------o                       .  ]
                      .                  .                       .
                      .                -OR-                      .
                      .                  .                       .
                    [ .                  .                       .  ]
                    [ .                  .   BID_WITHOUT_ATTACH  .  ]
2                   [ .                  o---------------------->o  ]
                    [ .                  .                       .  ]
                    [ .                  .          .            .  ]
                    [ .                  .          .            .  ]
                    [ .                  .          .            .  ]
           Bidder   [ .                  .                       .  ]
           Flows    [ .                  .       +BID_RSP        .  ]
3                   [ .                  o<----------------------o  ]
                    [ .                  .                       .  ]
                    [ .                  .    HS_PS_CONNECTED    .  ]
4                   [ .                  o---------------------->o  ]
                    [ .                  .                       .  ]
                    [ .  SESSION_ALLOCATED .                     .  ]
5                   [ o<-----------------o                       .  ]
```

Figure 3-5. Allocation of a Session Using BID_WITHOUT_ATTACH

The resources manager will always allocate a first-speaker session in preference to a bidder session, to avoid the bidding procedure. Figure 3-5 illustrates the flows that take place when the resources manager attempts to allocate a session. The records used in the figure are defined in "Appendix A. Node Data Structures" in more detail. The following description refers to the numbers in the figure.

1. Presentation services sends a GET_SESSION record to the resources manager. The RCB_ID identifies an RCB that was previously allocated by the resources manager.

2. If no first-speaker session is available, the resources manager must bid for use of a session. It sends BID_WITHOUT_ATTACH to the half-session. The bid will flow on the session to the resources manager at the partner LU. Between the time that the bid is sent and the bid response is received, the resources manager retains enough information to be able to proceed with session allocation when the bid response arrives. This information includes saving the HS_ID of the session and the GET_SESSION record in the RCB.

3. The BID_RSP arrives from the remote resources manager via the half-session. The positive response indicates that the bid for use of the session has been accepted and the resources manager can complete the session allocation. Not shown in this figure is the processing of a -BID_RSP. In this case, the resources manager would attempt allocation of a different session, if possible.

4. An HS_PS_CONNECTED record is sent by RM to the half-session to inform the half_session that it has been connected to a TP-PS process.

5. A SESSION_ALLOCATED record is sent by RM to presentation services to inform it that a session has been allocated to the conversation, satisfying the GET_SESSION request.

```
                              Resources                   Presentation
     HS                        Manager                      Services
     .                            .                            .
     .          BID               .                            .
1    o--------------------------->o                            .
     .                            .                            .
     .                            .                            .
   [ .                            .                            .  ]
   | .        BID_RSP             .                            .  |
2a | o<---------------------------o                            .  |
   | .                            .                            .  |
   | .      MU(Attach)            .                            .  |
3  | o--------------------------->o                            .  |
   | .                            .                            .  |
   | .                            .         MU(Attach)         .  |
4  [ .                            o--------------------------->o  ]
     .                            .                            .
     .                            .                            .
     .                          -OR-                           .
     .                            .                            .
   [ .                            .                            .  ]
   | .       -BID_RSP             .                            .  |
2b [ o<---------------------------o                            .  ]
```

Figure 3-6. Responding to a Bid for a Session

Figure 3-6 illustrates the flows that take place when a Bid request is received by the resources manager. The records used in the figure are defined in "Appendix A. Node Data Structures" in more detail. The following description refers to the numbers in the figure.

1. A BID record is received from the half-session. The half-session sends a BID record to RM whenever the partner LU sends BB, regardless of whether the partner LU is bidder or first speaker.

2a. If RM responds with a +BID_RSP, the request by the remote resources manager to use the session is accepted and processing continues with receipt of the Attach FM header from the half-session (3 and 4).

2b. If RM responds with a -BID_RSP, the request by the remote resources manager to use the session is rejected.

3. A message unit (MU) that includes FMH-5(Attach) is sent from the half-session to RM.

4. RM creates a new TP-PS and sends the MU to PS. See "Attaching a Transaction Program" on page 3-10 for further details.

```
                    Presentation             Resources
                      Services                Manager                    HS
                         .                       .                        .
                         .  ALLOCATE_RCB          .                        .
                         .  (IMMEDIATE_SESSION=YES)                        .
                         o---------------------->o                        .
                         .                       .                        .
                         .                       .                        .
                      [  .                       .                        .  ]
                      |  .                       .      HS_PS_CONNECTED   .  |
   First-Speaker      |  .                       o----------------------->o  |
      Session         |  .                       .                        .  |
     Available        |  .  RCB_ALLOCATED(RCB_ID) .                        .  |
                      |  o<----------------------o                        .  |
                      [  .                       .                        .  ]
                         .                       .                        .
                         .                     -OR-                       .
                         .                       .                        .
                      [  .                       .                        .  ]
   First-Speaker      |  .  RCB_ALLOCATED        .                        .  |
      Session         |  . (RETURN_CODE = UNSUCCESSFUL) .                 .  |
   Not Available      |  o<----------------------o                        .  |
                      [                                                      ]
```

Figure 3-7. Immediate Allocation of a Session

IMMEDIATE SESSION PROCESSING

Presentation services can request the resources manager to allocate both an RCB and a session with one record. ALLOCATE_RCB(IMMEDIATE_SESSION=YES) embodies the function of both ALLOCATE_RCB and GET_SESSION in that when the processing completes successfully, both an RCB and an SCB are allocated. ALLOCATE_RCB(IMMEDIATE_SESSION=YES) instructs the resources manager to allocate an RCB only if a first-speaker half-session is currently available. If such a half-session is not available, no allocation is performed. See Figure 3-7 for the specific steps involved.

```
                              Resources                      Presentation
HS                            Manager                        Services
.                                .                                .
.                                .                                .
.        MU (Attach)             .                                .
o-------------------------------->o                               .
.                                .                                .
.      HS_PS_CONNECTED           .                                .
o<--------------------------------o                               .
.                                .                                .
.                                .  MU (Attach, TCB_ID, RCB_ID,   .
.                                .      SENSE_DATA).              .
.                                o-------------------------------->o
```

Figure 3-8. Attach Flow

ATTACHING A TRANSACTION PROGRAM

One transaction program requests via an Attach FM header (FMH-5) that another transaction program be attached to a conversation. The resources manager handles the receipt of the message unit (MU) that contains the Attach. Only one Attach is sent per conversation. RM processes the Attach and later sends it to PS_INITIALIZE in the TP-PS process for further processing.

RM is responsible for checking certain fields of the Attach, such as the transaction program name field. RM performs all security checks of the Attach. (PS_INITIALIZE later checks the remaining fields). It notifies presentation services of the result of the checking through a field in the MU that RM sends to PS.

If the Attach violates established protocol (e.g., by sending an Already Verified indication to a partner LU that does not accept it, by sending multiple passwords on a single Attach, or by indicating a synchronization level of syncpoint when the level for the session is confirm-only), RM instructs SM to generate and return an UNBIND and RM does not create a new instance of the TP-PS process. For all other errors found in the Attach (e.g., invalid user ID, invalid parameter length), PS is responsible for returning an FMH-7 or for instructing SM, via RM, to return an UNBIND. These actions notify the transaction program that initiated the Attach of the error.

If, after checking the Attach, no protocol error is found and the requested TP's instance count (number of TP-PS instances) is less than its instance limit (as defined in the TRANSACTION_PROGRAM control block), the resources manager creates a new instance of the TP-PS process; it creates a new TCB and RCB; and it connects the TP-PS process to the half-session. RM notifies the half-session, via an HS_PS_CONNECTED record, that it has been connected to a TP-PS process. Finally, RM sends the MU containing the Attach to the new instance of the TP-PS process. The MU contains the Attach FM header, the FMH-7 sense data field (if applicable), and the IDs of the new TCB and RCB. Figure 3-8 depicts the steps involved in Attach processing.

If, after checking the Attach, no protocol error is found and the TP's instance count equals or exceeds the TP's instance limit, the resources manager creates a new RCB, connects the RCB with the half-session, informs the half-session of the RCB connection, and queues the MU containing the Attach to await an instance of the requested TP to become free.

TP instances are free when all processing and conversations have been completed and the TP and its associated PS are ready to accept a new Attach or, if no Attach is queued for this TP, are ready to be destroyed. PS informs RM that it is free via the TERMINATE_PS record.

Upon receipt of the TERMINATE_PS record, RM checks for a request queued for the transaction program. If it finds a queued request, RM updates the associated TCB and sends the request to the TP-PS instance; otherwise, RM destroys the TP-PS process.

```
                    First-Speaker                          Bidder

      Resources                                                        Resources
       Manager               HS         •••        HS                   Manager
          .                   .                     .                      .
          .                   .                     .                      .
          . BID_WITHOUT_ATTACH .                    . BID_WITHOUT_ATTACH .
          o-------------------o                     o<---------------------o
          .                   . \                 / .                      .
Create RCB                    .   \             /   .     Create RCB1
RCB.HS_ID  =  HS_ID           .     \         /     .     RCB1.HS_ID  =  HS_ID
SCB.RCB_ID =  RCB_ID          .       \     /       .     SCB.RCB_ID  =  NULL
STATE(FSM_RCB_STATUS) = IN_USE .         X          .     STATE(FSM_RCB1_STATUS) = PENDING_ATTACH
STATE(FSM_SCB_STATUS) = IN_USE .       /     \      .     STATE(FSM_SCB_STATUS)  = FREE
          .                   .     /         \     .                      .
          .                   .   /             \   .                      .
          .        BID        . /                 \ .     BID              .
          o<------------------o                     o--------------------->o
          .                   .                     .                      .
          .                   .                     .     Create RCB2
          .                   .                     .     RCB2.HS_ID  =  HS_ID
          .                   .                     .     SCB.RCB_ID  =  RCB_ID
          .                   .                     .     STATE(FSM_RCB2_STATUS) = IN_USE
STATE(FSM_SCB_STATUS) = IN_USE, so:                 .     STATE(FSM_SCB_STATUS)  = IN_USE
          .                   .                     .     ATTACH is sent to PS
          .                   .                     .                      .
          .      -BID_RSP     .-RSP(BID,0813|0814).    -BID_RSP            .
          o------------------>o-------------------->o--------------------->o

                                                          RCB1.HS_ID  =  NULL
                                                          STATE(FSM_RCB1_STATUS) = FREE
                                                          Retry on another session
```

Figure 3-9. Bid Races

RACES FOR THE USE OF A SESSION

It is possible for the resources manager on each end of a session to simultaneously choose that session to service separate GET_SESSION records, causing a bid race. The resources manager on the first-speaker side of the session always wins such a bid race. When it receives the bid from the bidder RM, it recognizes that the session is already in use and generates a negative RSP(BID). When the bidder RM receives the negative RSP(BID), it checks the free-session pool to see if another session is available and retries the GET_SESSION processing on that session. Figure 3-9 illustrates an example of a bid race and shows the RCB and SCB settings that allow a race condition to be detected.

The negative RSP(BID) that is generated for a bid rejection can have a sense code of either 0813 (Bracket Bid Reject—No RTR Forthcoming) or 0814 (Bracket Bid Reject—RTR Forthcoming). Either -RSP(BID,0813) or -RSP(BID,0814) may be sent, the decision being an implementation-dependent choice.

An implementation may permit a transaction program to reserve a session before a conversation is started on that session. A bid for a reserved session is always rejected with a -RSP(BID,0814), since the transaction program might never begin a conversation on the reserved session (if, for example, the transaction program terminated abnormally). The resources manager, by sending an RTR, informs the partner LU that it can bid on the session again.

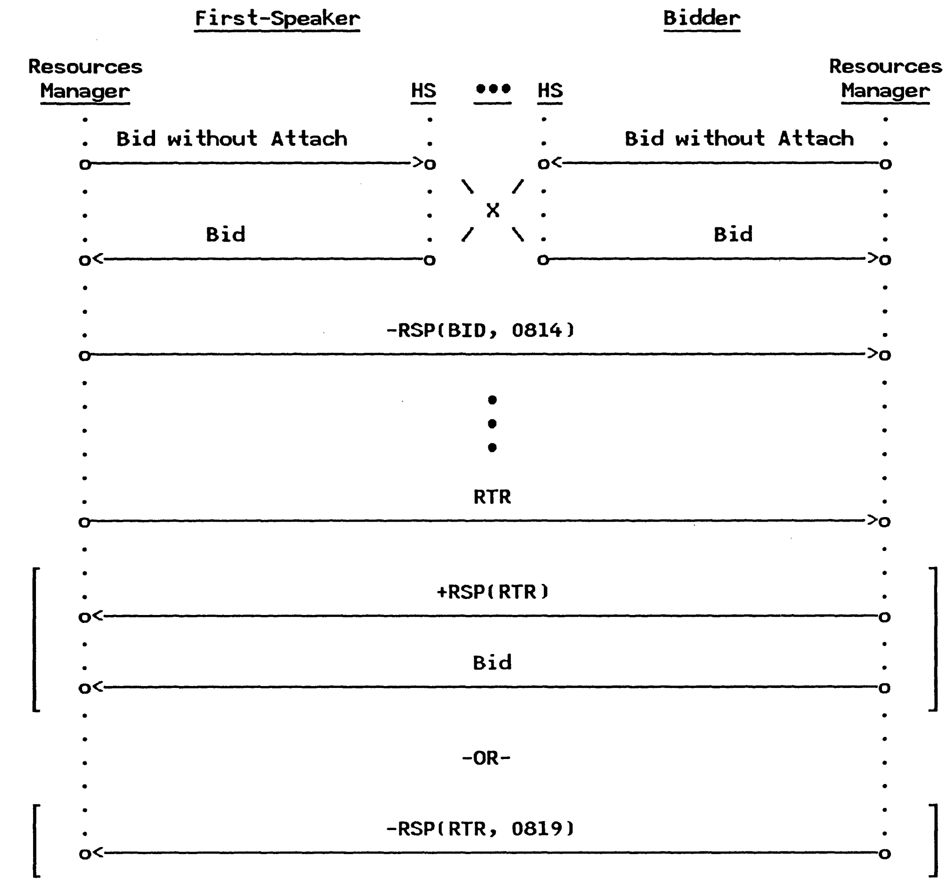


Figure 3-10. READY TO RECEIVE (RTR) Flow

Figure 3-10 depicts possible RTR flows. In the situation where there is a bid race and -RSP(BID,0814) is sent, the resources manager at the bidder side of the session cannot bid again for that session until it has received an RTR from the first-speaker RM. Upon receipt of a -RSP(BID,0814), the bidder resources manager updates a field in the SCB to remember that -RSP(BID,0814) was received and retries the Bid on another session. From this point until the RTR is received, whenever a conversation ends and the session becomes free, the session is not returned to the free session pool (as is the normal case), thereby preventing the session from being chosen for bidding.

When the current conversation ends, the first-speaker RM returns the session to the free-session pool and checks to see if any waiting requests can be satisfied by that session. The resources manager may use the session to service multiple GET_SESSION requests before sending the promised RTR.

At some implementation-defined or installation-controlled point, the resources manager at the first-speaker side sends an RTR to the resources manager at the bidder side. This is a notification to the bidder RM that it can now use the session. When the first-speaker RM sends the RTR, it removes the session from the free session pool to prevent that session from being chosen to service a request before the bidder RM has had a chance to respond to the RTR.

When the bidder RM receives the RTR, it places the session in the free session pool (for the first time since receiving the -RSP(BID,0814) to the Bid). It then checks to see if a GET_SESSION record is waiting to be serviced, if so RM then sends a +RSP(RTR) (indicating that it intends to use the session) and a Bid to the first-speaker resources manager. If no GET_SESSION records are waiting, the bidder sends a -RSP(RTR,0819). This indicates to the first-speaker RM that the bidder does not need the session. At this time, the first-speaker places the session back into the free session pool and checks for any waiting requests.

```
Presentation                              Resources
  Services                                 Manager                                    HS
       .                                      .                                        .
       .                                      .                                        .
[      .                                      .                                        .  ]
|      .  DEALLOCATE_RCB(RCB_ID)              .                                        .  |
|      o------------------------------------>o                                         .  |
|      .                                      .                                        .  |
|      .         RCB_DEALLOCATED              .                                        .  |
|      o<------------------------------------o                                         .  |
|      .                                      .                                        .  |
|      .                                      .                                        .  |
|      .                                      .                                        .  |
|      .                                      .                                        .  |
|      .                                      .          FREE_SESSION                  .  |
[      .                                      o<---------------------------------------o  ]
       .                                      .                                        .
       .                                      .                                        .
       .                                    -OR-                                       .
       .                                      .                                        .
[      .                                      .                                        .  ]
|      .                                      .          FREE_SESSION                  .  |
|      .                                      o<---------------------------------------o  |
|      .                                      .                                        .  |
|      .                                      .                                        .  |
|      .                                      .                                        .  |
|      .                                      .                                        .  |
|      .  DEALLOCATE_RCB(RCB_ID)              .                                        .  |
|      o------------------------------------>o                                         .  |
|      .                                      .                                        .  |
|      .         RCB_DEALLOCATED              .                                        .  |
[      o<------------------------------------o                                         .  ]
```

Note: DEALLOCATE_RCB and FREE_SESSION are independent records and can be sent to the resources manager in any order.

Figure 3-11. End of a Conversation

TERMINATING A CONVERSATION

After the resources manager has established a conversation between two transaction programs, it is not called upon to do any other processing for that conversation until the transaction programs are ready to end the conversation (see Figure 3-11). The resources manager is informed of the end of the conversation via two independent records.

One record is DEALLOCATE_RCB, sent from presentation services. The other is FREE_SESSION, sent from HS to inform the resources manager that the session is now available for use by another conversation. Whichever record is received first triggers the resources manager to disconnect PS and HS.

```
                  --PRIMARY--                                         --SECONDARY--

 Presentation   Resources    Session                      Network     Resources   Presentation
   Services      Manager     Manager     HS   •••   HS    Services     Manager      Services
      .             .           .         .          .        .            .
      .  GET_SESSION .          .         .          .        .            .
      o------------->o          .         .          .        .            .
      .             .           .         .          .        .            .
      .        ACTIVATE_SESSION           .          .        .            .
      .             o---------->o         .          .        .            .
      .             .           .         .    •     .        .            .
      .             .           .   (Normal BIND protocols)   .            .
      .             .           .         .    •     .        .            .
      .             .           .         .    •     .        .            .
      .        ACTIVATE_SESSION_RSP       .          .     SESSION_ACTIVATED
      .             o<----------o         .          .        o----------->o
      .             .                     .          .                     .
      .             .   RM_HS_CONNECTED   .          .   RM_HS_CONNECTED   .
      .             o-------------------->o          o<--------------------o
      .             .                     .          .                     .
   [ [ .            .                     .          .                     .        ] ]
 1 | < SESSION_ALLOCATED                  .          .                     .        ] |
   | [ o<-----------o                     .          .                     .        ] |
   |   .            .                     .          .                     .          |
   | -OR-           .                     .          .                     .          |
   | [ .            .                     .          .                     .        ] |
   | | .            .    ENCIPHERED_RD2   .          .                     .        | |
   | | .            o-------------------->o          .                     .        | |
   | | .            .                     . FMH-12   .                     .        | |
   | | .            .                     o--------->o                     .        | |
 2 | < .            .                     .          .    MU (FMH-12)      .        | |
   | | .            .                     .          o-------------------->o        | |
   | | .            .                     .          .                     .        | |
   | | SESSION_ALLOCATED                  .          .                     .        | |
   [ [ o<-----------o                     .          .                     .        ] ]
                    .                     .          .                     .
 -OR-               .                     .          .                     .
                    .                     .          .                     .
   [             [  .                     .          .                     .    ]        ]
   |             |  .    ENCIPHERED_RD2   .          .                     .    |        |
   |             |  o-------------------->o          .                     .    |        |
   |             |  .                     . FMH-12   .                     .    |        |
 3 |             <  .                     o--------->o                     .    |        |
   |             |  .                     .          .    MU (FMH-12)      .    |        |
   |             |  .                     .          o-------------------->o    |        |
   |             |  .                     .          .    FREE_SESSION     .    |        |
   |             |  .                     .          o-------------------->o    |        |
   |             [  .                     .          .                     .    ]        |
   |              -OR-                    .          .                     .             |
   |             [  .                     .          .                     .    ]        |
   |             |  .    YIELD_SESSION    .          .                     .    |        |
   |             |  o-------------------->o          .                     .    |        |
 4 |             <  .                     . LUSTAT   .                     .    |        |
   |             |  .                     o--------->o                     .    |        |
   |             |  .                     .          .    FREE_SESSION     .    |        |
   [             [  .                     .          o-------------------->o    ]        ]
```

Figure 3-12. Activation of a Session

ACTIVATING A NEW SESSION

The resources manager allocates sessions to be used by conversations. Presentation services requests the session be allocated with a GET_SESSION record. RM chooses sessions from the free session pool to satisfy the GET_SESSION request. If the pool is empty and the session limits allow the activation of a new session, the resources manager sends an ACTIVATE_SESSION record, containing the LU name and mode name of the desired session, to the session manager (SM, "Chapter 4. LU Session Manager"). Figure 3-12 on page 3-14 illustrates the steps involved in activating a session.

Although RM will not request session activation if it would cause the session limits to be exceeded, SM is ultimately responsible for checking to see that the number of active sessions is not greater than the maximum number of sessions allowed for that (LU name, mode name) pair. Some conditions (e.g., a BIND race) will cause RM to request a session activation that would exceed the session limits. In this case, the activation request from RM is rejected with a negative ACTIVATE_SESSION_RSP record.

If the session can be activated, normal BIND protocols take place. When the session has been successfully activated, the SM component sends the resources manager a positive ACTIVATE_SESSION_RSP record informing RM of the SCB_ID of the new session.

In the following discussion, the numbers in parentheses correspond to the numbers in Figure 3-12.

When a new session is activated, RM sends an RM_HS_CONNECTED record to the new half-session. This record informs the new half-session that RM is aware of its existence and is ready to accept records from it. A new session comes up in-bracket, with the resources manager on the primary side of the session having control of the session. This is true even if the resources manager on the secondary side of the session was the one that issued the ACTIVATE_SESSION record that caused the session to be activated (via INIT-SELF). Upon receipt of a positive ACTIVATE_SESSION_RSP (or SESSION_ACTIVATED in the case of activation by the partner LU), RM creates and initializes an SCB based on the information carried in the ACTIVATE_SESSION_RSP (or SESSION_ACTIVATED).

If the newly activated session is a primary half-session, RM determines if any requests are waiting to be serviced. If LU-LU verification is not active and a request is waiting (1), RM uses the new session to service the request and sends a SESSION_ALLOCATED record to presentation services. If LU-LU verification is active and a request is waiting (2), RM will generate and send to the half-session an ENCIPHERED_RD2 record containing an FMH-12. Parameters within the ENCIPHERED_RD2 record inform HS not to end the bracket nor to yield control of the session. RM then uses the new session to service the request and sends a SESSION_ALLOCATED record to presentation services. If no requests are waiting and LU-LU verification is active (3), RM will generate and send to the half-session an ENCIPHERED_RD2 record containing an FMH-12 and parameters that inform the half-session to relinquish control of the session and to end the bracket. If no requests are waiting and LU-LU verification is not active (4), RM sends a YIELD_SESSION record to HS, thus yielding its right to use the session and ending the bracket.

The resources manager at the partner LU (secondary half-session) is notified of the session activation by a SESSION_ACTIVATED record from its SM component. If LU-LU verification is active, the secondary LU's resources manager will await receipt of a message unit (MU) that contains the FMH-12. When the MU is received and verified by the secondary LU, normal processing continues.

```
     CNOS                                                                    CNOS
  Transaction      Resources      Session                        Resources   Transaction
    Program         Manager       Manager    HS   •••   HS        Manager      Program
       .               .             .        .          .           .            .
       .               .             .        .          .           .            .
       .               .      Change Number of Sessions              .            .
       o---------------------------------------------------------------------------->o
       .               .             .        .          .           .            .
       .               .      Change Number of Sessions              .            .
       o<----------------------------------------------------------------------------o
       .               .             .        .          .           .            .
  CHANGE_SESSIONS .                  .        .          .     CHANGE_SESSIONS
       . (Decrease)    .             .        .          .     . (Decrease)   .
       o-------------->o             .        .          .     o<------------o
       .               .             .        .          .           .            .
       .               .           BIS_RQ     .          .  BIS_RQ   .            .
       .               o------------------->o   •••      o--------->o             .
       .               .             .        .          .                        .
       .               .             .        .    •     .           .            .
       .               .             .        .    •     .           .            .
       .               .             .        .    •     .           .            .
       .               .             .        .          .           .            .
       .               .         BIS_REPLY    .          . BIS_REPLY .            .
       .               o<-------------------o   •••      o<---------o             .
       .               .             .        .          .           .            .
       .           DEACTIVATE_SESSION         .          .           .            .
       .               o---------->o          .          .           .            .
       .               .             .        .          .           .            .
       .               .             .    .(Normal .     .           .            .
       .               .             .    . UNBIND .     .           .            .
       .               .             .    Protocols)     .           .            .
```

Figure 3-13. Decreasing the Number of Sessions

CHANGING THE MAXIMUM SESSION LIMIT

The MODE control block (see page A-3) contains several session limit fields. These fields limit the number and polarity (first-speaker or bidder) of sessions that an LU can have with the partner LU and mode name represented by the MODE control block. The limit fields include:

- SESSION_LIMIT—maximum number of sessions.
- MIN_CONWINNERS_LIMIT—minimum number of potential first-speaker sessions, which limits the maximum number of bidder sessions. The SESSION_LIMIT less the number of bidder sessions must be greater than or equal to MIN_CONWINNERS_LIMIT.
- MIN_CONLOSERS_LIMIT—minimum number of potential bidder sessions, which limits the maximum number of first-speaker sessions. The SESSION_LIMIT less the number of first-speaker sessions must be greater than or equal to MIN_CONLOSERS_LIMIT.
- AUTO_ACTIVATIONS_LIMIT—the number of session that are activated independent of demand (explicit request). All such sessions are first-speaker sessions.

The change-number-of-sessions (CNOS) transaction program ("Chapter 5.4. Presentation Services--Control-Operator Verbs") can cause these session limits to change. The CNOS transaction programs at the two LUs come to an agreement on what the new session limits are to be via an exchange of Change Number of Sessions GDS variables (see SNA Formats). After an agreement on the new session limits is reached, each CNOS transaction program sends a CHANGE_SESSIONS record to its resources manager. The CHANGE_SESSIONS notifies the resources manager that a change in the session limits has occurred.

If the new session limits imply that new sessions may be activated, RM checks for any waiting session allocation requests. It creates multiple ACTIVATE_SESSION records, one for each waiting request, and sends them to the session manager (see "Activating a New Session" on page 3-15 for more on session activation). The resources manager does not, however, request that more sessions be activated than can be accommodated by the new session limits. The excess requests are retained for later processing.

The resources manager makes certain that at least the number of sessions equal to the AUTO_ACTIVATIONS_LIMIT are active. After this number of sessions is active, RM requests session activation only to satisfy waiting requests. For example, if AUTO_ACTIVATIONS_LIMIT = 2 and five requests are waiting, but the new session limits imply that seven sessions could be concurrently active, the resources manager sends to the session manager only five ACTIVATE_SESSION records.

When the session limits are decreased, one of the LUs is designated, by the CHANGE_SESSION_LIMIT verb's RESPONSIBLE parameter, as being "responsible" for deactivating sessions, as necessary to satisfy the new session limits. CHANGE_SESSION.RESPONSIBLE is set to YES if the resources manager is responsible to deactivate sessions.

The resources manager computes in TERMINATION_COUNT the number of sessions that its local LU is responsible to deactivate. RM chooses sessions to deactivate from the pool of free sessions with that LU and mode name, sending a BIS on each of the sessions that it has chosen and removing the entry for that session from the free session pool. The BIS is sent to inform the receiving resources manager that the sending RM will not initiate any subsequent brackets, and is sent only while the sending half-session is between brackets. When RM receives a BIS Reply (BIS_REPLY on page A-12) in response to its BIS, it decrements the TERMINATION_COUNT and sends to the session manager a DEACTIVATE_SESSION record for that session. The session manager then performs the normal UNBIND protocols. The exchange of BIS and its reply precedes a normal UNBIND (i.e., types X'01', X'02', or X'03'). See Figure 3-13 on page 3-16 for the steps involved.

If not enough free sessions can be deactivated to bring the TERMINATION_COUNT to 0, RM waits for sessions that are currently in use to become free before it sends any more BISs.

The value of the DRAIN_SELF field in the MODE control block determines whether RM will send BIS immediately when a session becomes free. If DRAIN_SELF = NO (i.e., waiting session allocation requests are not to be satisfied before session deactivation), RM will send BIS as soon as a session becomes free. If DRAIN_SELF = YES (i.e., waiting session allocation requests are to be satisfied before session activation), RM will send BIS only if no waiting requests can be satisfied by the free session. In the same way, DRAIN_SELF determines when BIS Reply is sent in reply to a BIS from the partner LU; i.e., if DRAIN_SELF = NO and the session is free, BIS Reply is sent immediately; otherwise, BIS Reply is sent only when no waiting requests can be satisfied by the session on which a BIS was received and the session is free.
The LU control operator may also explicitly request that a session be activated or deactivated. RM is notified of these control-operator requests with an RM_ACTIVATE_SESSION or RM_DEACTIVATE_SESSION record. The resources manager is responsible for sending ACTIVATE_SESSION or DEACTIVATE_SESSION records (preceded by the usual exchange of BIS and its reply for normal deactivation) to the session manager to satisfy these control-operator requests.

```
Presentation                     Resources                        Network
  Services                        Manager                         Services
     .                               .                               .
     .                               .                               .
     .                               .    SESSION_DEACTIVATED(SON)   .
     .                               o<------------------------------o
     .                               .                               .
     . CONVERSATION_FAILURE(RCB_ID)  .                               .
     o<------------------------------o                               .
     .                               .                               .
     .              •                .                               .
     .              •                .                               .
     .              •                .                               .
     .                               .                               .
     .    DEALLOCATE_RCB(RCB_ID)     .                               .
     o------------------------------>o                               .
     .                               .                               .
     .       RCB_DEALLOCATED         .                               .
     o<------------------------------o                               .
```

Figure 3-14. Session-Outage Flow

SESSION OUTAGE

An active session between two LUs sometimes fails. The session outage may be caused by a failure of one or both of the LUs, or by a failure in the path between the LUs. In the event of a session outage, the resources manager receives a SESSION_DEACTIVATED(REASON = SON) from the session manager. If the session is being used by a conversation, RM sends a CONVERSATION_FAILURE record to presentation services to inform it of the outage, and receives from PS a DEALLOCATE_RCB at some point. Regardless of whether the session is in use, RM destroys the associated SCB. Figure 3-14 illustrates a session-outage flow.

CREATION AND TERMINATION OF PRESENTATION SERVICES

The resources manager is responsible for creating and terminating instances of presentation services. (Presentation services, in turn, is responsible for starting up and taking down the transaction program with which it is to be associated.)

When a transaction program finishes its processing, presentation services notifies the resources manager via a TERMINATE_PS record.

RM

FUNCTION:		This process initializes the RM process giving RM addressability to pools (groups) of control blocks (e.g., LUCB, PARTNER_LU, MODE, TRANSACTION_PROGRAM, RCB, TCB, SCB), receives all input to the resources manager and routes the input to the appropriate procedure for processing.
INPUT:		A record is received asynchronously from session manager (SM), half-session (HS), presentation services (PS), and an initiator process.
OUTPUT:		Refer to the procedures that are called from this process for the outputs resulting from records received from other processes.
NOTES:	1.	An LUCB and TRANSACTION_PROGRAM (for initialization) are defined for this LU before RM is created.
	2.	An initiator process may send records to RM when a transaction program is to be started locally or when READY-TO-RECEIVE is to be sent on a session.
	3.	RM assumes that the LU, partner LUs, modes, and local transaction programs have been defined to the LU. RM also assumes that this definition is not changed by other components while RM is referencing the defined data.
	4.	Throughout this description, RM sends records to other processes (e.g., HS, PS, SM). If the send of a record fails, the recovery action is to destroy the record, log the failure in the system log, and continue processing. This send recovery action is not explicitly shown.

Referenced procedures, FSMs, and data structures:

PROCESS_SM_TO_RM_RECORD	page 3-23
PROCESS_HS_TO_RM_RECORD	page 3-20
PROCESS_INITIATOR_TO_RM_RECORD	page 3-20
PROCESS_PS_TO_RM_RECORD	page 3-22
PREVIOUS_TIME	page 3-92

```
Initialize PREVIOUS_TIME to the current system time (for more information refer to page 3-41)
Do forever:
   Receive a record.
   Select based on the sender of the record:
      When SM
         Call PROCESS_SM_TO_RM_RECORD(record received) (page 3-23).
      When HS
         Call PROCESS_HS_TO_RM_RECORD(record received) (page 3-20).
      When INITIATOR
         Call PROCESS_INITIATOR_TO_RM_RECORD(record received) (page 3-20).
      When PS
         Call PROCESS_PS_TO_RM_RECORD(record received) (page 3-22).
```

PROCESS_INITIATOR_TO_RM_RECORD

FUNCTION:	This procedure routes records received from an initiator process to the appropriate procedures for processing.
INPUT:	The current record from the initiator process
OUTPUT:	Refer to the procedures that are called from this process for the specific outputs.

Referenced procedures, FSMs, and data structures:

SEND_RTR_PROC	page 3-69
START_TP_PROC	page 3-77
START_TP	page A-19
SEND_RTR	page A-20

```
Select based on the type of record received:
   When START_TP
      Call START_TP_PROC(START_TP) (page 3-77).
   When SEND_RTR
      Call SEND_RTR_PROC(SEND_RTR) (page 3-69).
   Otherwise
      Log the error to the system log.
      Destroy the record.
```

PROCESS_HS_TO_RM_RECORD

FUNCTION:	This procedure routes records received from HS to the appropriate procedures for processing.
INPUT:	The current record from a half-session
OUTPUT:	Refer to the procedures that are called from this process for the specific outputs.
NOTES:	1. If an SCB is not found with an HS_ID matching the HS_ID in the received record, the record is discarded. This could occur, for example, if session outage occurred before RM had processed all the records from that half-session.
	2. If #FSM_BIS indicates that the session is closed, the record is discarded. This could occur, if the resources manager in the partner LU sends a -RSP(RTR) after having sent BIS_REPLY.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
BID_PROC	page 3-33
BID_RSP_PROC	page 3-35
ATTACH_PROC	page 3-30
FREE_SESSION_PROC	page 3-50
RTR_RQ_PROC	page 3-63
RTR_RSP_PROC	page 3-64
SECURITY_PROC	page 3-65
FSM_BIS_BIDDER	page 3-87
FSM_BIS_FSP	page 3-88
MU	page A-29
BID	page A-11
BID_RSP	page A-11
BIS_RQ	page A-12
BIS_REPLY	page A-12
FREE_SESSION	page A-12
RTR_RQ	page A-12
RTR_RSP	page A-13
SCB	page A-8

```
If no corresponding SCB is found for the HS process that sent the record then (Note 1)
   If the record is an MU then
      Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
       containing the MU (Appendix B).
   Else
      Destroy the record.
      Log the error to the system log.
Else
   If the state of #FSM_BIS ≠ CLOSED (page 3-87) then
      Select based on the type of record received:
         When BID
            Call BID_PROC(BID) (page 3-33).
         When BID_RSP
            Call BID_RSP_PROC(BID_RSP) (page 3-35).
         When MU
            If the MU contains an FMH-5 then
               Call ATTACH_PROC(MU) (page 3-30).
            If the MU contains an FMH-12 then
               Call SECURITY_PROC(MU) (page 3-65).
         When FREE_SESSION
            Call FREE_SESSION_PROC(FREE_SESSION) (page 3-50).
         When RTR_RQ
            Call RTR_RQ_PROC(RTR_RQ) (page 3-63).
         When RTR_RSP
            Call RTR_RSP_PROC(RTR_RSP) (page 3-64).
         When BIS_RQ
            Call #FSM_BIS(R, BIS_RQ, BIS_RQ.HS_ID) (page 3-87).
             associated with the half-session over which the BIS_RQ was received.
             (#FSM_BIS initialized in CREATE_SCB)
         When BIS_REPLY
            Call #FSM_BIS(R, BIS_REPLY, BIS_REPLY.HS_ID) (page 3-87)
             associated with the half-session over which the BIS_REPLY was received
             (#FSM_BIS initialized in CREATE_SCB).
   Else  (Note 2)
      If the record is an MU then
         Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
          containing the MU (refer to Appendix B).
         Log the error to the system log.
      Else
         Destroy the record.
         Log the error to the system log.
```

PROCESS_PS_TO_RM_RECORD

FUNCTION:	This procedure routes records received from presentation services to the appropriate procedures for processing.
INPUT:	The current record from presentation services
OUTPUT:	Refer to the procedures that are called from this procedure for the specific outputs.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
ALLOCATE_RCB_PROC	page 3-26
GET_SESSION_PROC	page 3-52
PS_TERMINATION_PROC	page 3-57
CHANGE_SESSIONS_PROC	page 3-39
RM_ACTIVATE_SESSION_PROC	page 3-61
RM_DEACTIVATE_SESSION_PROC	page 3-62
SEND_DEACTIVATE_SESSION	page 3-68
PS_ABEND_PROC	page 3-54
RCB	page A-6
SCB	page A-8
ALLOCATE_RCB	page A-15
GET_SESSION	page A-16
DEALLOCATE_RCB	page A-16
RCB_DEALLOCATED	page A-21
BRACKET_FREED	page A-18
TERMINATE_PS	page A-17
CHANGE_SESSIONS	page A-15
RM_ACTIVATE_SESSION	page A-16
RM_DEACTIVATE_SESSION	page A-17
UNBIND_PROTOCOL_ERROR	page A-17
ABEND_NOTIFICATION	page A-25

```
Select based on type of record received:
   When ALLOCATE_RCB
      Call ALLOCATE_RCB_PROC(ALLOCATE_RCB) (page 3-26).
   When GET_SESSION
      Call GET_SESSION_PROC(GET_SESSION) (page 3-52).
   When DEALLOCATE_RCB
      Find the RCB with RCB_ID equal to DEALLOCATE_RCB.RCB_ID.
      If there is no SCB with RCB_ID equal to DEALLOCATE_RCB.RCB_ID then
         Create a BRACKET_FREED (page A-18)
          with BRACKET_ID set to RCB.BRACKET_ID.
         Send the record to HS (Chapter 6.0).
      Discard the RCB.
      Build an RCB_DEALLOCATED record and send it to PS. (Chapter 5.0).
   When TERMINATE_PS
      Call PS_TERMINATION_PROC(TERMINATE_PS). (page 3-57).
   When CHANGE_SESSIONS
      Call CHANGE_SESSIONS_PROC(CHANGE_SESSIONS). (page 3-39).
   When RM_ACTIVATE_SESSION
      Call RM_ACTIVATE_SESSION_PROC(RM_ACTIVATE_SESSION). (page 3-61).
   When RM_DEACTIVATE_SESSION
      Call RM_DEACTIVATE_SESSION_PROC(RM_DEACTIVATE_SESSION). (page 3-62).
   When UNBIND_PROTOCOL_ERROR
      Call SEND_DEACTIVATE_SESSION(ACTIVE,UNBIND_PROTOCOL_ERROR.HS_ID,ABNORMAL,
       UNBIND_PROTOCOL_ERROR.SENSE_CODE).   (page 3-68).
   When ABEND_NOTIFICATION
      Call PS_ABEND_PROC(ABEND_NOTIFICATION) (page 3-54).
```

PROCESS_SM_TO_RM_RECORD

FUNCTION:	This procedure routes records received from SM to the appropriate procedures for processing.
INPUT:	The current record from SM
OUTPUT:	Refer to the procedures that are called from this procedure for the specific outputs.

Referenced procedures, FSMs, and data structures:

ACTIVATE_SESSION_RSP_PROC page 3-25
SESSION_ACTIVATED_PROC page 3-70
SESSION_DEACTIVATED_PROC page 3-72
ACTIVATE_SESSION_RSP page A-13
SESSION_ACTIVATED page A-14
SESSION_DEACTIVATED page A-14

```
Select based on the type of record received:
   When ACTIVATE_SESSION_RSP
      Call ACTIVATE_SESSION_RSP_PROC(ACTIVATE_SESSION_RSP) (page 3-25).
   When SESSION_ACTIVATED
      Call SESSION_ACTIVATED_PROC(SESSION_ACTIVATED) (page 3-70).
   When SESSION_DEACTIVATED
      Call SESSION_DEACTIVATED_PROC(SESSION_DEACTIVATED) (page 3-72).
```

ACTIVATE_NEEDED_SESSIONS

FUNCTION:	This procedure activates sessions as required by the number of waiting requests and change-number-of-sessions (CNOS) processing. Sessions are activated so as to satisfy the waiting requests, but not to exceed the (LU, mode) session limit. If all waiting requests are satisfied, additional sessions are activated to bring the number of sessions up to the minimum of the MODE.AUTO_ACTIVATIONS_LIMIT and MODE.MIN_CONWINNERS_LIMIT.
INPUT:	The LU name and mode name, respectively, of the partner LU
OUTPUT:	Zero or more ACTIVATE_SESSION records to SM

Referenced procedures, FSMs, and data structures:

SM	page 4-48
SESSION_ACTIVATION_POLARITY	page 3-71
SEND_ACTIVATE_SESSION	page 3-65
LU_NAME	page 3-91
MODE_NAME	page 3-91
ACTIVATE_SESSION	page A-20
MODE	page A-3

```
Get addressability to the MODE control block associated with LU_NAME and
 MODE_NAME.
Do for each waiting request for sessions identified by LU_NAME and MODE_NAME
 while the polarity returned by SESSION_ACTIVATION_POLARTIY(LU_NAME,MODE_NAME)
 (page 3-71) ≠ NONE.
   If polarity = FIRST_SPEAKER then
      Call SEND_ACTIVATE_SESSION(LU_NAME, MODE_NAME, FIRST_SPEAKER) (page 3-65)
        to send an ACTIVATE_SESSION record to SM.
   Else (BIDDER)
      Call SEND_ACTIVATE_SESSION(LU_NAME, MODE_NAME, BIDDER) (page 3-65).
Do while (MODE.ACTIVE_CONWINNERS_COUNT + MODE.PENDING_CONWINNERS_COUNT) <
 the minimum of (MODE.AUTO_ACTIVATIONS_LIMIT, MODE.MIN_CONWINNERS_LIMIT) and the polarity
 returned by SESSION_ACTIVATION_POLARITY(LU_NAME, MODE_NAME) (page 3-71) = FIRST_SPEAKER.
   Call SEND_ACTIVATE_SESSION(LU_NAME, MODE_NAME, FIRST_SPEAKER) (page 3-65).
```

ACTIVATE_SESSION_RSP_PROC

FUNCTION:	This procedure handles the processing of the response to a previously issued ACTIVATE_SESSION request. The session counts in the appropriate MODE entry are updated and further processing is invoked depending on the response type.
INPUT:	ACTIVATE_SESSION_RSP from SM
OUTPUT:	SESSION_ALLOCATED to PS, the mode session counts are adjusted, and pending activate session requests are discarded
NOTE:	The PENDING_ACTIVATION will not be found if RM had previously requested deactivation of the pending session as a result of change-number-of-sessions processing. In this case, no processing of the ACTIVATE_SESSION_RSP is performed, since the session is being deactivated.

Referenced procedures, FSMs, and data structures:

SUCCESSFUL_SESSION_ACTIVATION	page 3-80
UNSUCCESSFUL_SESSION_ACTIVATION	page 3-83
ACTIVATE_SESSION_RSP	page A-13
PENDING_ACTIVATION, see ACTIVATE_SESSION	page A-20
MODE	page A-3


```
If there exists a PENDING_ACTIVATION with a correlator equal to
 ACTIVATE_SESSION_RSP.CORRELATOR then
   Get addressability to the MODE control block associated with the LU_NAME and
    MODE_NAME of the PENDING_ACTIVATION.
   Decrement MODE.PENDING_CONWINNERS_COUNT or MODE.PENDING_CONLOSERS_COUNT by 1,
    as appropriate to the session polarity.
   Decrement MODE.PENDING_SESSION_COUNT by 1.
   If ACTIVATE_SESSION_RSP.TYPE = POS then
      Increment MODE.ACTIVE_CONWINNERS_COUNT or MODE.ACTIVE_CONLOSERS_COUNT by 1,
       as appropriate to the session polarity.
      Increment MODE.ACTIVE_SESSION_COUNT by 1.
      Call SUCCESSFUL_SESSION_ACTIVATION(PENDING_ACTIVATION.LU_NAME,
       PENDING_ACTIVATION.MODE_NAME, ACTIVATE_SESSION_RSP.SESSION_INFORMATION) (page 3-80).
   Else (negative response)
      Call UNSUCCESSFUL_SESSION_ACTIVATION(PENDING_ACTIVATION.LU_NAME,
       PENDING_ACTIVATION.MODE_NAME, ACTIVATE_SESSION_RSP.ERROR_TYPE) (page 3-83).
   Discard the PENDING_ACTIVATION
Else
   Do nothing (see Note).
```

ALLOCATE_RCB_PROC

FUNCTION:	This procedure handles the allocation of resource control blocks (RCBs). This procedure creates the RCB_ALLOCATED record and initializes the fields of the record. It then calls the appropriate procedure, depending upon the ALLOCATE_RCB parameter settings. The procedure that this procedure calls changes the setting of some of the RCB_ALLOCATED fields. The RCB_ALLOCATED is then sent to PS to inform it of the outcome of the ALLOCATE_RCB request.
INPUT:	ALLOCATE_RCB
OUTPUT:	RCB_ALLOCATED to PS
NOTE:	When ALLOCATE_RCB.IMMEDIATE_SESSION is set to YES, RM is to check to see if a first-speaker half-session is currently available for use. If such a session is available, the RCB_ID is passed to PS and the request completes successfully. (If IMMEDIATE_SESSION is set to NO, PS sends a separate GET_SESSION request to RM to request that a half-session be allocated to a particular conversation resource.)

Referenced procedures, FSMs, and data structures:

TEST_FOR_FREE_FSP_SESSION	page 3-82
CREATE_RCB	page 3-43
PS	page 5.0-8
ALLOCATE_RCB	page A-15
RCB_ALLOCATED	page A-21

```
Create an RCB_ALLOCATED record initializing RETURN_CODE to OK and RCB_ID to a null value.
If ALLOCATE_RCB.IMMEDIATE_SESSION is set to YES then
   Call TEST_FOR_FREE_FSP_SESSION(ALLOCATE_RCB, RCB_ALLOCATED) (page 3-82).
Else
   Call CREATE_RCB(ALLOCATE_RCB, RCB_ALLOCATED) (page 3-43).
Send the RCB_ALLOCATED record to PS (Chapter 5.0).
Destroy ALLOCATE_RCB.
```

ATTACH_CHECK

FUNCTION:	This procedure checks particular fields of the passed FM header 5 (FMH-5) for validity. (PS is responsible for additional checks.)
INPUT:	An FM header 5 and the HS_ID of the half-session (See *SNA Formats* for the formats of FM headers)
OUTPUT:	X'00000000', if no error; or sense data returned by ATTACH_LENGTH_CHECK; or data returned by ATTACH_SECURITY_CHECK; or one of the following sense data values: X'080F6051' Security Not Valid X'084B6031' TP Not Available--Retry Allowed X'084C0000' TP Not Available--No Retry X'1008600B' Unrecognized FMH Command X'10086011' Invalid Logical Unit of Work X'10086021' TP Name Not Recognized X'10086040' Invalid Attach Parameter X'10086041' Sync Level Not Supported

Referenced procedures, FSMs, and data structures:

ATTACH_LENGTH_CHECK	page 3-28
ATTACH_SECURITY_CHECK	page 3-32

```
Select based on the Command field of the FMH-5:
   When Attach  (The FMH-5 is an Attach FM header)
      Call ATTACH_LENGTH_CHECK(Attach) (page 3-28) to determine whether any
       FMH-5 fields have an invalid length.
      If ATTACH_LENGTH_CHECK indicates that a field length is invalid then
         Return with the sense data provided by ATTACH_LENGTH_CHECK.
      If a logical-unit-of-work ID (LUW ID) is present in the Attach then
         If the logical-unit-of-work ID's network-qualified LU name has a null network ID and
          the receiving LU has a non-null network ID in its network-qualified LU name then
             Return with sense data X'10086011'. (A null network ID in LUW is not valid unless
                                                  this LU's network ID is also null.)
      Else  (LUW ID not present)
         If the sync level specified in the Attach is SYNCPT then
             Return with sense data X'10086011' (LUW required on sync point conversations).
      If the transaction program specified in the Attach exists at this LU then
         Select based on the sync level specified in the Attach:
          (Optional receive check--the sync level support specified
          in the FMH-5 must be compatible with the sync level
          supported by the partner LU).
            When None or Confirm
              Do nothing.  (All LUs support sync level None and Confirm.)
            When Confirm, Sync Point, and Backout
               If the sessions to the remote LU do not support confirm, sync point,
                and backout then
                  Return with sense data X'10086040' (Invalid Attach Parameter).
            Otherwise
               Return with sense data X'10086040' (Unrecognized sync level).
         If the sync level specified in the Attach is not supported by
          the transaction program then
            Return with sense data X'10086041' (Sync Level Not Supported).
         If the transaction program is temporarily disabled then
            Return with sense data X'084B6031' (TP Not Available--Retry Allowed).
         If the transaction program is permanently disabled then
            Return with sense data X'084C0000' (TP Not Available--No Retry).
         If the transaction program requires security parameters in the Attach and the
          sending LU is not permitted by this LU to send them (as communicated in Bind) then
            Return with sense data X'080F6051' (Security Not Valid).
         Call ATTACH_SECURITY_CHECK(Attach) (page 3-32) to check that all security
          requirements are met.
         If ATTACH_SECURITY_CHECK indicates a security violation then
          Return with the data provided by ATTACH_SECURITY_CHECK.
      Else
         Return with sense data X'10086021' (TP Name Not Recognized).
   Otherwise
      Return with sense data X'1008600B' (Unrecognized FMH-5 command field).
Return with sense data X'00000000' indicating no error.
```

ATTACH_LENGTH_CHECK

FUNCTION:	This procedure checks the length fields in the passed Attach for validity.
INPUT:	An FMH-5 Attach header (see SNA Formats)
OUTPUT:	Sense data values reflecting the result of the length checks. One of the following sense data values is returned:
	X'00000000' No error X'10086000' FMH Length Not Correct X'10086005' Access Security Information Length Invalid X'10086009' Invalid Parameter Length X'10086011' Invalid Logical Unit of Work
NOTE:	The total length of the Attach can be greater than the sum of the lengths of the currently defined fields, to allow for the addition of new Attach fields.

```
Set BYTE_OFFSET to 5 (offset of Fixed Length Parameters field in Attach).
If the Attach length ≤ BYTE_OFFSET then
   Return with X'10086000' (FMH Length Not Correct).
If the value of the Fixed Length Parameters field < 3 then
   Return with X'10086009' (Invalid Parameter Length).
Set BYTE_OFFSET to BYTE_OFFSET + the value of the Fixed Length Parameters field + 1
 (offset of TP name Length field).
If the Attach length ≤ BYTE_OFFSET then
   Return with X'10086000' (FMH Length Not Correct).
Set BYTE_OFFSET to BYTE_OFFSET + the value of the TP name Length field + 1
 (offset of Access Security Information Length field).
Select based on the following comparisons:
   When the Attach length < BYTE_OFFSET
      Return with X'10086000' (FMH Length Not Correct).
   When the Attach length = BYTE_OFFSET
      Return with X'00000000' (Access Security Information and fields following not present).
   When the Attach length > BYTE_OFFSET (Access Security Information present)
      Continue processing.
If the value of the Access Security Information Length field > 0 then
 (Access Security information is present)
   If the Access Security subfield length > the allowed length (12) or
    more than three Access Security subfields are present or
    the sum of the lengths of the Access Security subfields does not equal
    the total length of the Access Security Information field then
      Return with X'10086005' (Access Security Information Length Invalid).
Set BYTE_OFFSET to BYTE_OFFSET + the value of the Access Security Information Length field + 1
 (offset of LUW Identifier Length field).
Select based on the following comparisons:
   When the Attach length < BYTE_OFFSET
      Return with X'10086000' (FMH Length Not Correct).
   When the Attach length = BYTE_OFFSET
      Return with X'00000000' (LUW Identifier and following fields not present).
   When the Attach length > BYTE_OFFSET (LUW Identifier present)
      Do nothing.
If the value of the LUW Identifier Length field > 0 then (LUW Identifier present)
   If the value of the LUW Identifier Length field < 10 or > 26 then
      Return with X'10086011' (Invalid Logical Unit of Work).
   If the value of the LUW Identifier Length field ≠ the value of the LUW Identifier
    LU name Length field + 9 then
      Return with X'10086011' (Invalid Logical Unit of Work).
Set BYTE_OFFSET to BYTE_OFFSET + the value of the LUW Identifier Length field + 1
 (offset of Conversation Correlator Length field).
Select based on the following comparisons:
   When the Attach length < BYTE_OFFSET
      Return with X'10086000' (FMH Length Not Correct).
   When the Attach length = BYTE_OFFSET
      Return with X'00000000' (Conversation Correlator not present).
   When the Attach length > BYTE_OFFSET (Conversation Correlator present)
      Do nothing.
```

```
Set BYTE_OFFSET to BYTE_OFFSET + the value of the Conversation Correlator Length field + 1
 (offset of byte following ATTACH).
If the Attach length < BYTE_OFFSET then
   Return with X'10086000' (FMH Length Not Correct).
Else
   Return with X'00000000' (All length fields in Attach are valid).
```

ATTACH_PROC

FUNCTION:	This procedure performs Attach processing. This procedure checks to see if the session is already in use. If the session is not in use, the appropriate subroutines are called to check certain fields in the Attach FM header for validity. If a partner-LU protocol error is found, the appropriate procedure is called to deactivate the session; otherwise, a new conversation with a new PS process is started.
INPUT:	An MU containing an FM Header 5
OUTPUT:	None
NOTES:	1. If the state of #FSM_SCB_STATUS (initialized in CREATE_SCB) is PENDING_ATTACH, the half-session is first-speaker and a prior BID was received, or the half-session is a secondary first-speaker or bidder and has just been activated. 2. This protocol error occurs, for example, when the first-speaker half-session sends an Attach FM header after having positively responded to a Bid from the bidder half-session, or when an Attach FM header is received for which there was no prior Bid.

Referenced procedures, FSMs, and data structures:

SEND_DEACTIVATE_SESSION	page 3-68
ATTACH_CHECK	page 3-26
PS_CREATION_PROC	page 3-55
QUEUE_ATTACH_PROC	page 3-60
PURGE_QUEUED_REQUESTS	page 3-59
SEND_ATTACH_TO_PS	page 3-66
FSM_SCB_STATUS_BIDDER	page 3-85
FSM_SCB_STATUS_FSP	page 3-86
CONNECT_RCB_AND_SCB	page 3-42
TRANSACTION_PROGRAM	page A-5
MU	page A-29
TCB_ID	page 3-91
RCB_ID	page 3-91
SCB	page A-8

```
Set TCB_ID and RCB_ID to null.
Find the SCB corresponding to the HS process that sent the Attach.
If the state of the #FSM_SCB_STATUS ¬= PENDING_ATTACH then (Note 2)
   Call SEND_DEACTIVATE_SESSION(ACTIVE, SCB.HS_ID, ABNORMAL, X'20030000'). (page 3-68)
Else
   Call ATTACH_CHECK(FMH_5, MU.HS_TO_RM.HS_ID) (page 3-26) to determine if the
    Attach contains any errors.
   Select based on the sense data returned by ATTACH_CHECK.
      When X'FFFFFFFF'
         Call SEND_DEACTIVATE_SESSION(ACTIVE, SCB.HS_ID, ABNORMAL, X'080F6051'). (page 3-68)
         Call buffer manager(FREE_BUFFER, buffer address) to release the buffer containing MU
          (Appendix B).
      When X'10086040'
         Call SEND_DEACTIVATE_SESSION(ACTIVE, SCB.HS_ID, ABNORMAL, X'10086040'). (page 3-68)
         Call buffer manager(FREE_BUFFER, buffer address) to release the buffer containing MU
          (Appendix B).
      When X'10086011'
         Call SEND_DEACTIVATE_SESSION(ACTIVE, SCB.HS_ID, ABNORMAL, X'10086011'). (page 3-68)
         Call buffer manager(FREE_BUFFER, buffer address) to release the buffer containing MU
          (Appendix B).
      Otherwise
         Select based on the FMH_5.COMMAND.
```

```
When ATTACH
   If the sense data returned by ATTACH_CHECK ¬= X'10086021' then
                                                           (TP name recognized)
      Find the TRANSACTION_PROGRAM structure corresponding to the
       transaction program named in the Attach record.
      If the sense data returned by ATTACH_CHECK ¬= X'00000000' or
       TRANSACTION_PROGRAM.INSTANCE_COUNT < TRANSACTION_PROGRAM.INSTANCE_LIMIT
       then
         Call PS_CREATION_PROC(MU, TCB_ID, RCB_ID, TRANSACTION_PROGRAM,CREATE_RC).
                                                           (page 3-55)
         If CREATE_RC is SUCCESS then
            Call #FSM_SCB_STATUS(R, ATTACH, UNDEFINED) (page 3-84).
            Set SCB.RCB_ID to RCB_ID.
            Call CONNECT_RCB_AND_SCB(RCB_ID, MU.HS_TO_RM.HS_ID). (page 3-42)
            Call SEND_ATTACH_TO_PS(MU, TCB_ID, RCB_ID, sense code). (page 3-66)
         Else (PS creation failed)
            If the TRANSACTION_PROGRAM.INSTANCE_COUNT is greater than 0 and
             the sense data returned by ATTACH_CHECK=X'00000000' then
               Call QUEUE_ATTACH_PROC(MU) (page 3-60).
            Else
               Call SEND_DEACTIVATE_SESSION(ACTIVE, MU.HS_TO_RM.HS_ID, ABNORMAL,
                                            X'08640000') (page 3-68).
               Call buffer manager(FREE_BUFFER, buffer address) to release the
                buffer containing MU (Appendix B).
               Log the creation failure in the system log.
            If the TRANSACTION_PROGRAM.INSTANCE_COUNT is 0 then
               Call PURGE_QUEUED_REQUESTS(TRANSACTION_PROGRAM) (page 3-59).
      Else
         Call QUEUE_ATTACH_PROC(MU). (page 3-60)
   Else  (TP name is not recognized)
      Set the pointer to the TRANSACTION_PROGRAM structure to null.
      Call PS_CREATION_PROC(MU, TCB_ID, RCB_ID, TRANSACTION_PROGRAM, CREATE_RC).
           (page 3-55)  (Create a PS to reject the Attach.)
      If CREATE_RC is SUCCESS then
         Call #FSM_SCB_STATUS(R, ATTACH, UNDEFINED) (page 3-84).
         Set SCB.RCB_ID to RCB_ID.
         Call CONNECT_RCB_AND_SCB(RCB_ID, MU.HS_TO_RM.HS_ID). (page 3-42)
         Call SEND_ATTACH_TO_PS(MU, TCB_ID, RCB_ID, sense code). (page 3-66)
      Else (PS creation failed)
         Call SEND_DEACTIVATE_SESSION(ACTIVE, MU.HS_TO_RM.HS_ID, ABNORMAL,
                                      X'08640000') (page 3-68).
         Call buffer manager(FREE_BUFFER, buffer address) to release the buffer
          containing MU (Appendix B).
         Log the creation failure in the system log.
```

ATTACH_SECURITY_CHECK

FUNCTION:	This procedure performs all security checks on an incoming Attach.
INPUT:	The received Attach (see SNA Formats)
OUTPUT:	A code or sense data value indicating the result of the security check: X'FFFFFFFF' local indication of a partner-LU security protocol error X'10086040' an Attach parameter is present that is not authorized by the BIND security indicators X'080F6051' a remote TP security error is found X'00000000' the Attach passes all security checks
NOTES:	1. All checks within this procedure are required receive checks for implementations that support the conversation-level security option. 2. If the use of profiles is not supported and one is received, it is ignored. If the use of profiles is supported, the option of requiring profiles on every Attach versus on Attach only to specific resources is installation-defined. If a profile is required on every Attach, connectivity problems with LUs that cannot send profiles may result. An unauthorized combination of user ID and profile means that the user that provides the profile is not permitted to supply that profile. Profiles are installation defined at the receiver of the Attach. Profiles assigned to specific user IDs are installation defined at the receiver of the Attach. The use and interpretation of profiles is not limited to access to secure transaction programs. Profiles may be used to restrict access to resources in implementation- and installation-defined ways (e.g., as group IDs).

```
If the Attach indicates End User Already Verified and
 this LU does not accept an Already Verified indication in an Attach from the partner LU then
   Return with sense data X'10086040' (Invalid Attach Parameter).
If the Attach contains security parameters and
 this LU does not accept security parameters in an Attach from the partner LU then
   Return with sense data X'10086040' (Invalid Attach Parameter).
If the target transaction program requires security parameters in an Attach and
 the Attach does not contain security parameters then
   Return with sense data X'080F6051' (Security Not Valid).
If the Attach contains no security parameters then
   Return with code X'00000000' (security fields not present nor required).
If there are multiple security subfields of the same type in the Attach then
   Return with code X'FFFFFFFF' (partner-LU security protocol error).
If there is a security subfield of an unrecognized type then
   Return with code X'FFFFFFFF' (partner-LU security protocol error).
If the Attach contains a profile and does not contain a user ID then
   Return with sense data X'080F6051' (Security Not Valid).
If the Attach contains a password and does not contain a user ID then
   Return with sense data X'080F6051' (Security Not Valid).
If the Attach indicates end user not already verified and
 the Attach contains a user ID and does not contain a password then
   Return with sense data X'080F6051' (Security Not Valid).
If the Attach indicates end user not already verified and
 the Attach contains an unauthorized combination of user ID and profile or (see note 2)
  the Attach contains an invalid combination of user ID and password then
   Return with sense data X'080F6051' (Security Not Valid).
If the Attach indicates end user is already verified and
 the Attach does not contain a user ID or the Attach does contain a password then
   Return with code X'FFFFFFFF' (partner-LU security protocol error).
If there is limited access to the target transaction program, which is based upon the Attach's
 user ID, and/or profile, and/or the LU name of the Attach sender then
   If the user ID and/or profile and/or partner-LU name is not permitted access
    to this transaction program then
      Return with sense data X'080F6051' (Security Not Valid).
Return with code X'00000000' (Attach passes all security checks).
```

BID_PROC

FUNCTION:	This procedure handles bids for the use of sessions. This procedure first checks whether the bid should be rejected because the local operator has reset the session limit to 0 with no draining of the partner LU's requests, and this LU does not support parallel sessions to the partner LU. In this case, a -BID_RSP(088B) is sent to HS. The -BID_RSP(088B) can be sent even if the partner LU is the first speaker. If -BID_RSP(088B) is not sent, the procedure checks to see if the requested session is free. If so, it removes the session from the free-session pool and sends a positive BID_RSP to HS. If the session is not free, it sends a negative BID_RSP to HS. An implementation may allow a transaction program to reserve a session for its own use before the conversation begins. If a session has been reserved, a negative BID_RSP is sent to HS even though a conversation has not been started on the session. Since the transaction program might never use the reserved session (e.g., the transaction program terminates abnormally before the conversation is started), the negative response carries an X'0814' sense code (Bracket Bid Reject--RTR Forthcoming) to allow the session to be freed, in case the reserved session is not needed by a conversation. Reserving a session is implementation dependent and is not shown here.
INPUT:	BID
OUTPUT:	BID_RSP to HS. The RTI field of the BID_RSP is set to either POS or NEG. If a protocol error is detected, the session is deactivated.
NOTE:	If RM issues an RTR to the partner LU and receives a positive response to the RTR, the HS_ID of the session over which the RTR flows will not be free when the BID is received. When RM issued the RTR, it removed that session from the free-session pool. When RM sends BIS on a session or when RM bids on a bidder session, that session is removed from the free session pool.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
FSM_SCB_STATUS_BIDDER	page 3-85
FSM_SCB_STATUS_FSP	page 3-86
FSM_BIS_BIDDER	page 3-87
FSM_BIS_FSP	page 3-88
SEND_DEACTIVATE_SESSION	page 3-68
BID	page A-11
MODE	page A-3
BID_RSP	page A-11

```
If the state of #FSM_BIS is BIS_RCVD or CLOSED (page 3-87) then
   Call SEND_DEACTIVATE_SESSION(ACTIVE, BID.HS_ID, ABNORMAL, X'20080000') (page 3-68)
    (optional receive check, No Begin Bracket).
Else
   Get addressability to the MODE control block associated with the LU and
    mode name for the session on which the BID was received.
   If parallel sessions are not supported to the partner LU and
    MODE.SESSION_LIMIT = 0 and MODE.DRAIN_PARTNER = NO and the state of
    #FSM_BIS (page 3-87) is BIS_SENT then
      Create a BID_RSP record with RTI set to NEG and SENSE_CODE set to
       X'088B0000' and send it to HS (Chapter 6.0).
   Else
      If the state of #FSM_SCB_STATUS is FREE (page 3-84) then
         Call #FSM_SCB_STATUS(R, BID, UNDEFINED) (page 3-84)
         Remove the session from the free-session pool.
         Create a BID_RSP record with RTI set to POS and SENSE_CODE set to
          X'00000000' and send it to HS (Chapter 6.0).
      Else
         If this is a first-speaker session then
            Create a BID_RSP record with RTI set to NEG and SENSE_CODE set to
             X'08130000' or X'08140000' (implementation-dependent choice)
             and send it to HS (Chapter 6.0).
            If SENSE_CODE was X'08140000' then
               Remember that this LU owes RTR to its partner (RTR must be sent to the
                partner LU before it can bid again for this session).
         Else
            Call SEND_DEACTIVATE_SESSION(ACTIVE, BID.HS_ID, ABNORMAL, X'20030000') (page 3-68)
             (optional receive check, Bracket Error).
Destroy the BID record.
```

BID_RSP_PROC

FUNCTION: This procedure handles the processing of responses to bids for the use of half-sessions.

A bid response is usually sent to the resources manager in response to a previous bid for a bidder half-session. In this case, when the input is a positive bid response, this procedure calls the appropriate subroutines to cause the RCB and SCB to point to each other and to establish the PS and HS connection. It then informs PS that a session has been successfully allocated via a SESSION_ALLOCATED record.

When the input is a negative bid response, this procedure changes the RCB so that it no longer points to the SCB that sent the bid response, and retries the GET_SESSION request, which was stored in the RCB when the BID_RQ was issued, on another half-session.

A negative bid response with sense data of X'088B0000' is handled specially. This bid response is sent by an LU to indicate that the session limit has been reset to 0 for a single-session connection and draining of the partner is not allowed. Sending of -BID_RSP(088B) is permitted only in the single-session case.

A -BID_RSP(088B) record may arrive from either a bidder or first-speaker session. If from a bidder session, it is in response to a previous bid. If from a first-speaker session, no previous bid was sent. A -BID_RSP(088B) record is the only bid response that can arrive from a first-speaker session.

INPUT: A positive or negative BID_RSP record

OUTPUT: SESSION_ALLOCATED to PS, or GET_SESSION to GET_SESSION_PROC (page 3-52)

NOTES:

1. When a BID_RQ record is sent to HS, the RCB is set to point to the SCB for which the bid is being sent; the SCB, however, does not point to the RCB until a positive BID_RSP record is received.

2. A -BID_RSP(088B) record indicates that the partner LU has reset the session limit to 0 and is not permitting draining of the local LU's requests. The session is deactivated with UNBIND(Cleanup).

3. PS stores in the RCB information that helps HS to set the fields in the request/response header (RH). Part of the information states whether the data being sent to HS is the beginning of a conversation (in which case HS will set BBI) or is part of an existing conversation (in which case the BBI is set to ¬BB). When RM chooses a bidder half-session to allocate to PS, the BID_WITHOUT_ATTACH record that RM sends to HS also triggers HS to set BBI to BB. Since PS is unaware of whether RM allocated a bidder or first-speaker half-session (and thus does not know whether the Begin Bracket, which is sent only once during a conversation, has already been sent), RM informs PS, in the SESSION_ALLOCATED record, as to whether the session assigned to the Allocate request is in-bracket (in conversation) or not.

Referenced procedures, FSMs, and data structures:

PS	page 5.0-8
HS	page 6.0-3
SEND_DEACTIVATE_SESSION	page 3-68
SET_RCB_AND_SCB_FIELDS	page 3-75
CONNECT_RCB_AND_SCB	page 3-42
GET_SESSION_PROC	page 3-52
FSM_RCB_STATUS_FSP	page 3-90
FSM_RCB_STATUS_BIDDER	page 3-89
BID_RSP	page A-11
GET_SESSION	page A-16
SCB	page A-8
RCB	page A-6
SESSION_ALLOCATED	page A-22

```
If BID_RSP.RTI = NEG and BID_RSP.SENSE_CODE = X'088B0000' then (see Note 2)
   If the partner LU does not support parallel sessions then
      Reset conwinner, conloser, and session limits for this mode to 0.
      Call SEND_DEACTIVATE_SESSION(ACTIVE, BID_RSP.HS_ID, CLEANUP, X'00000000')
        (page 3-68).
   Else  (X'088B' valid only from a single-session connection)
      Call SEND_DEACTIVATE_SESSION(ACTIVE, BID_RSP.HS_ID, ABNORMAL, X'20100000')
        (page 3-68).
Else
   Find the RCB associated with the conversation where
    state of #FSM_RCB_STATUS = PENDING_SCB (page 3-89) and
    RCB.HS_ID = BID_RSP.HS_ID.
   If BID_RSP.RTI = POS then
      Call SET_RCB_AND_SCB_FIELDS(RCB.RCB_ID, BID_RSP.HS_ID) (page 3-75).
      Call CONNECT_RCB_AND_SCB(RCB.RCB_ID, BID_RSP.HS_ID, REPLY) (page 3-42).
      Find the SCB associated with the HS that received the BID_RSP.
      Create a SESSION_ALLOCATED record with RETURN_CODE set to OK,
       SEND_RU_SIZE set to SCB.SEND_RU_SIZE,
       LIMITED_BUFFER_POOL_ID set to SCB.LIMITED_BUFFER_POOL_ID,
       PERMANENT_BUFFER_POOL_ID set to SCB.PERMANENT_BUFFER_POOL_ID, and
       IN_CONVERSATION set to YES. (Note 3)
      Send the SESSION_ALLOCATED record to the PS that requested the session.
   Else (-RSP(Bid)--retry request on another session)
      Set RCB.HS_ID to a null value.
      Call #FSM_RCB_STATUS(R, NEG_BID_RSP, UNDEFINED) (page 3-89).
       (State of #FSM_RCB_STATUS = FREE).
      If BID_RSP.SENSE_CODE = X'08140000' then
         Remember that the partner LU owes an RTR on this session.
           (Bidder cannot bid again for this session until RTR received).
      Create a GET_SESSION record initialized with the information from the original
       GET_SESSION record, saved in the RCB when the BID record was sent.
      Call GET_SESSION_PROC(GET_SESSION) (page 3-52).
```

BIDDER_PROC

FUNCTION:	This procedure handles the allocation processing for a bidder half-session.
	The HS_ID of the bidder half-session is placed in the RCB of the conversation for which the session was requested. The state of #FSM_RCB_STATUS is set to indicate that the conversation is pending confirmation that it can use the SCB. This procedure then creates a BID_WITHOUT_ATTACH record and sends it to HS.
INPUT:	GET_SESSION and HS_ID, the ID of the bidder half-session that was chosen by GET_SESSION_PROC (page 3-52)
OUTPUT:	BID_WITHOUT_ATTACH to HS. No SESSION_ALLOCATED record is sent to PS until confirmation that the bidder may use the session is received from the first-speaker (i.e., until a positive BID_RSP is received). RCB.#FSM_RCB_STATUS is set to FSM_RCB_STATUS_BIDDER.
NOTE:	A copy of the GET_SESSION record is created so that, if the bid for the session fails, the request can be retried on a different session.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
FSM_RCB_STATUS_FSP	page 3-90
FSM_RCB_STATUS_BIDDER	page 3-89
GET_SESSION	page A-16
HS_ID	page 3-91
BID_WITHOUT_ATTACH	page A-17
RCB	page A-6

Find the RCB identified by GET_SESSION.RCB_ID.
Set RCB.HS_ID to HS_ID.
Initialize #FSM_RCB_STATUS to FSM_RCB_STATUS_BIDDER (page 3-89).
Call #FSM_RCB_STATUS(S, GET_SESSION, UNDEFINED) (page 3-89).
Save the contents of the GET_SESSION record in the RCB (see Note).
Build and send a BID_WITHOUT_ATTACH record to HS (Chapter 6.0).

BIS_RACE_LOSER

FUNCTION:	This procedure performs the processing necessary when a BIS race occurs and this side of the session is the race loser. This procedure first decrements the PENDING_TERMINATION_COUNT and issues a BIS_REPLY. It then attempts to find another session from the free-session pool on which to send a BIS_RQ.
INPUT:	HS_ID, the ID of the session over which the BIS race occurred
OUTPUT:	BIS_REPLY and, if there is a free session, BIS_RQ to HS, the MODE pending termination counts are updated
NOTE:	When the SESSION_DEACTIVATION_POLARITY is EITHER, free first-speaker sessions are deactivated in preference to free bidder sessions.

Referenced procedures, FSMs, and data structures:

SEND_BIS_RQ	page 3-67
SESSION_DEACTIVATION_POLARITY	page 3-74
HS	page 6.0-3
HS_ID	page 3-91
LU_NAME	page 3-91
MODE_NAME	page 3-91
BIS_REPLY	page A-12
MODE	page A-3


```
Get addressability to the MODE control block associated with the partner LU and mode name
 of the session identified by HS_ID.
Decrement MODE.PENDING_TERMINATION_CONWINNERS or MODE.PENDING_TERMINATION_CONLOSERS by 1,
 as appropriate to the session polarity.
Create a BIS_REPLY record and send it to HS (Chapter 6.0).
Call SESSION_DEACTIVATION_POLARITY(LU_NAME, MODE_NAME) (page 3-74).
 to determine the polarity of an additional session to deactivate (if any).
If there is a free session of the appropriate type then (see Note)
   Call SEND_BIS_RQ(HS_ID) (page 3-67).
   Remove the session from the free-session pool.
```

CHANGE_SESSIONS_PROC

FUNCTION: This procedure performs the processing that is required when two LU service transaction programs exchange CHANGE_NUMBER_OF_SESSIONS requests and a new session limit is agreed upon. PS.COPR (Chapter 5.4) sends CHANGE_SESSIONS to RM after CHANGE_NUMBER_OF_SESSIONS requests have been successfully exchanged.

A new TERMINATION_COUNT is computed based on the information in the CHANGE_SESSIONS record. If the new TERMINATION_COUNT is greater than 0, sessions have to be deactivated. Pending-active sessions are deactivated first followed by free sessions. If the TERMINATION_COUNT is still greater than 0, sessions will be deactivated later when they become free.

After pending-active and free sessions have been deactivated as required, additional sessions may be activated if the current session count (by polarity, i.e., CONWINNER or CONLOSER) is less than the minimum limits. This procedure may have to request both deactivation and activation of sessions if, for example, the total session limit remains constant, but the mix of first-speaker and bidder sessions changes.

INPUT: CHANGE_SESSIONS

OUTPUT: MODE.TERMINATION count set, waiting GET_SESSION records possibly rejected and destroyed, CHANGE_SESSIONS record destroyed

NOTES:
1. An implementation may choose not to deactivate pending-active sessions. If, however, the TERMINATION_COUNT is nonzero when the session becomes active, the session has to then be deactivated.
2. The MODE pending termination counts indicate the number of sessions that this LU has sent BIS on.

Referenced procedures, FSMs, and data structures:


```
If CHANGE_SESSIONS.RESPONSIBLE is YES then
   Get addressability to the MODE control block associated with CHANGE_SESSIONS.LU_NAME
    and CHANGE_SESSIONS.MODE_NAME.
   Set CONWINNER_COUNT to MODE.ACTIVE_CONWINNERS_COUNT + MODE.PENDING_CONWINNERS_COUNT.
   Set CONLOSER_COUNT to MODE.ACTIVE_CONLOSERS_COUNT + MODE.PENDING_CONLOSERS_COUNT.
   Set OLD_SESSION_LIMIT to MODE.SESSION_LIMIT - CHANGE_SESSIONS.DELTA.
   Set PLATEAU to
    min(MODE.ACTIVE_SESSION_COUNT + MODE.PENDING_SESSION_COUNT, OLD_SESSION_LIMIT).
   Set CONWINNER_INCREMENT to the maximum of (0, MODE.MIN_CONWINNERS_LIMIT - CONWINNER_COUNT).
   Set SESSION_DECREMENT to the maximum of (0, PLATEAU - MODE.SESSION_LIMIT).
   Set CONLOSER_INCREMENT to the maximum of (0, MODE.MIN_CONLOSERS_LIMIT - CONLOSER_COUNT).
   Set NEED_TO_ACTIVATE to CONWINNER_INCREMENT + CONLOSER_INCREMENT.
   Set ROOM_FOR_ACTIVATION to the maximum of (0, MODE.SESSION_LIMIT - PLATEAU).
   Set DECREMENT_FOR_POLARITY to the maximum of (0, NEED_TO_ACTIVATE - ROOM_FOR_ACTIVATION).
   Set MODE.TERMINATION_COUNT to MODE.TERMINATION_COUNT + SESSION_DECREMENT +
    DECREMENT_FOR_POLARITY.
   If MODE.TERMINATION_COUNT > 0 then
       Call DEACTIVATE_PENDING_SESSIONS(CHANGE_SESSIONS.LU_NAME, CHANGE_SESSIONS.MODE_NAME)
        (page 3-47, see Note 1).
   If MODE.TERMINATION_COUNT > 0 then
       Call DEACTIVATE_FREE_SESSIONS(CHANGE_SESSIONS.LU_NAME, CHANGE_SESSIONS.MODE_NAME)
        (page 3-46).
```

```
If MODE.SESSION_LIMIT = 0, and
 (MODE.DRAIN_SELF = NO or (MODE.ACTIVE_SESSION_COUNT -                (see Note 2)
 (MODE.PENDING_TERMINATION_CONWINNERS + MODE.PENDING_TERMINATION_CONLOSERS)=0)) then
   Do for each GET_SESSION request waiting for a session with (CHANGE_SESSIONS.LU_NAME,
    CHANGE_SESSIONS.MODE_NAME):
      Create a SESSION_ALLOCATED record with RETURN_CODE set to UNSUCCESSFUL_NO_RETRY
       and send it to the PS that made the request.
      Destroy the GET_SESSION request.
Call ACTIVATE_NEEDED_SESSIONS(CHANGE_SESSIONS.LU_NAME, CHANGE_SESSIONS.MODE_NAME) to
 activate new sessions if possible and if needed (page 3-24).
```

CHECK_FOR_BIS_REPLY

FUNCTION:	This procedure checks to see if a BIS_REPLY should be sent at the present time to respond to a received BIS_RQ.
INPUT:	HS_ID, the ID of the half-session that sent the BIS_RQ
OUTPUT:	BIS_REPLY to HS, or no output
NOTE:	BIS_REPLY is sent if there are no waiting GET_SESSION requests for the session.

Referenced procedures, FSMs, and data structures:

SEND_BIS_REPLY	page 3-67
HS_ID	page 3-91
GET_SESSION	page A-16
MODE	page A-3

```
Get addressability to the MODE control block associated with the LU name and mode
 name of the session identified by HS_ID.
If MODE.DRAIN_SELF = NO or there are no GET_SESSION records waiting for the LU name and
 mode name then
   If the session identified by HS_ID is free (between brackets) then
      Call SEND_BIS_REPLY(HS_ID) (page 3-67).
      Remove the session from the free-session pool.
```

COMPLETE_LUW_ID

FUNCTION: This procedure creates a new LUW instance and sequence number.

INPUT: TCB

OUTPUT: The LUW instance and sequence number set in the TCB, PREVIOUS_TIME reset

NOTES:

1. Bits 0-31 of the time value is the accurate local time (S/370 time-of-day clock format); the remaining bits, 32-47, may be used to provide uniqueness of the LUW instance field. Each LUW instance has a greater value than previously generated values.

2. If the value of the first 5 bytes found in the TIME variable is equal to the value of the last LUW instance (found in PREVIOUS_TIME), the value contained in bits 32 to 47 of the TIME variable is incremented by 1. Unless bits 32 to 47 contain all binary 1's, incrementing the TIME variable will create a value that can be used in the LUW instance field. If bits 32 to 47 contain all binary 1's, the incrementation will cause a wrap, creating an invalid value less than the previous time. Under this circumstance, the TIME variable will again be set to the local system time. Implementations should assign bits 32 to 47 based upon their clock data, when translating their clock into the 370 clock format, or should treat bits 32 to 47 as a counter to be incremented by one whenever the TIME variable is set to the local system time. The counter may be initialized to binary 0's. When all bits 32 to 47 are set (binary 1's) and the time function is called again, the counter should wrap (all bits 32 to 47 as binary 0) with no carry past bit 32.

Referenced procedures, FSMs, and data structures:

TCB	page A-9
PREVIOUS_TIME	page 3-92
TIME, see PREVIOUS_TIME	page 3-92

```
Repeat until a valid TIME is generated
   Set TIME variable to the local system time.
   Translate TIME variable to IBM S/370 time-of-day clock format.
    (Refer to SYSTEM/370 Principles of Operation, <GA22-7000>, for the defined format.)
   If TIME(bits 0 to 47) is less than or equal to PREVIOUS_TIME (Note 1)
    and the TIME value has not wrapped then
      Add binary 1 to the TIME(bits 32 to 47) value. (Note 2)
      If TIME(bits 32 to 47) has wrapped then
         TIME is not valid.
      Else
         TIME is valid.
   Else
      TIME is valid.
Set TCB.LUW_IDENTIFIER.LUW_INSTANCE to TIME(bits 0 to 47).
Set PREVIOUS_TIME to TIME(bits 0 to 47).
Set TCB.LUW_IDENTIFIER.LUW_SEQUENCE_NUMBER to 1.
```

CONNECT_RCB_AND_SCB

FUNCTION:	This procedure connects a PS and HS process, and informs HS when the connection is complete.
INPUT:	RCB_ID and HS_ID, the IDs of the RCB representing the conversation resource and the SCB representing the half-session
OUTPUT:	The RCB and SCB are updated; HS_PS_CONNECTED record is sent to HS.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
RCB	page A-6
SCB	page A-8
RCB_ID	page 3-91
HS_ID	page 3-91
HS_PS_CONNECTED	page A-18

```
Find the half-session (SCB) identified by HS_ID.
Find the conversation (RCB) identified by RCB_ID.
Set RCB.SESSION_IDENTIFIER to SCB.SESSION_IDENTIFIER.
Set SCB.BRACKET_ID to RCB.BRACKET_ID.
Create an HS_PS_CONNECTED record with BRACKET_ID set to RCB.BRACKET_ID
 and PS_ID equal to RCB.TCB_ID.
Send the record to HS.     (Chapter 6.0).
```

CREATE_RCB

FUNCTION:	This procedure handles the creation of new RCBs for outgoing ALLOCATE requests, for incoming Attaches see (PS_CREATION_PROC on page 3-55).
INPUT:	ALLOCATE_RCB and RCB_ALLOCATED. The RCB_ALLOCATED was created by ALLOCATE_RCB_PROC (page 3-26).
OUTPUT:	RCB_ALLOCATED with the RCB_ID field set to the ID of the new RCB, an RCB is created and initialized.
NOTE:	#FSM_RCB_STATUS is a generic FSM that can be either FSM_RCB_STATUS_FSP or FSM_RCB_STATUS_BIDDER, depending on whether the conversation resource is using a first-speaker or a bidder half-session. When a new RCB is created, it is not usually known which type of half-session will be available (except for ALLOCATE_RCB(IMMEDIATE), which must use a first-speaker half-session in order to be successful). Therefore, when the RCB is created, the FSM is initialized to FSM_RCB_STATUS_FSP, and is changed later if the conversation will be running on a bidder half-session. Until this determination is made, the state of the #FSM_RCB_STATUS remains FREE (01).

Referenced procedures, FSMs, and data structures:

FSM_RCB_STATUS_FSP	page 3-90
FSM_RCB_STATUS_BIDDER	page 3-89
ALLOCATE_RCB	page A-15
RCB_ALLOCATED	page A-21
TCB	page A-9
RCB	page A-6

Create RCB, initializing RCB_ID and BRACKET_ID to unique values, HS_ID to a null value, LU_NAME to ALLOCATE_RCB.LU_NAME, and MODE_NAME to ALLOCATE_RCB.MODE_NAME.
Copy TCB_ID, SYNC_LEVEL, and SECURITY_SELECT from the ALLOCATE_RCB record to the RCB.
Set RCB_ALLOCATED.RCB_ID to RCB.RCB_ID.
Set #FSM_RCB_STATUS = FSM_RCB_STATUS_FSP (page 3-90; see Note).
Call #FSM_RCB_STATUS(S, ALLOCATE_RCB, UNDEFINED) page 3-89).
Set RCB.CONVERSATION_CORRELATOR to a unique value, RCB.SESSION_IDENTIFIER to a null value, RCB.TP_NAME to the TRANSACTION_PROGRAM_NAME in the TCB specified by ALLOCATE_RCB.

CREATE_SCB

FUNCTION:	This procedure creates a new SCB based on the LU_NAME, MODE_NAME, and SESSION_INFORMATION arguments.
INPUT:	LU_NAME and MODE_NAME of the partner LU; and SESSION_INFORMATION, which describes the session attributes
OUTPUT:	A new SCB is created.

Referenced procedures, FSMs, and data structures:


```
Create an SCB, set SCB.HS_ID to SESSION_INFORMATION.HS_ID, SCB.LU_NAME to
 LU_NAME, SCB.MODE_NAME to MODE_NAME, SCB.RCB_ID to a null value,
 SCB.SESSION_IDENTIFIER to SESSION_INFORMATION.SESSION_IDENTIFIER,
 SCB.SEND_RU_SIZE to SESSION_INFORMATION.SEND_RU_SIZE,
 SCB.LIMITED_BUFFER_POOL_ID to SESSION_INFORMATION.LIMITED_BUFFER_POOL_ID,
 SCB.PERMANENT_BUFFER_POOL_ID to SESSION_INFORMATION.PERMANENT_BUFFER_POOL_ID,
 SCB.BRACKET_ID to null, SCB.RANDOM_DATA to SESSION_INFORMATION.RANDOM_DATA,
 and SCB.RTR_OWED to FALSE.
Select based on SESSION_INFORMATION.BRACKET_TYPE:
If the half-session is a first-speaker then
   Assign finite-state machines to be used by setting
    #FSM_BIS to FSM_BIS_FSP (page 3-88)
    and #FSM_SCB_STATUS to FSM_SCB_STATUS_FSP (page 3-86).
   Set SCB.FIRST_SPEAKER to TRUE.
Else (bidder session)
   Assign finite-state machines to be used by setting
    #FSM_BIS to FSM_BIS_BIDDER (page 3-87)
    and #FSM_SCB_STATUS to FSM_SCB_STATUS_BIDDER (page 3-85).
   Set SCB.FIRST_SPEAKER to FALSE.
```

CREATE_TCB_AND_PS

FUNCTION:	This procedure creates a TCB and PS as a result of START_TP processing.
INPUT:	The START_TP request record, a non-null TRANSACTION_PROGRAM record
OUTPUT:	A new TCB and PS, and the START_TP with the new TCB_ID. All shared TCB fields are initialized in this procedure. When a PS creation failure occurs, the TCB is destroyed, START_TP.TCB_ID is set to null, and the failure is logged.

Referenced procedures, FSMs, and data structures:

PS	page 5.0-8
TCB	page A-9
START_TP	page A-19
LUCB	page A-1
PS_CREATE_PARMS	page A-27
TRANSACTION_PROGRAM	page A-5
COMPLETE_LUW_ID	page 3-41

```
Create a TCB.
Set TCB.TCB_ID to a unique value.
Set TCB.TRANSACTION_PROGRAM_NAME to START_TP.TARGET_TP_NAME.
Set TCB.OWN_LU_ID to LUCB.LU_ID.
Set TCB.LUW_IDENTIFIER.FULLY_QUALIFIED_LU_NAME to START_TP.FULLY_QUALIFIED_LU_NAME.
Call COMPLETE_LUW_ID(TCB) (page 3-41).
Set TCB.CONTROLLING_COMPONENT to TP.
If a user ID is present in the START_TP then
   Set TCB.INITIATING_SECURITY.USERID to START_TP.SECURITY.USERID.
Else
   Set TCB.INITIATING_SECURITY.USERID to null.
If a profile is present in the START_TP then
   Set TCB.INITIATING_SECURITY.PROFILE to START_TP.SECURITY.PROFILE.
Else
   Set TCB.PROFILE to null.
Create PS_CREATE_PARMS initializing the fields to the addresses and IDs of
 of the data structures to which PS requires access (See page A-27).
Create a new PS process with the PS_CREATE_PARMS as parameter (Chapter 5.0).
If PS was successfully created then
   Increment TRANSACTION_PROGRAM.INSTANCE_COUNT by 1.
   Set START_TP.TCB_ID to TCB.TCB_ID.
Else
   Destroy the TCB.
   Log the PS creation failure to the system log.
   Set START_TP.TCB_ID to a null value.
```

DEACTIVATE_FREE_SESSIONS

FUNCTION:	This procedure requests deactivation of free sessions between this LU and the partner LU identified by (LU_NAME, MODE_NAME). Deactivations are requested until either all free sessions have had deactivation requested, or this LU is not responsible for any more deactivations.
INPUT:	The LU_NAME of the partner LU and the MODE_NAME of the sessions to be deactivated
OUTPUT:	Zero or more sessions are removed from the free-session pool.
NOTE:	First-speaker sessions are deactivated before bidder sessions.

Referenced procedures, FSMs, and data structures:

SESSION_DEACTIVATION_POLARITY	page 3-74
SEND_BIS	page 3-66
SCB	page A-8
LU_NAME	page 3-91
MODE_NAME	page 3-91

```
Do while there exists a free session of a polarity matching that returned by
 SESSION_DEACTIVATION_POLARITY(LU_NAME, MODE_NAME) (page 3-74):
 (If SESSION_DEACTIVATION_POLARITY returns EITHER, a first-speaker session is
 deactivated in preference to a bidder session.)
   Find the session's corresponding SCB.
   Remove the session from the free-session pool.
   Call SEND_BIS(SCB.HS_ID) (page 3-66).
```

DEACTIVATE_PENDING_SESSIONS

FUNCTION:	This procedure requests deactivation of pending-active sessions between this LU and the partner LU identified by (LU_NAME, MODE_NAME). Deactivations are requested until either all pending-active sessions have had deactivation requested, or this LU is not responsible for any more deactivations.
INPUT:	LU_NAME of the partner LU and the MODE_NAME of the sessions to be deactivated
OUTPUT:	MODE termination count decremented, queued RM_ACTIVATE_SESSION requests destroyed, RM_SESSION_ACTIVATED records created and sent to PS
NOTE:	Deactivation requests for pending-active sessions are type Cleanup. This addresses the possiblity that the session may already be established without RM yet knowing (via ACTIVATE_SESSION_RSP). Under this circumstance, the type of UNBIND sent should be Cleanup.

Referenced procedures, FSMs, and data structures:

PS	page 5.0-8
SESSION_DEACTIVATION_POLARITY	page 3-74
SEND_DEACTIVATE_SESSION	page 3-68
PENDING_ACTIVATION, see ACTIVATE_SESSION	page A-20
LU_NAME	page 3-91
MODE_NAME	page 3-91
MODE	page A-3
RM_SESSION_ACTIVATED	page A-22
RM_ACTIVATE_SESSION	page A-16


```
Get addressability to the MODE control block associated with (LU_NAME, MODE_NAME).
Do while there are PENDING_ACTIVATION records for first-speaker sessions
 for (LU_NAME, MODE_NAME), and SESSION_DEACTIVATION_POLARITY(LU_NAME,MODE_NAME)
 (page 3-74) indicates FIRST_SPEAKER or EITHER:
   Call SEND_DEACTIVATE_SESSION(PENDING, PENDING_ACTIVATION.CORRELATOR, CLEANUP, X'08A00002')
    (page 3-68).
   Decrement MODE.TERMINATION_COUNT by 1.
Do while there are PENDING_ACTIVATION records for bidder sessions
 for (LU_NAME, MODE_NAME), and SESSION_DEACTIVATION_POLARITY(LU_NAME,MODE_NAME)
 (page 3-74) indicates BIDDER or EITHER:
   Call SEND_DEACTIVATE_SESSION(PENDING, PENDING_ACTIVATION.CORRELATOR, CLEANUP, X'08A00002')
    (page 3-68).
   Decrement MODE.TERMINATION_COUNT by 1.
Do while the number of pending CNOS operator activation requests
 for (LU_NAME,MODE_NAME) > MODE.PENDING_SESSION_COUNT:
   Find a pending operator RM_ACTIVATE_SESSION request for (LU_NAME,MODE_NAME).
   Create an RM_SESSION_ACTIVATED with RETURN_CODE equal to LU_MODE_SESSION_LIMIT_EXCEEDED
    and send it to the PS that sent the request.
   Discard the pending operator RM_ACTIVATE_SESSION request.
```

DEQUEUE_WAITING_REQUEST

FUNCTION:	This procedure checks to see if any eligible GET_SESSION requests are waiting to be serviced. This procedure dequeues the first eligible request and invokes GET_SESSION_PROC (page 3-52) to process the request.
INPUT:	HS_ID, the ID of the half-session that sent the request
OUTPUT:	GET_SESSION_PROC is invoked to process the waiting request and the waiting request is removed from the waiting request list.

Referenced procedures, FSMs, and data structures:

GET_SESSION_PROC	page 3-52
GET_SESSION	page A-16
MODE	page A-3
HS_ID	page 3-91

```
Get addressability to the MODE of the session identified by HS_ID.
If there is a waiting GET_SESSION request for a session on this MODE then
   Remove the GET_SESSION from the waiting request queue.
   Call GET_SESSION_PROC(GET_SESSION) (page 3-52) to service the request.
```

FIRST_SPEAKER_PROC

FUNCTION:	This procedure handles the allocation processing for a first-speaker half-session. This procedure causes the SCB associated with the first-speaker half-session and the RCB of the conversation for which the session was requested to be connected to each other. RM creates a SESSION_ALLOCATED record, which it sends to PS to inform PS that a session has been successfully allocated.
INPUT:	GET_SESSION and HS_ID, the ID of the first-speaker half-session that was chosen by GET_SESSION_PROC (page 3-52)
OUTPUT:	SESSION_ALLOCATED to PS

Referenced procedures, FSMs, and data structures:

PS	page 5.0-8
SET_RCB_AND_SCB_FIELDS	page 3-75
CONNECT_RCB_AND_SCB	page 3-42
GET_SESSION	page A-16
SESSION_ALLOCATED	page A-22
HS_ID	page 3-91
SCB	page A-8

```
Call SET_RCB_AND_SCB_FIELDS(GET_SESSION.RCB_ID, HS_ID) (page 3-75).
Call CONNECT_RCB_AND_SCB(GET_SESSION.RCB_ID, HS_ID) (page 3-42).
Create a SESSION_ALLOCATED record with RETURN_CODE equal to OK.
Get addressability to the SCB identified by HS_ID.
Set SEND_RU_SIZE, LIMITED_BUFFER_POOL_ID, and PERMANENT_BUFFER_POOL_ID
 in the SESSION_ALLOCATED record to the corresponding fields in the SCB.
Set SESSION_ALLOCATED.IN_CONVERSATION to NO.
Send the record to PS.
```

FREE_SESSION_PROC

FUNCTION: This procedure handles the processing that occurs when a session becomes free.

This procedure first checks to see if a Bid is outstanding on this session. If so, the session is not returned to the free-session pool. If not, the procedure checks to see if an RTR_RQ or a BIS request or reply is to be sent. If either RTR_RQ or BIS is sent, the session is not returned to the free-session pool. If neither BIS nor RTR is sent, the free-session is returned to the free-session pool, and a waiting session allocation request (if any) is serviced.

INPUT: FREE_SESSION

OUTPUT: BRACKET_FREED, BIS_RQ, BIS_REPLY, or RTR_RQ to HS; or GET_SESSION to GET_SESSION_PROC (page 3-52); or no output

NOTES:

1. Upon receipt of DEALLOCATE_RCB (a request to deallocate the conversation) from PS, RM destroys the RCB associated with the conversation previously using the half-session (see PROCESS_PS_TO_RM_RECORD on page 3-22). If the search for the RCB identified by the SCB.RCB_ID fails, PS has already deallocated the conversation. When this occurs, RM sends BRACKET_FREED to the half-session.

2. If an RTR is owed on this session (either the partner LU owes RTR to the local LU or the local LU owes RTR to the partner), the bidder has to wait for an RTR from the first-speaker before it can again bid for the session. Therefore, the deallocated bidder session is not returned to the free-session pool and a waiting request is not serviced.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
DEQUEUE_WAITING_REQUEST	page 3-48
SHOULD_SEND_BIS	page 3-76
SEND_BIS	page 3-66
SEND_DEACTIVATE_SESSION	page 3-68
FSM_SCB_STATUS_BIDDER	page 3-85
FSM_SCB_STATUS_FSP	page 3-86
FSM_RCB_STATUS_FSP	page 3-90
FSM_RCB_STATUS_BIDDER	page 3-89
FSM_BIS_BIDDER	page 3-87
FSM_BIS_FSP	page 3-88
BRACKET_FREED	page A-18
FREE_SESSION	page A-12
SCB	page A-8
RCB	page A-6
RTR_RQ	page A-12
GET_SESSION	page A-16

```
Find the SCB associated with the session identified by FREE_SESSION.HS_ID.
Find the RCB identified by the SCB.RCB_ID.
If the RCB cannot be found  (Note 1) then
   Create a BRACKET_FREED record, initializing the BRACKET_ID to SCB.BRACKET_ID.
   Send the BRACKET_FREED record to the HS that sent the FREE_SESSION.
Set SCB.RCB_ID to a null value.
If the state of #FSM_SCB_STATUS is PENDING_FMH12 then (page 3-84).
   Call SEND_DEACTIVATE_SESSION(ACTIVE, SCB.HS_ID, ABNORMAL, X'080F6051')  (page 3-68).
   Optionally log an error in the system log.
   Return to the calling routine.
Else
   Call #FSM_SCB_STATUS(R, FREE_SESSION, UNDEFINED) (page 3-84).
If there is an RCB for which the state of #FSM_RCB_STATUS is PENDING_SCB,
 and RCB.HS_ID = SCB.HS_ID then
   Take no action and return to the calling routine (a Bid is pending).
If SCB.RTR_OWED is TRUE then
   If this is a first-speaker session (i.e., this LU owes RTR) then
      If there are no waiting GET_SESSION requests for a session
       of the partner LU and of the mode of the free session then
         If RTR is to be sent now (implementation-defined choice) then
            Send RTR_RQ to HS (Chapter 6.0).
            Set SCB.RTR_OWED to false.
         Else
            Return the session to the free-session pool.
         Return to the calling routine.
   Else (bidder session; i.e., other LU owes RTR)
      Take no action and return to the calling routine (Note 2).
Call SHOULD_SEND_BIS(SCB.HS_ID) (page 3-76) to determine
 whether BIS should be sent now.
If BIS should be sent now then
   Call SEND_BIS(SCB.HS_ID) (page 3-66).
If the state of #FSM_BIS (page 3-87) is BIS_SENT or CLOSED then
   Take no action and return to the calling routine (BIS has been sent).
Else (the session is available for reuse)
   Return the session to the free-session pool.
   If there are waiting GET_SESSION requests for this (partner LU, mode) then
      Call DEQUEUE_WAITING_REQUEST(SCB.HS_ID) (page 3-48).
Destroy the FREE_SESSION record.
```

GET_SESSION_PROC

FUNCTION:		This procedure handles the allocation of half-sessions to be used by conversation resources. The procedure checks for an available half-session and calls the appropriate procedure, depending upon whether the half-session found was a first-speaker or a bidder half-session. If no half-sessions are available and the current session limit has not been reached, SEND_ACTIVATE_SESSION is called, which requests that SM activate a new session.
INPUT:		GET_SESSION
OUTPUT:		See called procedures for output.
NOTES:	1.	RM does the following: attempts to service the request with a first-speaker half-session; if none is available, RM attempts to service the request with a bidder half-session; failing that, RM requests the session manager to activate a new session if the current session limit has not been reached. If a first-speaker half-session is available, that session is used to service the session request. If no first-speaker half-sessions are available, an implementation can choose to service the request with a free bidder half-session, activate a new first-speaker half-session, or both of the above. An implementation could, for example, choose to implement the following order: choose a free first-speaker half-session; request a new first-speaker half-session be activated; and, finally, choose a free bidder half-session. (Another possibility is that an implementation could service the session request with a bidder half-session, if no first-speaker half-sessions are available, but at the same time ask that a new first-speaker half-session be activated.) However, if there are no free first-speaker half-sessions and the session limit for the desired (LU name, mode name) pair has been reached, the session request is serviced with a bidder half-session, if available. If a bidder half-session is available, an implementation does not wait for a first-speaker half-session to become free before servicing the session request.
	2.	A mode is closed if no sessions are active for the mode name and a session cannot be activated without operator intervention (e.g., the operator must increase the session limit above 0). In this case, the GET_SESSION request is rejected with a return code of UNSUCCESSFUL_NO_RETRY.

Referenced procedures, FSMs, and data structures:

PS	page 5.0-8
FIRST_SPEAKER_PROC	page 3-49
BIDDER_PROC	page 3-37
SESSION_ACTIVATION_POLARITY	page 3-71
SEND_ACTIVATE_SESSION	page 3-65
SEND_BIS	page 3-66
GET_SESSION	page A-16
RCB	page A-6
SCB	page A-8
PARTNER_LU	page A-2
SESSION_ALLOCATED	page A-22

```
Find the RCB with RCB_ID equal to GET_SESSION.RCB_ID.
Find the PARTNER_LU identified by the RCB.LU_NAME.
Find the MODE identified by the RCB.LU_NAME and RCB.MODE_NAME.
If the mode is closed then (see Note 2)
   Create and send a SESSION_ALLOCATED record with a return code of UNSUCCESSFUL_NO_RETRY to
    PS (Chapter 5.1).
   Destroy the GET_SESSION record.
Else
   If the (RCB.LU_NAME, RCB.MODE_NAME) sessions do not support the
    requested sync level then
      Create and send SESSION_ALLOCATED record with a return code of SYNC_LEVEL_NOT_SUPPORTED.
      Destroy the GET_SESSION record.
   Else
      If the (RCB.LU_NAME, RCB.MODE_NAME) sessions do not support the
       requested security level then
         Downgrade the SECURITY_SELECT field of the RCB by setting it to NONE.
      If a free session exists for RCB.LU_NAME and RCB.MODE_NAME then
         Find the SCB associated with the free session.
         If SCB.FIRST_SPEAKER is YES then
            Call FIRST_SPEAKER_PROC(GET_SESSION, SCB.HS_ID) (page 3-49).
            Destroy the GET_SESSION record.
         Else (bidder half-session)
            Call BIDDER_PROC(GET_SESSION, SCB.HS_ID) (page 3-37).
         Remove the session from the free-session pool.
      Else (no free session exists)
         If the number of waiting GET_SESSION requests queued for sessions
          on this mode equals or exeeds the number of pending active session requests then
            Call SESSION_ACTIVATION_POLARITY(RCB.LU_NAME, RCB.MODE_NAME)  (page 3-71)
             to determine the polarity of the next activated session (if any).
            Select based on session activation polarity:
                   When NONE (no new sessions can be activated)
                      If PARTNER_LU.PARALLEL_SESSSION is NOT_SUPPORTED and an active session
                       for another mode exists (other than the mode requested by the
                       GET_SESSION) then
                          If the session is free then
                             Get addressability to the SCB of the free session.
                             Remove the session from the free session pool.
                             Call SEND_BIS(SCB.HS_ID) (page 3-66).
                   When FIRST_SPEAKER
                      Call SEND_ACTIVATE_SESSION(RCB.LU_NAME, RCB.MODE_NAME,
                       FIRST_SPEAKER) (page 3-65).
                   When BIDDER
                      Call SEND_ACTIVATE_SESSION(RCB.LU_NAME, RCB.MODE_NAME,
                       BIDDER) (page 3-65).
         Queue the GET_SESSION request to await a session.
```

PS_ABEND_PROC

FUNCTION:	This procedure recovers from a PS abend. The procedure deletes the control blocks, data structure entries, and counts associated with the abended PS.
INPUT:	ABEND_NOTIFICATION
OUTPUT:	Queued GET_SESSION requests from the abended PS are destroyed, RCB control blocks and the TCB control block associated with the abended PS are destroyed, the sessions that the PS was using or bidding on are unbound, the TP instance count associated with the abended PS is decremented, queued Attach and START_TP requests for the TP are recycled by updating associated control blocks and recalling the appropriate procedure.
NOTE:	In order to recall ATTACH_PROC the state of the SCB FSM must be PENDING_ATTACH. Both FSM calls are needed to change the state of the FSM from IN_USE to PENDING_ATTACH.

Referenced procedures, FSMs, and data structures:

PS	page 5.0-8
SM	page 4-48
FSM_SCB_STATUS_BIDDER	page 3-85
FSM_SCB_STATUS_FSP	page 3-86
FSM_RCB_STATUS_FSP	page 3-90
FSM_RCB_STATUS_BIDDER	page 3-89
SESSION_DEACTIVATED_PROC	page 3-72
ATTACH_PROC	page 3-30
START_TP_PROC	page 3-77
TCB	page A-9
SCB	page A-8
ABEND_NOTIFICATION	page A-25
RCB	page A-6
DEACTIVATE_SESSION	page A-21
TRANSACTION_PROGRAM	page A-5
MU	page A-29
START_TP	page A-19
SESSION_DEACTIVATED	page A-14
MODE	page A-3
GET_SESSION	page A-16

```
Find the TCB representing the abended PS.
Destroy all queued GET_SESSION requests from the abended PS.
If the TCB is found
   Do for each RCB associated with the abended PS:
      If the state of the associated FSM_RCB_STATUS is FREE then
         Find the MODE that corresponds to RCB.LU_NAME, RCB.MODE_NAME

      If the state of the associated FSM_RCB_STATUS is IN_USE or PENDING_SCB then
         Find the SCB with HS_ID equal to RCB.HS_ID.
         Destroy the RCB.
         If the SCB is found then
            Create a DEACTIVATE_SESSION with STATUS set to ACTIVE,
             HS_ID set to SCB.HS_ID, TYPE set to ABNORMAL, and
             SENSE_CODE set to X'08640000'.
            Send the DEACTIVATE_SESSION record to SM.
            Create a SESSION_DEACTIVATED with HS_ID set to SCB.HS_ID,
             REASON set to ABNORMAL_RETRY, and SENSE_CODE set to X'08640000'.
            Call SESSION_DEACTIVATED_PROC(SESSION_DEACTIVATED) (page 3-72).
```

```
Find the TRANSACTION_PROGRAM where TRANSACTION_PROGRAM.TRANSACTION_PROGRAM_NAME equals
 the TCB.TRANSACTION_PROGRAM_NAME of the abended PS.
If the TRANSACTION_PROGRAM is found then
   Decrement the TRANSACTION_PROGRAM.INSTANCE_COUNT by 1.
   If there is an initiation request (Attach or START_TP) queued for the transaction
    program named by TRANSACTION_PROGRAM.TRANSACTION_PROGRAM_NAME and the
    TRANSACTION_PROGRAM.INSTANCE COUNT is less than the TRANSACTION_PROGRAM.INSTANCE_LIMIT
    then
      Remove the initiation request from the queue.
      If the initiation request is an MU (containing an Attach) then
         Find the RCB with RCB_ID equal to MU.RM_TO_PS.RCB_ID.
         Set MU.HS_TO_RM.HS_ID to the RCB's HS_ID.
         Find the SCB with the HS_ID equal to the RCB's HS_ID.
         Call FSM_SCB_STATUS(R, FREE_SESSION, UNDEFINED).
         Call FSM_SCB_STATUS(R, BID, UNDEFINED) (See Note).
         Set the SCB's BRACKET_ID and RCB_ID to null.
         Destroy the RCB.
         Call ATTACH_PROC(MU) (page 3-30)
      If the queued initiation request is a START_TP then
         Call START_TP_PROC(START_TP) (page 3-77).
Log the abend to the system log.
Destroy the TCB of the abended PS.
```

PS_CREATION_PROC

FUNCTION:	This procedure creates a new PS process. This procedure is called upon receipt of an Attach. Along with creating the PS process, it also creates a new TCB and RCB. It returns to the calling procedure the IDs of the newly created TCB and RCB, which the calling procedure will send to PS along with the received MU containing an Attach.
INPUT:	An MU containing an Attach, variables in which the TCB_ID and RCB_ID will be returned, and the pointer to the TRANSACTION_PROGRAM structure that represents the target transaction program
OUTPUT:	A TCB, RCB, and new PS process created and initialized, CREATE_RC (creation return code, SUCCESS or FAILURE) is set and returned to the calling proceedure, if the creation fails, the TCB and RCB destroyed, if the creation succeeds, the transaction program instance count incremented

Referenced procedures, FSMs, and data structures:

PS	page 5.0-8
COMPLETE_LUW_ID	page 3-41
FSM_RCB_STATUS_FSP	page 3-90
FSM_RCB_STATUS_BIDDER	page 3-89
TRANSACTION_PROGRAM	page A-5
PS_CREATE_PARMS	page A-27
MU	page A-29
TCB_ID	page 3-91
RCB_ID	page 3-91
TCB	page A-9
SCB	page A-8
RCB	page A-6
LUCB	page A-1

```
Create a TCB with a unique TCB_ID, initializing TRANSACTION_PROGRAM_NAME to the
 transaction program name contained in the Attach, CONTROLLING_COMPONENT to TP
 and OWN_LU_ID to LUCB.LU_ID.
If the Attach contains a user ID then
   Set TCB.INITIATING_SECURITY.USERID to the user ID contained in the Attach.
Else
   Set TCB.INITIATING_SECURITY.USERID to null.
If the Attach contains a profile then
   Set TCB.INITIATING_SECURITY.PROFILE to the profile contained in the Attach.
Else
   Set TCB.INITIATING_SECURITY.PROFILE to null.
```

```
If the Attach contains a logical-unit-of-work ID then
   Save the logical-unit-of-work ID from the Attach in the TCB.
Else
   Set the TCB.LUW_IDENITFIER.FULLY_QUALIFIED_LU_NAME to the LUCB.FULLY_QUALIFIED_LU_NAME.
   Call COMPLETE_LUW_ID(TCB)  (page 3-41).
Find the SCB identified by MU.HS_TO_RM.HS_ID.
Create an RCB with a unique RCB_ID, initializing RCB.TCB_ID to TCB.TCB_ID,
 RCB.LU_NAME to SCB.LU_NAME, RCB.MODE_NAME to SCB.MODE_NAME,
 TP_NAME to the transaction program name contained in the Attach,
 RCB.BRACKET_ID to a unique value, RCB.SYNC_LEVEL to the sync level of the Attach,
 and RCB.HS_TO_PS_BUFFER_LIST to empty.
If a conversation correlator is present in the Attach then
   Set the RCB.CONVERSATION_CORRELATOR to the conversation correlator in the Attach.
Else
   Set the RCB.CONVERSATION_CORRELATOR to null.
If the session is a first speaker then
   Set #FSM_RCB_STATUS to FSM_RCB_STATUS_FSP (page 3-90).
Else
   Set #FSM_RCB_STATUS to FSM_RCB_STATUS_BIDDER (page 3-89).
Call #FSM_RCB_STATUS(R, ATTACH, HS) (page 3-89).
Set RCB.HS_ID to MU.HS_TO_RM.HS_ID.
Create PS_CREATE_PARMS initializing the fields to the addresses and IDs of
 of the data structures to which PS requires access (See page A-27).
Create a new PS process with the PS_CREATE_PARMS as a parameter
If PS was successfully created then
   Set CREATE_RC to SUCCESS.
   If the pointer to TRANSACTION_PROGRAM is a non-null value then
      Increment TRANSACTION_PROGRAM.INSTANCE_COUNT by 1.
Else
   Set CREATE_RC to FAILURE.
   Destroy the TCB and RCB.
```

PS_TERMINATION_PROC

FUNCTION:	This procedure handles the termination of a PS process. If there are no queued Attach or START_TP requests for this TP, the procedure destroys the PS process and discards the TCB corresponding to the PS being destroyed. If there are waiting requests for the TP-PS process, PS is normally not terminated, instead the waiting request is sent to the PS instance requesting termination.
INPUT:	TERMINATE_PS
OUTPUT:	The PS process is destroyed , or an MU (containing an FMH-5) is sent to PS and HS_PS_CONNECTED is sent to HS, or a START_TP record is sent to the PS process
NOTES:	1. TRANSACTION_PROGRAM will not exist when the PS instance was brought up to reject an Attach that specified an unknown transaction program name. Under this circumstance the instance count for a transaction program has no meaning. 2. The TP instance count may exceed the instance limit when a PS process is brought up to reject an Attach that contained an error (except as noted above). When the instance limit is exceeded, the PS process is terminated regardless of any queued requests.

Referenced procedures, FSMs, and data structures:

PS	page 5.0-8
HS	page 6.0-3
TERMINATE_PS	page A-17
LUCB	page A-1
TRANSACTION_PROGRAM	page A-5
HS_PS_CONNECTED	page A-18
MU	page A-29
START_TP	page A-19
START_TP_REPLY	page A-20
TCB	page A-9
RCB	page A-6
COMPLETE_LUW_ID	page 3-41

```
Find the TCB and TRANSACTION_PROGRAM corresponding to the PS being destroyed
If a TRANSACTION_PROGRAM is found (Note 1) and
 if there are queued waiting initiation requests for the TRANSACTION_PROGRAM and
 the TRANSACTION_PROGRAM.INSTANCE_LIMIT ≥ TRANSACTION_PROGRAM.INSTANCE COUNT then (Note 2)
   Select based on the first queued request's record type:
      When MU (MU containing an Attach)
         Set the MU.RM_TO_PS.TCB_ID to TCB.TCB_ID.
         Set the TCB.CONTROLLING_COMPONENT to TP.
         If the security subfields are present in the Attach then
            Set the TCB.INITIATING_SECURITY fields (PROFILE and USERID) to the values
             contained in the corresponding fields of the Attach.
         If an LUW_IDENTIFIER is present in the Attach then
            Set TCB.LUW_IDENTIFIER to the corresponding Attach field.
         Else  (LUW_IDENTIFIER not in Attach)
            Set the TCB.LUW_IDENTIFIER.FULLY_QUALIFIED_LU_NAME to
             LUCB.FULLY_QUALIFIED_LU_NAME.
            Call COMPLETE_LUW_ID(TCB) (page 3-41).
         Find the RCB with RCB_ID equal to MU.RM_TO_PS.RCB_ID.
         Set RCB.TCB_ID to TCB.TCB_ID.
         Create an HS_PS_CONNECTED record with BRACKET_ID set to RCB.BRACKET_ID
          and PS_ID set to RCB.TCB_ID.
         Send the HS_PS_CONNECTED record to HS.
         Send the MU to PS.
         If the send to PS fails then
            Call buffer manager (FREE_BUFFER, buffer address) to release
             the buffer containing MU.  ("Appendix B. Buffer Manager")
```

```
            When START_TP
               Set START_TP.TCB_ID to TCB.TCB_ID.
               Set the TCB.LUW_IDENTIFIER.FULLY_QUALIFIED_LU_NAME to
                START_TP.FULLY_QUALIFIED_LU_NAME.
               Call COMPLETE_LUW_ID(TCB) (page 3-41).
               Set the TCB.CONTROLLING_COMPONENT to TP.
               If the START_TP.SECURITY_SELECT is PGM then
                  Set the TCB.INITIATING_SECURITY fields (PROFILE and USERID) to the values
                   contained in the corresponding fields of the START_TP.
               Else
                  Set TCB.INITIATING_SECURITY fields to null.
               If START_TP.REPLY equals YES then
                  Create a START_TP_REPLY with a RESPONSE_CODE of OK
                   and a TCB_ID equal to START_TP.TCB_ID.
                  Send the START_TP_REPLY to the process that issued the
                   START_TP request.
      Else
         Decrement the INSTANCE_COUNT of the TRANSACTION_PROGRAM (corresponding
          to the PS instance that is being destroyed) by 1.
         Destroy the TCB and destroy the PS process corresponding to TERMINATE_PS.TCB_ID.
      Destroy the TERMINATE_PS record.
```

PURGE_QUEUED_REQUESTS

FUNCTION:	This procedure purges Attach and START_TP requests queued for a TP-PS process that currently has no instances running (possibly they abended) and none can be created.
INPUT:	TRANSACTION_PROGRAM
OUTPUT:	Queued Attachs and related RCBs destroyed, associated sessions unbound, queued START_TPs destroyed, TP initiating process notified

Referenced procedures, FSMs, and data structures:

SM	page 4-48
START_TP_REPLY	page A-20
TRANSACTION_PROGRAM	page A-5
MU	page A-29
START_TP	page A-19
RCB	page A-6
SCB	page A-8
DEACTIVATE_SESSION	page A-21
SESSION_DEACTIVATED	page A-14
SESSION_DEACTIVATED_PROC	page 3-72


```
If the pointer to the TRANSACTION_PROGRAM record is not null then
   Do while there are waiting initiation requests for the transaction program identified
    by TRANSACTION_PROGRAM:
      Select based on the first queued request's record type:
         When MU (MU containing Attach)
            Find the RCB identified by MU.RM_TO_PS.RCB_ID.
            Find the SCB identified by RCB.HS_ID.
            Destroy the RCB.
            Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
             containing MU (Appendix B).
            If the SCB is found then
               Create a DEACTIVATE_SESSION record initializing STATUS to ACTIVE,
                HS_ID to SCB.HS_ID, TYPE to ABNORMAL, and SENSE_CODE to X'08640000'.
               Send the DEACTIVATE_SESSION record to SM.
               Create a SESSION_DEACTIVATED record initializing HS_ID to SCB.HS_ID,
                REASON to ABNORMAL_RETRY, and SENSE_CODE to X'08640000'.
               Call SESSION_DEACTIVATED_PROC(SESSION_DEACTIVATED) (page 3-72)
         When START_TP
            If START_TP.REPLY is YES then
               Create a START_TP_REPLY record initializing RESPONSE_CODE to
                PS_CREATION_FAILURE and TCB_ID to a null value.
               Send the START_TP_REPLY to the initiating process.
            Destroy the START_TP record.
```

QUEUE_ATTACH_PROC

FUNCTION:	For an Attach that cannot be immediately satisfied because it is to a limited instance TP that is currently in use, this procedure creates and initializes an RCB and queues the Attach request.
INPUT:	An MU containing an FMH-5 (Attach)
OUTPUT:	A newly created RCB and the MU placed on a queue waiting for a TP to become available

Referenced procedures, FSMs, and data structures:

FSM_RCB_STATUS_FSP	page 3-90
FSM_RCB_STATUS_BIDDER	page 3-89
FSM_SCB_STATUS_BIDDER	page 3-85
FSM_SCB_STATUS_FSP	page 3-86
MU	page A-29
RCB	page A-6
SCB	page A-8

```
Create an RCB with a unique RCB_ID, setting the TCB_ID to null,
 the TP_NAME to the transaction program name contained in the Attach,
 HS_ID to MU.HS_TO_RM.HS_ID, BRACKET_ID to a unique value,
 SYNC_LEVEL to the sync level of the Attach
 and the HS_TO_PS_BUFFER_LIST to empty.
If there is a conversation correlator in the Attach then
   Set the RCB.CONVERSATION_CORRELATOR to the conversation correlator in the Attach.
Else
   Set the RCB.CONVERSATION_CORRELATOR to null.
Find the SCB identified by MU.HS_TO_RM.HS_ID.
Set RCB.LU_NAME to SCB.LU_NAME and RCB.MODE_NAME to SCB.MODE_NAME.
If the session is a first speaker then
   Set #FSM_RCB_STATUS to FSM_RCB_STATUS_FSP (page 3-90).
Else
   Set #FSM_RCB_STATUS to FSM_RCB_STATUS_BIDDER (page 3-89).
Call #FSM_RCB_STATUS(R, ATTACH, HS) (page 3-90).
Set SCB.BRACKET_ID to RCB.BRACKET_ID.
Set SCB.RCB_ID to RCB.RCB_ID.
Set RCB.SESSION_IDENTIFIER TO SCB.SESSION_IDENTIFIER.
Call #FSM_SCB_STATUS(R, ATTACH, UNDEFINED) (page 3-84).
Set the MU.RM_TO_PS fields as follows:  RCB_ID to RCB.RCB_ID,
 SEND_RU_SIZE to SCB.SEND_RU_SIZE, LIMITED_BUFFER_POOL_ID to SCB.LIMITED_BUFFER_POOL_ID,
 PERMANENT_BUFFER_POOL_ID to SCB.PERMANENT_BUFFER_POOL_ID, and SENSE_CODE to X'00000000'
 (waiting Attach record has passed all RM error checks).
Queue the Attach MU to await the freeing of an active target TP-PS instance.
```

RM_ACTIVATE_SESSION_PROC

FUNCTION:	This procedure processes the RM_ACTIVATE_SESSION record. An RM_ACTIVATE_SESSION record is sent to RM by PS.COPR (Chapter 5.4) when the control operator issues an ACTIVATE_SESSION command. The command directs RM to activate a new session to the partner LU identified by LU_NAME with the mode specified by MODE_NAME. RM replies to the RM_ACTIVATE_SESSION record with an RM_SESSION_ACTIVATED record. The RETURN_CODE field of RM_SESSION_ACTIVATED indicates the success or failure of the session activation.
INPUT:	RM_ACTIVATE_SESSION
OUTPUT:	ACTIVATE_SESSION to SM, or RM_SESSION_ACTIVATED with RETURN_CODE = LU_MODE_SESSION_LIMIT_EXCEEDED to PS, or RM_ACTIVATE_SESSION saved as a pending operator activation request

Referenced procedures, FSMs, and data structures:

PS	page 5.0-8
SESSION_ACTIVATION_POLARITY	page 3-71
SEND_ACTIVATE_SESSION	page 3-65
RM_ACTIVATE_SESSION	page A-16
RM_SESSION_ACTIVATED	page A-22

```
Call SESSION_ACTIVATION_POLARITY(RM_ACTIVATE_SESSION.LU_NAME,
 RM_ACTIVATE_SESSION.MODE_NAME) (page 3-71)
 to determine the polarity of the next activated session (if any).
Select based on the activation polarity:
   When NONE (session limit exceeded)
      Create an RM_SESSION_ACTIVATED record.
      Set RM_SESSION_ACTIVATED.RETURN_CODE to LU_MODE_SESSION_LIMIT_EXCEEDED.
      Send the RM_SESSION_ACTIVATED record to PS (Chapter 5.4).
      Destroy RM_ACTIVATE_SESSION record.
   When FIRST_SPEAKER
      Call SEND_ACTIVATE_SESSION(RM_ACTIVATE_SESSION.LU_NAME,
       RM_ACTIVATE_SESSION.MODE_NAME, FIRST_SPEAKER) (page 3-65).
      Save the RM_ACTIVATE_SESSION record as a pending operator activation request.
   When BIDDER
      Call SEND_ACTIVATE_SESSION(RM_ACTIVATE_SESSION.LU_NAME,
       RM_ACTIVATE_SESSION.MODE_NAME, BIDDER) (page 3-65).
      Save the RM_ACTIVATE_SESSION record as a pending operator activation request.
```

RM_DEACTIVATE_SESSION_PROC

FUNCTION:	This procedure performs the processing of the RM_DEACTIVATE_SESSION record. An RM_DEACTIVATE_SESSION record is sent to RM by PS.COPR (Chapter 5.4) when the control operator issues a DEACTIVATE_SESSION command. The command directs RM to deactivate the session identified by HS_ID.
INPUT:	RM_DEACTIVATE_SESSION
OUTPUT:	DEACTIVATE_SESSION to SM, BIS_RQ to HS, session possibly removed from the free-session pool

Referenced procedures, FSMs, and data structures:

SEND_DEACTIVATE_SESSION	page 3-68
SEND_BIS_RQ	page 3-67
FSM_BIS_BIDDER	page 3-87
FSM_BIS_FSP	page 3-88
RM_DEACTIVATE_SESSION	page A-17
SCB	page A-8

```
Find the SCB that has an HS_ID that matches the RM_DEACTIVATE_SESSION.SESSION_ID.
If the SCB is found then
   Select based on RM_DEACTIVATE_SESSION.TYPE:
      When CLEANUP
         Call SEND_DEACTIVATE_SESSION(ACTIVE, RM_DEACTIVATE_SESSION.SESSION_ID,
          CLEANUP, X'00000000') (page 3-68).
         Destroy the RM_DEACTIVATE_SESSSION record.
      When NORMAL
         If the session is in use then
            If state of #FSM_BIS (page 3-87) ≠ BIS_SENT then (BIS not already sent)
               Queue the deactivation request.
            Else
               Destroy the RM_DEACTIVATE_SESSSION record.
         Else (session not in use)
            Queue the deactivation request.
            Call SEND_BIS_RQ(HS_ID) (page 3-67).
            Remove the session from the free-session pool.
Else
   Destroy the RM_DEACTIVATE_SESSION record.
```

RTR_RQ_PROC

FUNCTION:	This procedure handles the receipt of RTR requests from a first-speaker half-session. The session is returned to the free-session pool, and if there is a waiting request, the request is processed and a +RSP(RTR) is sent to the resources manager of the first-speaker half-session. If not, a -RSP(RTR, 0819) is sent to the resources manager to indicate that the resources manager of the bidder half-session has nothing to send.
INPUT:	RTR_RQ from HS
OUTPUT:	Positive RTR_RSP, or negative RTR_RSP(SENSE_CODE = X'08190000') to HS

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
GET_SESSION_PROC	page 3-52
SEND_DEACTIVATE_SESSION	page 3-68
SHOULD_SEND_BIS	page 3-76
SEND_BIS	page 3-66
RTR_RQ	page A-12
GET_SESSION	page A-16
RTR_RSP	page A-13
SCB	page A-8

```
Get addressability to the SCB representing the session on which the RTR_RQ was received.
If SCB.RTR_OWED is TRUE (the partner LU owes an RTR) then
   If there are any GET_SESSION requests waiting for sessions with
    the partner LU and mode then
      Return the session to the free-session pool.
      Create an RTR_RSP record with RTI set to POS and SENSE_CODE set to X'00000000'.
      Send the RTR_RSP record to HS (Chapter 6.0).
      Remove a GET_SESSION record from the waiting request queue.
      Call GET_SESSION_PROC(GET_SESSION) (page 3-52) to process the request.
   Else (no waiting requests)
      Create an RTR_RSP record with RTI set to NEG and SENSE_CODE set to X'08190000'.
      Send the RTR_RSP record to HS (Chapter 6.0).
      Call SHOULD_SEND_BIS(RTR_RQ.HS_ID) (page 3-76) to determine whether
       BIS should be sent on this session.
      If BIS should be sent then
         Call SEND_BIS(RTR_RQ.HS_ID) (page 3-66).
      Else
         Return the session to the free-session pool.
   Set SCB.RTR_OWED to FALSE.
Else (RTR not expected)
  Call SEND_DEACTIVATE_SESSION(ACTIVE, RTR_RQ.HS_ID, ABNORMAL, X'20030000') (page 3-68).
```

RTR_RSP_PROC

FUNCTION:	This procedure handles the receipt of RTR responses from a bidder half-session. If the input is a positive RTR_RSP, no processing is performed. If the input is a negative RTR_RSP(SENSE_CODE = X'0819'), the session is returned to the free-session pool, and the session is used to service a waiting request (if any).
INPUT:	Positive or negative RTR_RSP from HS
OUTPUT:	The session may be returned to the free-session pool.
NOTE:	If #FSM_BIS is not in the RESET state when RTR_RSP is received, this procedure does not return the session to the free-session pool, since the session is in the process of being shut down. This can occur, for example, when the first speaker has sent a BIS and the bidder responds negatively to a previous RTR before sending its own BIS.

Referenced procedures, FSMs, and data structures:

DEQUEUE_WAITING_REQUEST	page 3-48
SHOULD_SEND_BIS	page 3-76
SEND_BIS	page 3-66
FSM_BIS_BIDDER	page 3-87
FSM_BIS_FSP	page 3-88
RTR_RSP	page A-13

```
If RTR_RSP.RTI is NEG and the state of #FSM_BIS is RESET (page 3-87) then (see Note)
   Call SHOULD_SEND_BIS(RTR_RSP.HS_ID) (page 3-76)
    to determine whether BIS should be sent on this session.
   If BIS should be sent then
      Call SEND_BIS(RTR_RSP.HS_ID) (page 3-66).
   Else
      Return the session to the free-session pool.
      Call DEQUEUE_WAITING_REQUEST(RTR_RSP.HS_ID) (page 3-48) to process any waiting requests.
Destroy the RTR_RSP record.
```

SECURITY_PROC

FUNCTION:	This procedure checks the FMH-12, checks that the session is in the proper state to receive an FMH-12, and verifies the enciphered data found in the FMH-12.
INPUT:	An MU that contains the FMH-12 (see SNA Formats)
OUTPUT:	UNBIND processing if in error, state change of FSM_SCB_STATUS if OK, the buffer used by the MU freed

Referenced procedures, FSMs, and data structures:

SEND_DEACTIVATE_SESSION	page 3-68
FSM_SCB_STATUS_BIDDER	page 3-85
FSM_SCB_STATUS_FSP	page 3-86
MU	page A-29
SCB	page A-8
LUCB	page A-1

```
Find the SCB identified by MU.HS_TO_RM.HS_ID.
Remove the random data sent in the RSP(BIND) (found in SCB.RANDOM_DATA)
 from the LUCB.PENDING_RANDOM_DATA_LIST.
If the state of #FSM_SCB_STATUS ≠ PENDING_FMH12 (page 3-84) or
 the FMH_12 length ≠ 10 or the enciphered random data received in the FMH-12 ≠
 this LU's enciphered version of the same random data then
   Call SEND_DEACTIVATE_SESSION(ACTIVE, SCB.HS_ID, ABNORMAL, X'080F6051') (page 3-68).
   Optionally log the error in the system log.
Else
   Call #FSM_SCB_STATUS(R, FMH_12, UNDEFINED) (page 3-84)
    (initialized by CREATE_SCB).
Call buffer manager (FREE_BUFFER, buffer address) to release the buffer containing MU
 (Appendix B).
```

SEND_ACTIVATE_SESSION

FUNCTION:	This procedure sends an ACTIVATE_SESSION record to the session manager to request activation of a half-session. The appropriate pending session counts are incremented.
INPUT:	LU_NAME, the name of the partner LU; MODE_NAME, the name of the mode; the session polarity (FIRST_SPEAKER or BIDDER)
OUTPUT:	ACTIVATE_SESSION to SM, the pending session counts incremented

Referenced procedures, FSMs, and data structures:

SM	page 4-48
ACTIVATE_SESSION	page A-20
PENDING_ACTIVATION, see ACTIVATE_SESSION	page A-20
MODE	page A-3
LU_NAME	page 3-91
MODE_NAME	page 3-91

```
Find the MODE control block associated with LU_NAME and MODE_NAME.
Create an ACTIVATE_SESSION record and set the subfields as follows:
 CORRELATOR to a unique value, LU_NAME and MODE_NAME to the LU_NAME
 and MODE_NAME inputs, and SESSION_TYPE to the session polarity input.
Create a PENDING_ACTIVATION record initializing its subfields to the
 same values as in the ACTIVATE_SESSION fields.
Queue the PENDING_ACTIVATION.
Increment the MODE.PENDING_SESSION_COUNT by 1.
Increment the MODE.PENDING_CONWINNERS_COUNT or MODE.PENDING_CONLOSERS_COUNT by 1,
 as appropriate to the session polarity.
Send the ACTIVATE_SESSION record to SM (Chapter 4).
```

SEND_ATTACH_TO_PS

FUNCTION:	This procedure fills in the RM_TO_PS header information in the MU and sends it to the appropriate instance of PS.CONV.
INPUT:	The MU containing the FMH-5 (Attach) and the information to be inserted in the MU: TCB ID, RCB ID, and error sense data
OUTPUT:	The updated MU sent to PS

Referenced procedures, FSMs, and data structures:

PS	page 5.0-8
SCB	page A-8
MU	page A-29
RCB_ID	page 3-91
TCB_ID	page 3-91

Find the SCB identified by MU.HS_TO_RM.HS_ID.
Set the MU.RM_TO_PS fields as follows: RCB_ID to input RCB_ID, TCB_ID to input TCB_ID SEND_RU_SIZE to SCB.SEND_RU_SIZE, LIMITED_BUFFER_POOL_ID to SCB.LIMITED_BUFFER_POOL_ID, PERMANENT_BUFFER_POOL_ID to SCB.PERMANENT_BUFFER_POOL_ID, and SENSE_CODE to the input sense data.
Send the MU to PS (Chapter 5.0).
If the send fails then
  Call buffer manager (FREE_BUFFER, buffer address) to release the buffer containing MU (Appendix B).
  Optionally log the error in the system log.

SEND_BIS

FUNCTION:	This procedure causes either BIS_RQ or BIS_REPLY to be sent on the session identified by HS_ID. The choice of BIS_RQ or BIS_REPLY is dependent on the state of #FSM_BIS.
INPUT:	HS_ID, the ID of the session
OUTPUT:	BIS_RQ or BIS_REPLY to HS

Referenced procedures, FSMs, and data structures:

SEND_BIS_RQ	page 3-67
SEND_BIS_REPLY	page 3-67
FSM_BIS_BIDDER	page 3-87
FSM_BIS_FSP	page 3-88
HS_ID	page 3-91

Select based on the state of #FSM_BIS (page 3-87):
  When RESET
    Call SEND_BIS_RQ(HS_ID) (page 3-67).
  When BIS_RCVD
    Call SEND_BIS_REPLY(HS_ID) (page 3-67).
  Otherwise
    Do nothing.

SEND_BIS_REPLY

FUNCTION:	This procedure creates a BIS_REPLY and sends it to HS.
INPUT:	HS_ID, the ID of the half-session over which the BIS_REPLY will flow
OUTPUT:	BIS_REPLY sent to HS, MODE termination count incremented

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
FSM_BIS_BIDDER	page 3-87
FSM_BIS_FSP	page 3-88
BIS_REPLY	page A-12
MODE	page A-3
HS_ID	page 3-91

Call #FSM_BIS(S, BIS_REPLY, HS_ID) (page 3-87) for the session
 identified by HS_ID.
Create a BIS_REPLY record and send it to HS (Chapter 6.0).
Get addressability to the MODE control block associated with the LU and mode
 name of the session identified by HS_ID.
Increment MODE.PENDING_TERMINATION_CONWINNERS or MODE.PENDING_TERMINATION_CONLOSERS by 1,
 as appropriate to the session polarity.

SEND_BIS_RQ

FUNCTION:	This procedure creates a BIS_RQ and sends it to HS. After the BIS_RQ is sent to the half-session, the appropriate pending termination count is incremented.
INPUT:	HS_ID, the ID of the half-session over which the BIS_RQ will flow
OUTPUT:	BIS_RQ to HS, pending termination counts adjusted, queued RM_DEACTIVATE_SESSION records possibly destroyed
NOTE:	The TERMINATION_COUNT is not decremented if the BIS_RQ was sent as a result of a control operator RM_DEACTIVATE_SESSION request.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
FSM_BIS_BIDDER	page 3-87
FSM_BIS_FSP	page 3-88
BIS_RQ	page A-12
MODE	page A-3
RM_DEACTIVATE_SESSION	page A-17
HS_ID	page 3-91

Create a BIS_RQ record and send it to HS (Chapter 6.0).
Call #FSM_BIS(S, BIS_RQ, HS_ID) (page 3-87) for the session identified by HS_ID.
Get addressability to the MODE control block associated with the LU and mode
 name of the session identified by HS_ID.
Increment MODE.PENDING_TERMINATION_CONWINNERS or MODE.PENDING_TERMINATION_CONLOSERS by 1,
 as appropriate to the session polarity.
If there is a queued (pending) CNOS operator session deactivation request for the session
 identified by HS_ID then
   Discard all queued RM_DEACTIVATE_SESSION requests for the session identified by HS_ID.
Else (see Note)
   Decrement MODE.TERMINATION_COUNT by 1.

SEND_DEACTIVATE_SESSION

FUNCTION:	This procedure sends a DEACTIVATE_SESSION record to SM. If the STATUS is PENDING, the appropriate pending-session counts are decremented. If STATUS is ACTIVE, a SESSION_DEACTIVATED record is created and SESSION_DEACTIVATED_PROC is called to continue processing the session deactivation. SM does not send SESSION_DEACTIVATED in reply to DEACTIVATE_SESSION. Thus, the SESSIONS_DEACTIVATED is created in this procedure and SESSION_DEACTIVATED_PROC is called to perform common processing.
INPUT:	STATUS (ACTIVE or PENDING), CORRELATOR (HS_ID if STATUS = ACTIVE, else correlator used on ACTIVATE_SESSION request), TYPE (NORMAL, CLEANUP, ABNORMAL), and SENSE_CODE (X'00000000' if TYPE = NORMAL)
OUTPUT:	DEACTIVATE_SESSION to SM, MODE pending counts adjusted, waiting activation requests destroyed, waiting GET_SESSION requests rejected and destroyed, SESSION_DEACTIVATED records created

Referenced procedures, FSMs, and data structures:

SESSION_DEACTIVATED_PROC	page 3-72
PS	page 5.0-8
SM	page 4-48
SENSE_CODE	page 3-92
MODE	page A-3
GET_SESSION	page A-16
PENDING_ACTIVATION, see ACTIVATE_SESSION	page A-20
DEACTIVATE_SESSION	page A-21
SCB	page A-8
SESSION_DEACTIVATED	page A-14
SESSION_ALLOCATED	page A-22


```
Select based on the value of session status:
   When PENDING
      If there is a PENDING_ACTIVATION record with a matching CORRELATOR then
       (the pending activation is known to RM)
         Create a DEACTIVATE_SESSION record with DEACTIVATE_SESSION.STATUS set to PENDING,
          DEACTIVATE_SESSION.CORRELATOR set to CORRELATOR,
          DEACTIVATE_SESSION.TYPE set to TYPE, and
          DEACTIVATE_SESSION.SENSE_CODE set to SENSE_CODE.
         Send the DEACTIVATE_SESSION record to SM (Chapter 4).
         Get addressability to the MODE control block associated with the LU
          and mode name of the pending active session.
         Decrement MODE.PENDING_CONWINNERS_COUNT or MODE.PENDING_CONLOSERS_COUNT by 1,
          as appropriate to the session polarity.
         Decrement MODE.PENDING_SESSION_COUNT by 1.
         Destroy the PENDING_ACTIVATION record.
         If MODE.ACTIVE_SESSION_COUNT + MODE.PENDING_SESSION_COUNT = 0 then
            Do for each GET_SESSION request waiting for a session to this LU name for this
             mode name:
               Create a SESSION_ALLOCATED record with RETURN_CODE set to
                UNSUCCESSFUL_NO_RETRY and send it to the PS (Chapter 5.1)
                that initiated the session request.
               Destroy the waiting GET_SESSION record.
   When ACTIVE
      If there exists an SCB where SCB.HS_ID = CORRELATOR then (session is known to RM)
         Create a DEACTIVATE_SESSION record with DEACTIVATE_SESSION.STATUS set to ACTIVE,
          DEACTIVATE_SESSION.HS_ID set to CORRELATOR,
          DEACTIVATE_SESSION.TYPE set to TYPE, and
          DEACTIVATE_SESSION.SENSE_CODE set to SENSE_CODE.
         Send the DEACTIVATE_SESSION to SM (Chapter 4).
         Create a SESSION_DEACTIVATED record with HS_ID set to CORRELATOR.
         If TYPE is NORMAL then
            Set SESSION_DEACTIVATED.REASON to NORMAL.
         Else
            Set SESSION_DEACTIVATED.REASON to ABNORMAL_NO_RETRY.
            Set SESSION_DEACTIVATED.SENSE_CODE to SENSE_CODE.
         Call SESSION_DEACTIVATED_PROC(SESSION_DEACTIVATED) (page 3-72).
```

SEND_RTR_PROC

FUNCTION:	This procedure handles the processing that occurs when a SEND_RTR is received by RM.
INPUT:	SEND_RTR
OUTPUT:	RTR request sent on the session identified by the SEND_RTR record and the free session is removed from the free-session pool; or (if SEND_RTR is in error) a log entry is made; SEND_RTR record destroyed
NOTE:	The session is not free if it is in use. After the use of the session, a FREE_SESSION record will be received by RM and the logic to send RTR will again be exercised (see FREE_SESSION_PROC on page 3-50). When the session is in use, the SEND_RTR is ignored and discarded.

Referenced procedures, FSMs, and data structures:

SCB	page A-8
RTR_RQ	page A-12
SEND_RTR	page A-20


```
Find the SCB identified by SEND_RTR.HS_ID.
If the SCB exists and the session it represents is free and first speaker then
   Create and send an RTR_RQ to the associated half-session process.
   Set SCB.RTR_OWED to FALSE.
   Remove the session from the free-session pool.
Destroy the SEND_RTR record.
```

SESSION_ACTIVATED_ALLOCATION

FUNCTION:	This procedure handles the allocation processing for a newly activated first-speaker or bidder half-session. This procedure causes the SCB associated with the half-session and the RCB of a conversation for which a session was requested to point to each other. It then creates a SESSION_ALLOCATED record, which it sends to PS to inform it that the session has been allocated.
INPUT:	GET_SESSION and HS_ID, the ID of the new half-session
OUTPUT:	SESSION_ALLOCATED to PS and destruction of the GET_SESSION

Referenced procedures, FSMs, and data structures:

SET_RCB_AND_SCB_FIELDS	page 3-75
CONNECT_RCB_AND_SCB	page 3-42
PS	page 5.0-8
FSM_RCB_STATUS_FSP	page 3-90
FSM_RCB_STATUS_BIDDER	page 3-89
GET_SESSION	page A-16
HS_ID	page 3-91
SESSION_ALLOCATED	page A-22
SCB	page A-8

```
If the session identified by HS_ID is a bidder session then
    For the conversation identified by GET_SESSION.RCB_ID,
     Call #FSM_RCB_STATUS(S, GET_SESSION, UNDEFINED) (page 3-89).
Call SET_RCB_AND_SCB_FIELDS(GET_SESSION.RCB_ID, HS_ID) (page 3-75).
Call CONNECT_RCB_AND_SCB(GET_SESSION.RCB_ID, HS_ID, NORMAL) (page 3-42).
Find the SCB identified by HS_ID
Create a SESSION_ALLOCATED record with RETURN_CODE set to OK,
 SEND_RU_SIZE set to SCB.SEND_RU_SIZE,
 LIMITED_BUFFER_POOL_ID set to SCB.LIMITED_BUFFER_POOL_ID,
 PERMANENT_BUFFER_POOL_ID set to SCB.PERMANENT_BUFFER_POOL_ID,
 IN_CONVERSATION set to 'YES',
 and send the record to PS (Chapter 5.1).
```

SESSION_ACTIVATED_PROC

FUNCTION:	This procedure performs the processing of a SESSION_ACTIVATED record from SM. SESSION_ACTIVATED is received from SM as a result of session activation initiated by the partner LU.
INPUT:	SESSION_ACTIVATED from SM
OUTPUT:	Active session counts incremented

Referenced procedures, FSMs, and data structures:

SUCCESSFUL_SESSION_ACTIVATION	page 3-80
SESSION_ACTIVATED	page A-14
MODE	page A-3

```
Get addressability to the MODE control block associated with the LU and
 mode name of the newly activated session.
Increment MODE.ACTIVE_CONWINNERS_COUNT or MODE.ACTIVE_CONLOSERS_COUNT by 1,
 as appropriate to the session polarity.
Increment MODE.ACTIVE_SESSION_COUNT by 1.
Call SUCCESSFUL_SESSION_ACTIVATION(SESSION_ACTIVATED.LU_NAME,
 SESSION_ACTIVATED.MODE_NAME, SESSION_ACTIVATED.SESSION_INFORMATION) (page 3-80).
Destroy the SESSION_ACTIVATED record.
```

SESSION_ACTIVATION_POLARITY

FUNCTION:	This procedure determines the polarity for a session activation request. If no session can be activated now (because MODE.SESSION_LIMIT would be exceeded), NONE is returned. If either a first-speaker or bidder session could be activated, FIRST_SPEAKER is returned. Thus, first-speaker sessions are activated in preference to bidder sessions.
INPUT:	LU_NAME, the name of the LU to which a session is to be activated; and MODE_NAME, the name of the mode
OUTPUT:	NONE, if no session can be activated; FIRST_SPEAKER, if a first-speaker session can be activated; BIDDER, otherwise

Referenced procedures, FSMs, and data structures:

PARTNER_LU	page A-2
LU_NAME	page 3-91
MODE_NAME	page 3-91
MODE	page A-3

```
Get addressability to the PARTNER_LU control block associated with LU_NAME.
Get addressability to the MODE control block associated with LU_NAME and MODE_NAME.
If the number of pending and active sessions for the MODE
 is ≥ MODE.SESSION_LIMIT then
   Return with an indication that no additional sessions can be activated.
If the total number of pending and active sessions for the MODE
 is > 0 and PARTNER_LU.PARALLEL_SESSION is SUPPORTED then
   Return with an indication that no additional sessions can be activated.
If MODE.SESSION_LIMIT - MODE.MIN_CONLOSERS_LIMIT >
 MODE.ACTIVE_CONWINNERS_COUNT + MODE.PENDING_CONWINNERS_COUNT then
   Return with an indication that a first-speaker session can be activated.
Else
   Return with an indication that a bidder session can be activated.
```

SESSION_DEACTIVATED_PROC

FUNCTION: This procedure handles the processing that occurs when a session is deactivated.

When SESSION_DEACTIVATED.REASON = NORMAL and the session was not being used by a conversation, no processing (except destruction of the SCB) takes place, since the decision to close down a session was mutually reached by the resources managers of the half-sessions via BIS protocols, and all necessary processing has already been performed.

When SESSION_DEACTIVATED.REASON = NORMAL, ABNORMAL_RETRY, or ABNORMAL_NO_RETRY and the session was being used by a conversation, this procedure sends a CONVERSATION_FAILURE record to PS. If the session was not in use, the session is removed from the free-session pool. Regardless of whether the session was in use, this procedure deletes the SCB entry for that half-session.

INPUT: SESSION_DEACTIVATED

OUTPUT: CONVERSATION_FAILURE to PS, destruction of the SCB; a queued Attach may be purged; the active contention-winner, contention-loser, and session counts are adjusted; pending termination counts are adjusted; sessions are activated; if they cannot be satisfied, waiting requests are rejected; the session deactivated TP may be started; the SESSION_DEACTIVATED record is destroyed; RM_ACTIVATE_SESSION records are rejected using RM_SESSION_ACTIVATED records.

NOTES:

1. When PS receives a CONVERSATION_FAILURE, it generates a DEALLOCATE_RCB and sends it to RM, which performs the usual RCB deallocation processing.

2. An UNBIND type Normal not preceded by BIS protocols can occur when an SSCP has issued a CTERM type Forced to an LU. Under any other circumstance, an UNBIND type Normal not preceded by BIS protocols is a protocol violation.

3. It is possible for two RCBs to be associated with the same SCB when SON occurs. This happens when RM has issued a Bid for the use of a bidder half-session and, prior to receiving the response to the Bid, subsequently receives an Attach from the first-speaker side of the session. When RM receives the session-outage notification, it notifies the PS that was created as a result of the incoming Attach that a conversation failure has occurred. The PS associated with the RCB that is pending a response to the Bid, however, never learns of the session outage. RM treats the SON as a -BID_RSP and attempts to satisfy the session request with another session.

Referenced procedures, FSMs, and data structures:

PS	page 5.0-8
GET_SESSION_PROC	page 3-52
ACTIVATE_NEEDED_SESSIONS	page 3-24
FSM_SCB_STATUS_BIDDER	page 3-85
FSM_SCB_STATUS_FSP	page 3-86
FSM_RCB_STATUS_FSP	page 3-90
FSM_RCB_STATUS_BIDDER	page 3-89
SESSION_ALLOCATED	page A-22
MU	page A-29
SESSION_DEACTIVATED	page A-14
CONVERSATION_FAILURE	page A-21
GET_SESSION	page A-16
RM_ACTIVATE_SESSION	page A-16
RM_SESSION_ACTIVATED	page A-22
SCB	page A-8
RCB	page A-6
MODE	page A-3
PARTNER_LU	page A-2

```
If an SCB associated with the half-session identified by
 SESSION_DEACTIVATED.HS_ID exists then
   Get addressability to the MODE control block associated with the LU name and
    mode name of the deactivated session.
   If the state of #FSM_SCB_STATUS (page 3-84) is IN_USE then
      If the RCB identified by the SCB.RCB_ID exists then
         If the RCB.TCB_ID is not null (in conversation) then
            Create a CONVERSATION_FAILURE record with RCB_ID set to SCB.RCB_ID.
            Select based on SESSION_DEACTIVATED.REASON:
               When NORMAL
                  Set CONVERSATION_FAILURE.REASON to SON.     (Note 2)
               When ABNORMAL_RETRY
                  Set CONVERSATION_FAILURE.REASON to SON.
               When ABNORMAL_NO_RETRY
                  Set CONVERSATION_FAILURE.REASON to PROTOCOL_VIOLATION.
            Send the CONVERSATION_FAILURE record to the PS process that was
             using the deactivated session.
         Else  (MU containing an Attach from the ended HS is queued awaiting a TP)
            Find the MU queued for the transaction program identified by RCB.TP_NAME
             and where MU.RM_TO_PS.RCB_ID = RCB.RCB_ID.
               Destroy the RCB.
               Remove the MU from the queue.
               Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
                containing MU (Appendix B).
   Else (session not in use by a conversation)
      Remove the session from the free-session pool.
   If there is an RCB where RCB.HS_ID = SESSION_DEACTIVATED.HS_ID and
    the state of #FSM_RCB_STATUS = PENDING_SCB (page 3-89) then
    (A bid for the deactivated session is in progress; see Note 3).
      Set RCB.HS_ID to a null value.
      Call #FSM_RCB_STATUS(R, NEG_BID_RSP, UNDEFINED) (page 3-89).
      Create a GET_SESSION record from information saved in the RCB. (see BIDDER_PROC)
      Call GET_SESSION_PROC(GET_SESSION) (page 3-52)
       to retry the bid on another session.
   Decrement MODE.ACTIVE_CONWINNERS_COUNT or MODE.ACTIVE_CONLOSERS_COUNT by 1,
    as appropriate to the session polarity.
   Decrement MODE.ACTIVE_SESSION_COUNT by 1.
   If there is a pending deactivation for the failed session then
      Decrement MODE.PENDING_TERMINATION_CONWINNERS or MODE.PENDING_TERMINATION_CONLOSERS
       by 1, as appropriate to the session polarity.
   If SESSION_DEACTIVATED.REASON ≠ ABNORMAL_NO_RETRY then
      Call ACTIVATE_NEEDED_SESSIONS(SCB.LU_NAME, SCB.MODE_NAME) (page 3-24).
   If MODE.ACTIVE_SESSION_COUNT + MODE.PENDING_SESSION_COUNT = 0 then
      If the PARTNER_LU for the failed session does not support parallel sessions
       the SESSION_DEACTIVATED.REASON is not ABNORMAL_NO_RETRY then
         If there is a waiting request on another mode
            CALL ACTIVATE_NEEDED_SESSIONS(PARTNER_LU.LOCAL_LU_NAME,MODE.NAME)
      Do for each GET_SESSION request waiting for a session to (LU_NAME, MODE_NAME):
         Create a SESSION_ALLOCATED record with RETURN_CODE set to  UNSUCCESSFUL_NO_RETRY
          and send it to the PS (Chapter 5.1) that initiated the session request.
         Destroy the waiting GET_SESSION request.
      Do for each pending RM_ACTIVATE_SESSION request for a session to
       (LU_NAME, MODE_NAME):
         Create RM_SESSION_ACTIVATED with RETURN_CODE set to ACTIVATION_FAILURE_NO_RETRY
          and send it to the PS (Chapter 5.1) that initiated the activation request.
         Destroy the RM_ACTIVATE_SESSION request.
   Destroy the SCB.
Destroy the SESSION_DEACTIVATED record.
```

SESSION_DEACTIVATION_POLARITY

FUNCTION:	This procedure determines the polarity of a session to partner LU (LU_NAME, MODE_NAME) that this LU is responsible for deactivating.
INPUT:	LU_NAME, the name of the partner LU; and MODE_NAME, the name of the mode
OUTPUT:	One of the following indications to the caller: NONE, if this LU is not responsible for any deactivations; BIDDER, if this LU is responsible to deactivate a bidder session only; FIRST_SPEAKER, if this LU is responsible to deactivate a first speaker session only; EITHER, if this LU is responsible to deactivate either a first speaker or bidder session. The TERMINATION_COUNT is reset to 0 if it was positive and this LU is not responsible for any deactivations.

Referenced procedures, FSMs, and data structures:

LU_NAME	page 3-91
MODE_NAME	page 3-91
MODE	page A-3

```
Get addressability to the MODE control block associated with LU_NAME and MODE_NAME.
If MODE.TERMINATION_COUNT is 0 then
   Return with an indication that no sessions need to be deactivated.
Let CONWINNER_COUNT be MODE.ACTIVE_CONWINNERS_COUNT + MODE.PENDING_CONWINNERS_COUNT -
 MODE.PENDING_TERMINATION_CONWINNERS.
Let CONLOSER_COUNT be MODE.ACTIVE_CONLOSERS_COUNT + MODE.PENDING_CONLOSERS_COUNT -
 MODE.PENDING_TERMINATION_CONLOSERS.
Select based on the following conditions:
   When CONWINNER_COUNT <= MODE.MIN_CONWINNERS_LIMIT, and
    CONLOSER_COUNT <= MODE.MIN_CONLOSERS_LIMIT
      Set MODE.TERMINATION_COUNT to 0.
      Return with an indication (NONE) that no sessions need to be deactivated.
   When CONWINNER_COUNT <= MODE.MIN_CONWINNERS_LIMIT, and
    CONLOSER_COUNT > MODE.MIN_CONLOSERS_LIMIT
      Return with an indication (BIDDER) that a bidder session needs to be deactivated.
   When CONWINNER_COUNT > MODE.MIN_CONWINNERS_LIMIT, and
    CONLOSER_COUNT <= MODE.MIN_CONLOSERS_LIMIT
      Return with an indication (FIRST_SPEAKER) that a first-speaker session needs
       to be deactivated.
   When CONWINNER_COUNT > MODE.MIN_CONWINNERS_LIMIT, and
    CONLOSER_COUNT > MODE.MIN_CONLOSERS_LIMIT
      Return with an indication (EITHER) that a session of either polarity needs
       to be deactivated.
```

SET_RCB_AND_SCB_FIELDS

FUNCTION:	This procedure initializes fields in the RCB and SCB entries having the passed RCB and HS IDs. The RCB is set to point to the associated SCB (by placing the HS_ID in the RCB), and the SCB to point to the RCB (by placing the RCB_ID in the SCB). The FSMs that maintain the status of the RCB and SCB are set to the IN_USE state.
INPUT:	RCB_ID and HS_ID, the IDs of the RCB and SCB, respectively, for which fields are to be set
OUTPUT:	Fields in the RCB and SCB initialized.
NOTE:	When this procedure is called from BID_RSP_PROC, RCB.HS_ID has already been initialized. (It was initialized when the BID record for the session was generated.) Rather than test for this condition, the field is reset to the same value.

Referenced procedures, FSMs, and data structures:

FSM_SCB_STATUS_BIDDER	page 3-85
FSM_SCB_STATUS_FSP	page 3-86
FSM_RCB_STATUS_FSP	page 3-90
FSM_RCB_STATUS_BIDDER	page 3-89
RCB_ID	page 3-91
HS_ID	page 3-91
SCB	page A-8
RCB	page A-6

Find the SCB associated with the half-session identified by HS_ID.
Set SCB.RCB_ID to RCB_ID.
Find the RCB associated with the conversation identified by RCB_ID.
Set RCB.HS_ID to HS_ID (see Note).
If the session identified by HS_ID is a first-speaker session then
   Call #FSM_SCB_STATUS(S, GET_SESSION, UNDEFINED) (page 3-84).
   Call #FSM_RCB_STATUS(S, GET_SESSION, UNDEFINED) (page 3-89).
Else (bidder session)
   Call #FSM_SCB_STATUS(R, POS_BID_RSP, UNDEFINED) (page 3-84).
   Call #FSM_RCB_STATUS(R, POS_BID_RSP, UNDEFINED) (page 3-89).

SHOULD_SEND_BIS

FUNCTION:	This procedure determines whether a BIS (either BIS_RQ or BIS_REPLY) should be sent on the session identified by HS_ID.
INPUT:	HS_ID, containing the ID of the session
OUTPUT:	TRUE, if BIS (BIS_RQ or BIS_REPLY) should be sent now; else FALSE
NOTE:	BIS is sent if there are no waiting requests for a session for this mode

Referenced procedures, FSMs, and data structures:


```
Find the PARTNER_LU and MODE control block associated with
 the half-session identified by HS_ID.
If there are no waiting requests for a session for this MODE and
 PARTNER_LU.PARALLEL_SESSIONS is NOT_SUPPORTED then
   If there is a waiting request for another mode with this partner LU then
      Return to the calling routine with the value TRUE (BIS should be sent).
SELECT based on the state of #FSM_BIS (page 3-87):
   When RESET
      Call SESSION_DEACTIVATION_POLARITY(LU_NAME, MODE_NAME) (page 3-74)
       to determine the type of session (if any) to deactivate.
      If the deactivation polarity is EITHER, or
       the deactivation polarity matches the session polarity then
         If MODE.DRAIN_SELF is NO or there are no waiting requests for
          sessions to this LU and mode name (See Note) then
            Return to the calling routine with the value TRUE (BIS should be sent).
         If there is a pending RM_DEACTIVATE_SESSION request for this session then
            Return to the calling routine with the value TRUE (BIS should be sent).
         Return to the calling routine with the value FALSE (BIS should not be sent).
   When BIS_RCVD
      If MODE.DRAIN_SELF = NO or there are no waiting requests for sessions
       for this LU name and mode name (See Note) then
           Return to the calling routine with the value TRUE (BIS should be sent).
      Else
         Return to the calling routine with the value FALSE (BIS should not be sent).
   When BIS_SENT (BIS already sent)
      Return to the calling routine with the value FALSE (BIS should not be sent).
```

START_TP_PROC

FUNCTION:	This procedure performs the processing of a received START_TP record.
INPUT:	START_TP request record
OUTPUT:	A reply (RESPONSE_CODE) to the START_TP and if the request can be satisfied, the START_TP updated and sent to the new PS process, or queued awaiting a freed instance
NOTES:	1. A null network ID in the Fully Qualified LU Name field is invalid unless this LU's network ID is also null. Procedures that are able to send START_TP records to RM are considered privileged, protected processes with code content integrity. These procedures may supply the fully-qualified (network-qualified) LU name of the requester or an Already-Verified indication for security (i.e., a user ID indicated as already verified, eliminating the need for a password).
	2. This logic requires support of the limited-instance transaction program option and requires that all transaction programs are able to perform as a limited-instance TP (i.e., able to accept an Attach or a START_TP after sending TERMINATE_PS). If either of these assumptions is not true, the logic in the Else section is performed.

Referenced procedures, FSMs, and data structures:

PS	page 5.0-8
START_TP	page A-19
TRANSACTION_PROGRAM	page A-5
LUCB	page A-1
RESPONSE_CODE	page 3-92
START_TP_REPLY	page A-20
CREATE_TCB_AND_PS	page 3-45
START_TP_SECURITY_VALID	page 3-79
PURGE_QUEUED_REQUESTS	page 3-59


```
Set RESPONSE_CODE to OK.
Find the TRANSACTION_PROGRAM corresponding to the START_TP.TARGET_TP_NAME.
If the TRANSACTION_PROGRAM cannot be found then
   Set RESPONSE_CODE to TPN_NOT_RECOGNIZED.
Else
   If the TRANSACTION_PROGRAM.STATUS is DISABLED_TEMPORARY then
      Set RESPONSE_CODE to TRANS_PGM_NOT_AVAILABLE_RETRY.
   If the TRANSACTION_PROGRAM.STATUS is DISABLED_PERMANENT then
      Set RESPONSE_CODE to TRANS_PGM_NOT_AVAIL_NO_RETRY.
   If the TRANSACTION_PROGRAM.VERIFY_PIP is YES and RESPONSE_CODE is OK then
      If the number of PIP subfields in the START_TP is not equal to
       TRANSACTION_PROGRAM.NUMBER_OF_PIP_SUBFIELDS then
         If the TRANSACTION_PROGRAM.NUMBER_OF_PIP_SUBFIELDS is 0 then
            Set RESPONSE_CODE to PIP_NOT_ALLOWED.
         Else
            Set RESPONSE_CODE to PIP_NOT_SPECIFIED_CORRECTLY.
   If the RESPONSE_CODE is OK then
      Call START_TP_SECURITY_VALID(START_TP, TRANSACTION_PROGRAM) (page 3-79).
      If START_TP_SECURITY_VALID returns indicating that
       security requirements are not met then
         Set RESPONSE_CODE to SECURITY_NOT_VALID.
   If the RESPONSE_CODE is OK and a network-qualified LU name is present in START_TP
    and the network-qualified LU name does not have the proper format (see SNA Formats for
    the correct format) then
      Set RESPONSE_CODE to INVALID_FULLY_QUALIFIED_LU_NAME.
If the RESPONSE_CODE is OK then
   If a network-qualified LU name is not present in the START_TP (Note 1) then
      Set START_TP.FULLY_QUALIFIED_LU_NAME to LUCB.FULLY_QUALIFIED_LU_NAME.
```

```
        If TRANSACTION_PROGRAM.INSTANCE_COUNT is less than TRANSACTION_PROGRAM.INSTANCE_LIMIT then
           Call CREATE_TCB_AND_PS(START_TP, TRANSACTION_PROGRAM) (page 3-45).
           If START_TP.TCB_ID is a non-null value (PS is successfully created) then
              Send a copy of the START_TP record to PS.
              If START_TP.REPLY is YES then
                 Create a START_TP_REPLY record, initializing START_TP_REPLY.RESPONSE_CODE to
                  RESPONSE_CODE.
                 If RESPONSE_CODE is equal to OK then
                    Set the START_TP_REPLY.TCB_ID to START_TP.TCB_ID.
                 Send the START_TP_REPLY record to the initiating process.
              Destroy the START_TP record.
           Else
              If TRANSACTION_PROGRAM.INSTANCE_COUNT is greater than 0 (Note 2) then
                 Queue the START_TP to await the freeing of an active target TP-PS instance.
              Else
                 If START_TP.REPLY is YES then
                    Create a START_TP_REPLY record.
                    Set the START_TP_REPLY.RESPONSE_CODE to PS_CREATION_FAILURE.
                    Send the START_TP_REPLY record to the initiating process.
                 Destroy the START_TP record.
                 Purge any queued START_TP or Attach records for this transaction program .
                  by calling PURGE_QUEUED_REQUESTS(TRANSACTION_PROGRAM) (page 3-59).
        Else
           Queue the START_TP to await the freeing of an active target TP-PS instance.
     Else (RESPONSE_CODE is not OK)
        If START_TP.REPLY is YES then
           Create a START_TP_REPLY record, initializing START_TP_REPLY.RESPONSE_CODE to
            RESPONSE_CODE.
           Set the START_TP_REPLY.TCB_ID to a null value.
           Send the START_TP_REPLY record to the initiating process.
        Destroy the START_TP record.
```

START_TP_SECURITY_VALID

FUNCTION:	This procedure performs all security checks on an incoming START_TP.
INPUT:	The START_TP record containing security fields (e.g., user ID, password, and profile) and the pointer to the control block of the transaction program that the START_TP targets
OUTPUT:	An indication as to the validity of the security tokens on the START_TP: TRUE indicates they are valid; FALSE, invalid

Referenced procedures, FSMs, and data structures:

START_TP page A-19

```
If START_TP.SECURITY_SELECT = NONE then
   If the transaction program requires security parameters then
      Return FALSE (security check failed).
   Else
      Return TRUE (security check passed).
If the START_TP contains a profile but does not contain a user ID then
   Return FALSE.
If the START_TP contains a password but does not contain a user ID then
   Return FALSE.
If START_TP.SECURITY_SELECT is PGM then
   If the START_TP contains a user ID but does not contain a password then
      Return FALSE.
   If the START_TP does not contain a user ID or a password and
     the transaction program requires security parameters then
      Return FALSE.
   If the transaction program does not require security parameters and
     the START_TP does not contain a user ID or a password then
      Return TRUE.
   If the START_TP contains an unauthorized combination of user ID and profile or
     the START_TP contains an invalid combination of user ID and password then
      Return FALSE.
Else  (SECURITY_SELECT is ALREADY_VERIFIED)
   If the START_TP does not contain a user ID or does contain a password then
      Return FALSE (Already verified indication requires a user ID and precludes a password).
If there is limited access to the target transaction program  (The access authorization is
 based upon the definition of the transaction program's access requirements and the START_TP
 record's user ID and/or profile and/or LU name of the origin of the START_TP) then
   If the user ID and/or profile and/or LU name is permitted access to the requested
     transaction program then
      Return TRUE.
   Else
      Return FALSE.
Return TRUE.
```

SUCCESSFUL_SESSION_ACTIVATION

FUNCTION: This procedure handles the processing that occurs when a new session is successfully activated.

When a new session is successfully activated, RM informs the new half-session that RM is aware of its existence and ready to accept records from the new half-session. A new session comes up "in-conversation" with the primary side of the session in control of the conversation. This procedure checks to see whether the new half-session is primary or secondary. If the half-session is primary and a request is waiting, the support levels (i.e., sync level and conversation-level security) specified in the request are checked against the support levels of the session. If the support levels are compatible, and LU-LU verification (session-level security) is active, the FMH-12 to complete LU-LU verification is built and sent to the partner-LU resources manager; then the request is sent to SESSION_ACTIVATED_ALLOCATION (page 3-70) to be processed. If the support levels are not compatible, the request is rejected with an ALLOCATION_ERROR return code. If no requests are waiting, the session is returned to the free-session pool. If no request is waiting and LU-LU verification (session-level security) is active, the FMH-12 is built and sent to the partner-LU resources manager, and this FMH-12 relinquishes control of the session; otherwise, a YIELD_SESSION record is created and sent to HS to inform the secondary half-session that the primary is relinquishing control of the conversation. The YIELD_SESSION record is translated into a FREE_SESSION record by the secondary half-session and sent to its RM.

If the new half-session is a secondary half-session and LU-LU verification is active, the FSM that maintains the status of the SCB is set to indicate that the next record it expects to receive is an FMH-12 (Security). If the new half-session is a secondary half-session and LU-LU verification is not active, the FSM that maintains the status of the SCB is set to indicate that the next record it expects to receive is either an Attach or a FREE_SESSION. (It will receive an Attach if the primary half-session decides to use the session; it will receive a FREE_SESSION if the primary has no GET_SESSION requests waiting to be serviced).

INPUT: LU_NAME and MODE_NAME, the LU name and mode name of the newly activated session; and SESSION_INFORMATION (page A-32), which describes the attributes of the activated session

OUTPUT: GET_SESSION to SESSION_ACTIVATED_ALLOCATION (page 3-70), YIELD_SESSION to HS, SESSION_ALLOCATED to PS, waiting GET_SESSION records destroyed, an RM_SESSION_ACTIVATED record possibly created to respond to a CNOS RM_ACTIVATE_SESSION record that will be dequeued from the PENDING_CNOS_ACTIVATION_LIST and destroyed

NOTE: PS stores information in the RCB that tells HS what bit settings to use when HS sends data out over a link. Part of the information indicates whether the data being sent to HS is the beginning of a conversation or part of an existing conversation. Since a new session comes up in-conversation (a fact unknown by PS), RM changes the information in the RCB to indicate to HS that the next record it receives from PS will not be the start of a conversation.

Referenced procedures, FSMs, and data structures:

CREATE_SCB	page 3-44
SESSION_ACTIVATED_ALLOCATION	page 3-70
PS	page 5.0-8
HS	page 6.0-3
FSM_SCB_STATUS_BIDDER	page 3-85
FSM_SCB_STATUS_FSP	page 3-86
LU_NAME	page 3-91
MODE_NAME	page 3-91
SESSION_INFORMATION	page A-32
RM_HS_CONNECTED	page A-18
SCB	page A-8
RM_ACTIVATE_SESSION	page A-16
RM_SESSION_ACTIVATED	page A-22

GET_SESSION page A-16
YIELD_SESSION page A-19
SESSION_ALLOCATED page A-22
ENCIPHERED_RD2 page A-18

```
Call CREATE_SCB(LU_NAME, MODE_NAME, SESSION_INFORMATION) (page 3-44).
Create an RM_HS_CONNECTED record and send it to the HS
 identified by SESSION_INFORMATION.HS_ID.
If this is a primary half-session then
   Call #FSM_SCB_STATUS(R, SESSION_ACTIVATED, PRI) (page 3-84).
   Do until the activated session is used to service a waiting request, or
    the session is yielded:
      If a GET_SESSION request is waiting for a session to this partner LU and
       on this mode then
         If the session does not support the security level specified by the
          waiting request then
            Downgrade the specified security level to NONE.
         If the session does not support the sync level specified by the
          waiting request then
            Create a SESSION_ALLOCATED record with RETURN_CODE set to
             SYNC_LEVEL_NOT_SUPPORTED and send it to the PS (Chapter 5.1) associated
             with the waiting request.
            Destroy the waiting GET_SESSION request.
         Else (session support is OK)
            If LU-LU verification is active (random data is present in the SCB) then
               Create an ENCIPHERED_RD2 containing an FMH-12 (refer to SNA Formats)
                initialized with the enciphered version of the random data present in the SCB.
               Set ENCIPHERED_RD2.SEND_PARM.ALLOCATE to NO.
               Set ENCIPHERED_RD2.SEND_PARM.FMH to YES.
               Set ENCIPHERED_RD2.SEND_PARM.TYPE to FLUSH.
               Send the ENCIPHERED_RD2 to the HS (Chapter 6.0) representing
                the newly activated session.
            Call SESSION_ACTIVATED_ALLOCATION(GET_SESSION, SCB.HS_ID) (page 3-70).
            Destroy the waiting GET_SESSION request.
      Else (no waiting requests)
         Call #FSM_SCB_STATUS(S, YIELD_SESSION, UNDEFINED) (page 3-84).
         If LU-LU verification is active (random data is present in the SCB) then
            Create an ENCIPHERED_RD2 containing an FMH-12 initialized with the enciphered
             version of the random data present in the SCB.
            Set ENCIPHERED_RD2.SEND_PARM.ALLOCATE to NO.
            Set ENCIPHERED_RD2.SEND_PARM.FMH to YES.
            Set ENCIPHERED_RD2.SEND_PARM.TYPE to DEALLOCATE_FLUSH (yields the session).
            Send the ENCIPHERED_RD2 to the HS (Chapter 6.0) representing
             the newly activated session.
         Else
            Create a YIELD_SESSION record and send it to the HS (Chapter 6.0)
             representing the newly activated session.
Else (secondary half-session)
   If LU-LU verification is active (random data is present in the SCB) then
      Call #FSM_SCB_STATUS(R, SESSION_ACTIVATED, SECURE) (page 3-84).
   Else
      Call #FSM_SCB_STATUS(R, SESSION_ACTIVATED, SEC) (page 3-84).
If an RM_ACTIVATE_SESSION request is pending then
   Create an RM_SESSION_ACTIVATED record with RETURN_CODE set to OK and send
    it to the PS (Chapter 5.4) that originally issued
    the RM_ACTIVATE_SESSION record to RM.
   Destroy the pending RM_ACTIVATE_SESSION request.
```

TEST_FOR_FREE_FSP_SESSION

FUNCTION:	This procedure tests for a free first-speaker half-session. If one is found, a new RCB is created and the support levels (conversation-level security and sync level) provided by the session are checked to see if they are compatible with those requested in the ALLOCATE_RCB. If they are not compatible, the RETURN_CODE on the RCB_ALLOCATED record is set to indicate an unsuccessful allocation, or, in the case of a security incompatibility, the security level is downgraded to a compatible level. If the support levels are compatible, the half-session is allccated to the RCB, the ID of the RCB is placed in the passed RCB_ALLOCATED record. If a free first-speaker half-session is not found, the RETURN_CODE in the passed RCB_ALLOCATED record is changed to indicate an unsuccessful allocation.
INPUT:	ALLOCATE_RCB and RCB_ALLOCATED (the latter created by ALLOCATE_RCB_PROC)
OUTPUT:	RCB_ALLOCATED with the RCB_ID field set to the ID of the allocated RCB, or with the RETURN_CODE set to UNSUCCESSFUL

Referenced procedures, FSMs, and data structures:

CREATE_RCB	page 3-43
SET_RCB_AND_SCB_FIELDS	page 3-75
CONNECT_RCB_AND_SCB	page 3-42
ALLOCATE_RCB	page A-15
RCB_ALLOCATED	page A-21
RCB	page A-6
SCB	page A-8

```
If a free first-speaker session exists for ALLOCATE_RCB.LU_NAME and
 ALLOCATE_RCB.MODE_NAME then
   Call CREATE_RCB(ALLOCATE_RCB, RCB_ALLOCATED) (page 3-43).
   If the security level requested in the RCB.SECURITY_SELECT
    is not supported by the partner LU then
      Downgrade the requested level of security to NONE.
   If the sync level requested in ALLOCATE_RCB is not supported by the partner LU then
      Set RCB_ALLOCATED.RETURN_CODE to SYNC_LEVEL_NOT_SUPPORTED.
   Else
      Call SET_RCB_AND_SCB_FIELDS(RCB_ID, HS_ID) (page 3-75).
      Call CONNECT_RCB_AND_SCB(RCB_ID, HS_ID) (page 3-42).
      Set the following fields in the RCB_ALLOCATED:  RETURN_CODE to OK,
       SEND_RU_SIZE to SCB.SEND_RU_SIZE, LIMITED_BUFFER_POOL_ID to
       SCB.LIMITED_BUFFER_POOL_ID, and PERMANENT_BUFFER_POOL_ID to
       SCB.PERMANENT_BUFFER_POOL_ID.
      Remove the session from the free-session pool.
Else (no free first-speaker sessions).
   Set RCB_ALLOCATED.RETURN_CODE to UNSUCCESSFUL.
```

UNSUCCESSFUL_SESSION_ACTIVATION

FUNCTION:	This procedure handles the processing that occurs when a new session could not be activated by the session manager. This procedure checks to see if any session has been activated for this (LU_NAME, MODE_NAME) pair. If so, no action is taken by this procedure. The (one or more) previously allocated sessions will eventually be available for use by the transaction programs that requested a session. Similarly, if no sessions have been activated for this (LU_NAME, MODE_NAME) pair, but there are outstanding (pending) session activation requests that the session manager has not yet responded to, no action is taken. Some of the pending requests may succeed in activating sessions, and these sessions can eventually be used by other transaction programs. If, on the other hand, no session has been successfully activated for this LU_NAME and MODE_NAME and there are no other pending activation requests for this LU_NAME and MODE_NAME (i.e., all session activation requests have been responded to by the session manager), the procedure will send a SESSION_ALLOCATED record to all instances of presentation services that have requested sessions for this LU_NAME and MODE_NAME. The RETURN_CODE field of the SESSION_ALLOCATED record is set to UNSUCCESSFUL_RETRY or UNSUCCESSFUL_NO_RETRY depending on the ERROR_TYPE parameter.
INPUT:	LU_NAME and MODE_NAME of the LU to which session activation was unsuccessful; and ERROR_TYPE, indicating RETRY or NO_RETRY
OUTPUT:	SESSION_ALLOCATED to PS, WAITING_REQUEST records destroyed, RM_SESSION_ACTIVATED records created and sent to PS, RM_ACTIVATE_SESSION records destroyed

Referenced procedures, FSMs, and data structures:

PS	page 5.0-8
ACTIVATE_NEEDED_SESSIONS	page 3-24
LU_NAME	page 3-91
MODE_NAME	page 3-91
GET_SESSION	page A-16
MODE	page A-3
RM_ACTIVATE_SESSION	page A-16
RM_SESSION_ACTIVATED	page A-22
SESSION_ALLOCATED	page A-22


```
Get addressability to the MODE control block associated with LU_NAME and MODE_NAME.
If MODE.ACTIVE_SESSION_COUNT is 0 and MODE.PENDING_SESSION_COUNT is 0 then
   Do for each waiting request for a session
    to this partner (LU_NAME) using this mode (MODE_NAME):
      Create a SESSION_ALLOCATED record with RETURN_CODE set to
       UNSUCCESSFUL_RETRY or UNSUCCESSFUL_NO_RETRY according to
       ERROR_TYPE.
      Send the SESSION_ALLOCATED record to the PS (Chapter 5.1)
       that issued the original request.
      Destroy the waiting request.
   If this partner LU does not support parallel sessions and
    there are other waiting GET_SESSION requests for a session to this partner
    for a different mode then
      Try to activate a session for the other mode to satisfy a waiting request by
       calling ACTIVATE_NEEDED_SESSIONS(LU_NAME, MODE_NAME for the mode
       with waiting requests) (page 3-24).
Do while the number of pending RM_ACTIVATE_SESSION operator requests is greater than
 the MODE.PENDING_SESSION_COUNT:
   Create an RM_SESSION_ACTIVATED record with RETURN_CODE set to
    ACTIVATION_FAILURE_RETRY or ACTIVATION_FAILURE_NO_RETRY according to ERROR_TYPE.
   Send the RM_SESSION_ACTIVATED record to the PS (Chapter 5.1)
    that issued the original request.
   Destroy the pending RM_ACTIVATE_SESSION request.
```

#FSM_SCB_STATUS

#FSM_SCB_STATUS is a generic FSM that maintains the state of a half-session. There is one #FSM_SCB_STATUS for each session known to the resources manager. #FSM_SCB_STATUS is initialized to either FSM_SCB_STATUS_BIDDER or FSM_SCB_STATUS_FSP, depending on the session polarity, when the resources manager becomes aware of the existence of a new session. This initialization occurs in CREATE_SCB (page 3-44).

The states of FSM_SCB_STATUS_BIDDER and FSM_SCB_STATUS_FSP are:

- SESSION ACTIVATION--the initial state, following activation of the session
- FREE--the session is free for use by a conversation
- PENDING ATTACH--the session is in the in-bracket state and the local LU is waiting for an Attach FM header from the remote LU
- IN USE--the session is in use by a conversation
- PENDING FMH12--the session is waiting for the Security FM header from the remote LU before beginning normal Attach processing

The first input denotes whether a record has been sent (S) or received (R) by RM, and the second input denotes the particular record type. PRI (primary), SEC (secondary), and SECURE (session-level security) are half-session attributes.

FSM_SCB_STATUS_BIDDER

FUNCTION: To remember the status of a bidder half-session

NOTES:

1. The initial state of this FSM is SESSION_ACTIVATION.

2. When HS on the bidder side of a session receives an MU containing an Attach, HS generates a separate BID and sends it to RM. RM (bidder side) always sends a positive BID_RSP to HS (unless a protocol error has occurred). HS (bidder side) discards the BID_RSP and then sends the Attach MU to RM. RM on the first-speaker side does not generate the BID record, and does not expect a BID_RSP, since a first-speaker half-session always gains access to the session.

3. A YIELD_SESSION changes the FSM from SESSION_ACTIVATION state to the IN_USE state. A FREE_SESSION record is expected from the half-session to then cause RM to change the state to FREE.

INPUTS — STATE NAMES----> / STATE NUMBERS-->	SESSION ACTIVATION 01	FREE 02	PENDING ATTACH 03	IN USE 04	PENDING FMH12 05
R, POS_BID_RSP	4	4	/	/	/
R, BID	/	3	/	/	/
R, ATTACH	/	/	4	/	/
R, FMH_12	/	/	/	/	3
R, FREE_SESSION	/	/	2	2	/
S, YIELD_SESSION	4	/	/	/	/
R, SESSION_ACTIVATED, PRI	-	/	/	/	/
R, SESSION_ACTIVATED, SEC	3	/	/	/	/
R, SESSION_ACTIVATED, SECURE	5	/	/	/	/

FSM_SCB_STATUS_FSP

FUNCTION: To remember the status of a first-speaker half-session

NOTES: 1. The initial state of this FSM is SESSION_ACTIVATION.

2. A YIELD_SESSION changes the FSM from SESSION_ACTIVATION state to the IN_USE state. A FREE_SESSION record is expected from the half-session to then cause RM to change the state to FREE.

STATE NAMES----> INPUTS STATE NUMBERS-->	SESSION ACTIVATION 01	FREE 02	PENDING ATTACH 03	IN USE 04	PENDING FMH12 05
S, GET_SESSION	4	4	/	/	/
R, BID	/	3	/	/	/
R, ATTACH	/	/	4	/	/
R, FMH_12	/	/	/	/	3
R, FREE_SESSION	/	/	2	2	/
S, YIELD_SESSION	4	/	/	/	/
R, SESSION_ACTIVATED, PRI	-	/	/	/	/
R, SESSION_ACTIVATED, SEC	3	/	/	/	/
R, SESSION_ACTIVATED, SECURE	5	/	/	/	/

#FSM_BIS

#FSM_BIS is a generic FSM that maintains the state of the BIS protocol for a half-session. There is one #FSM_BIS for each session known to the resources manager. #FSM_BIS is initialized to either FSM_BIS_BIDDER or FSM_BIS_FSP, depending on the session polarity, when the resources manager becomes aware of the existence of a new session. This initialization occurs in CREATE_SCB (page 3-44).

The states of FSM_BIS_BIDDER and FSM_BIS_FSP are:

- RESET--the initial state; BIS has been neither sent nor received
- BIS SENT--the local LU has sent BIS
- BIS RCVD--the local LU has received BIS
- CLOSED--the local LU has both sent and received BIS

The first input denotes whether a record has been sent (S) or received (R) by RM, and the second input denotes the particular record type.

FSM_BIS_BIDDER

FUNCTION:		To remember the status of a bidder half-session with respect to BIS_RQ and BIS_REPLY
NOTES:	1.	The initial state of this FSM is RESET.
	2.	After BIS_RQ and BIS_REPLY have been exchanged over a session, this FSM will be in the CLOSED state, indicating that the session is being deactivated. The CLOSED state is a terminating state, in that the FSM will not leave this state until it (along with its corresponding SCB) is destroyed.

Referenced procedures, FSMs, and data structures:

SEND_DEACTIVATE_SESSION	page 3-68
CHECK_FOR_BIS_REPLY	page 3-40
BIS_RACE_LOSER	page 3-38
SEND_DEACTIVATE_SESSION	page 3-68
HS_ID	page 3-91

INPUTS STATE NAMES----> / STATE NUMBERS-->	RESET 01	BIS SENT 02	BIS RCVD 03	CLOSED 04
S, BIS_RQ	2	/	/	/
R, BIS_REPLY	>(ERROR)	4(A)	>(ERROR)	/
R, BIS_RQ	3(B)	4(C)	>(ERROR)	/
S, BIS_REPLY	/	/	4	/

OUTPUT CODE	FUNCTION
A	Call SEND_DEACTIVATE_SESSION(ACTIVE, HS_ID, NORMAL, X'00000000') (page 3-68).
B	Call CHECK_FOR_BIS_REPLY(HS_ID) (page 3-40).
C	Call BIS_RACE_LOSER(HS_ID) (page 3-38).
ERROR	Call SEND_DEACTIVATE_SESSION(ACTIVE, HS_ID, ABNORMAL, X'20100000') (page 3-68).

FSM_BIS_FSP

FUNCTION: To remember the status of a first-speaker half-session with respect to BIS_RQ and BIS_REPLY

NOTES:
1. The initial state of this FSM is RESET.
2. After BIS_RQ and BIS_REPLY have been exchanged over a session, this FSM will be in the CLOSED state, indicating that the session is being deactivated. The CLOSED state is a terminating state, in that the FSM will not leave this state until it (along with its corresponding SCB) is destroyed.

Referenced procedures, FSMs, and data structures:

SEND_DEACTIVATE_SESSION	page 3-68
CHECK_FOR_BIS_REPLY	page 3-40
HS_ID	page 3-91

INPUTS — STATE NAMES----> / STATE NUMBERS-->	RESET 01	BIS SENT 02	BIS RCVD 03	CLOSED 04
S, BIS_RQ	2	/	/	/
R, BIS_REPLY	>(ERROR)	4(A)	>(ERROR)	/
R, BIS_RQ	3(B)	-	>(ERROR)	/
S, BIS_REPLY	/	/	4	/

OUTPUT CODE	FUNCTION
A	Call SEND_DEACTIVATE_SESSION(ACTIVE, HS_ID, NORMAL, X'00000000') (page 3-68).
B	Call CHECK_FOR_BIS_REPLY(HS_ID) (page 3-40).
ERROR	Call SEND_DEACTIVATE_SESSION(ACTIVE, HS_ID, ABNORMAL, X'20100000') (page 3-68).

#FSM_RCB_STATUS

#FSM_RCB_STATUS is a generic FSM that maintains the state of a conversation resource. There is one #FSM_RCB_STATUS for each conversation known to the resources manager. When resources manager creates the conversation resource, #FSM_RCB_STATUS is initialized to either FSM_RCB_STATUS_BIDDER or FSM_RCB_STATUS_FSP, depending on the polarity of the underlying session. This initialization occurs in BIDDER_PROC (page 3-37), CREATE_RCB (page 3-43), and PS_CREATION_PROC (page 3-55).

The states of FSM_RCB_STATUS_BIDDER and FSM_RCB_STATUS_FSP are:

- FREE--the initial state; the conversation is inactive
- IN USE--the conversation is in progress
- PENDING SCB (BIDDER only)--the conversation is awaiting allocation of a session, pending receipt of RSP(Bid)

The first input denotes whether a record has been sent (S) to RM by PS or received (R) by RM from HS, and the second input denotes the particular record type. HS (half-session) represents the sender of the Attach.

FSM_RCB_STATUS_BIDDER

FUNCTION: To remember the status of a conversation resource associated with a bidder half-session

NOTES:

1. The initial state of this FSM is FREE.
2. The RCB may be in the FREE state when a DEALLOCATE_RCB is issued if RM discovers that an ALLOCATION_ERROR exists before it attempts to get a session for the transaction program. The ALLOCATION_ERRORs that can occur in this situation are ALLOCATION_FAILURE_* and SYNC_LEVEL_NOT_SUPPORTED_BY_LU.

INPUTS / STATE NAMES----> / STATE NUMBERS-->	FREE 01	IN USE 02	PENDING SCB 03
S, GET_SESSION	3	/	/
R, POS_BID_RSP	/	/	2
R, NEG_BID_RSP	/	/	1
R, ATTACH, HS	2	/	/
S, DEALLOCATE_RCB	-	1	/

FSM_RCB_STATUS_FSP

FUNCTION: To remember the status of a conversation resource associated with a first-speaker half-session

NOTES:

1. The initial state of this FSM is FREE.
2. The RCB may be in the FREE state when a DEALLOCATE_RCB is issued if RM discovers that an ALLOCATION_ERROR exists before it attempts to get a session for the transaction program. The ALLOCATION_ERRORs that can occur in this situation are ALLOCATION_FAILURE_* and SYNC_LEVEL_NOT_SUPPORTED_BY_LU.

STATE NAMES----> INPUTS STATE NUMBERS-->	FREE 01	IN USE 02
S, ALLOCATE_RCB S, GET_SESSION	- 2	/ /
R, ATTACH, HS	2	/
S, DEALLOCATE_RCB	-	1

LOCAL DATA STRUCTURES

LU_NAME

LU_NAME: LU name of a partner LU

MODE_NAME

MODE_NAME: mode name

HS_ID

HS_ID: half-session identifier

RCB_ID

RCB_ID: conversation resource identifier

TCB_ID

TCB_ID: TP-PS process identifier

SENSE_CODE

SENSE_CODE: 4-byte sense data

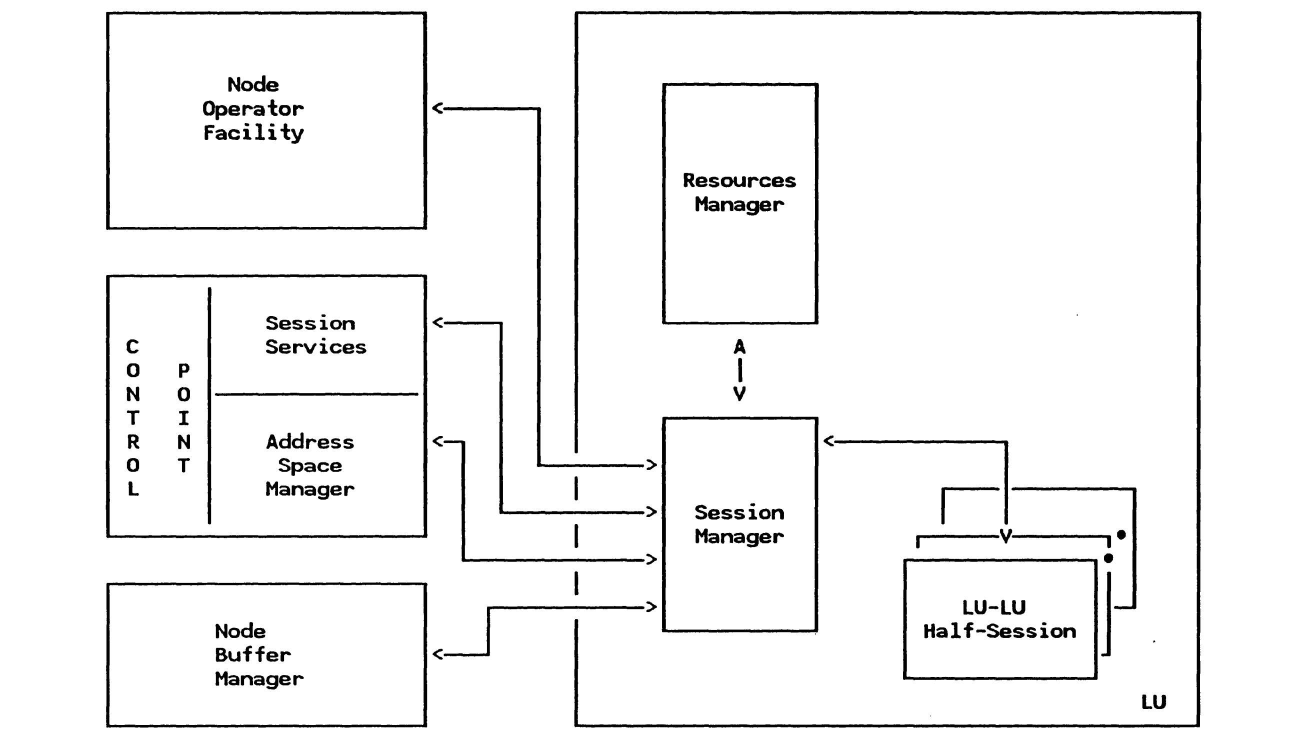
PREVIOUS_TIME

PREVIOUS_TIME: 48-bit time value

RESPONSE_CODE

RESPONSE_CODE: possible values: OK, PIP_NOT_ALLOWED, PIP_NOT_SPECIFIED_CORRECTLY, TPN_NOT_RECOGNIZED, TRANS_PGM_NOT_AVAILABLE_RETRY, TRANS_PGM_NOT_AVAIL_NO_RETRY, SECURITY_NOT_VALID, PS_CREATION_FAILURE, INVALID_FULLY_QUALIFIED_LU_NAME);

CHAPTER 4. LU SESSION MANAGER



Note: Only the components having a protocol boundary with LU session manager are shown.

Figure 4-1. Protocol Boundaries between LU Session Manager and Other Node Components

GENERAL DESCRIPTION

This chapter describes the session manager (SM) component within an LU. Figure 4-1 shows the LU session manager and its relation to other components within the node. The arrows joining the components represent the protocol boundaries that exist between SM and the other components.

The LU session manager initiates and terminates sessions in response to requests from the resources manager and from the remote LU.

The initiation and termination of sessions involves exchanging records between the LU and the local control point (CP), and exchanging session-control RUs between the LU and a partner LU. The exchange of session-control RUs performs the actual activation and deactivation of the sessions. The exchange of records between the LU and CP precedes and follows the activation and deactivation of the sessions.

Session-control requests and responses are sent on the expedited flow with the RU category indicating session control (SC). Full details of the formats are given in SNA Formats.

The LU resources manager (RM) in one of the partner LUs directs the activation or deactivation of an session. Upon completion of the activation or deactivation, SM in each of the two LUs informs its local RM that the session has been activated or deactivated.

OVERVIEW OF SESSION INITIATION

RM directs the LU to activate a session by sending SM an ACTIVATE_SESSION record across an internal protocol boundary. SM processes the ACTIVATE_SESSION record and initiates the session. The SM components in the two LUs activate the session by exchanging a BIND request and response. SM's processing of the ACTIVATE_SESSION record, which constitutes its part of the session initiation, includes the following:

- Check if by activating the session a session limit would be exceeded.
- Send the ASSIGN_PCID record to the session services (SS) component of the control point requesting a fully qualified procedure correlator identifier (FQPCID), which will uniquely identify the session and procedures related to the session. The FQPCID is assigned by the node initiating the session.
- Receive the ASSIGN_PCID_RSP record from SS. This record contains a Fully Qualified PCID (FQPCID) control vector.
- Send an INIT_SIGNAL record to SS. The request directs the control point to mediate the initiation of the session.
- Receive a CINIT_SIGNAL record from SS. That record contains the path control ID and characteristics (e.g., maximum send and receive BTU sizes) for the session and the information on what to include in the BIND.
- Send an ASSIGN_LFSID record to the address space manager (ASM) component of the control point. This record asks ASM to assign a local-form session identifier (LFSID) for the session. The LFSID logically connects a half-session (HS) with the path control (PC).
- Receive an ASSIGN_LFSID_RSP record with the LFSID for the session from ASM.
- Send the BIND with the desired parameters for the session to the partner (secondary) LU.
- Receive the RSP(BIND) with the negotiated session parameters from the partner. Check the admissibility of the negotiated parameters.
- Obtain buffers for the session from the node's buffer manager (BM). These buffers will be used by the half-session HS.
- Create and initialize the HS for this LU's side of the session.
- Notify RM that the requested session is active.

The partner LU of the one initiating the session is directed to activate the session by means of receiving the BIND. Its processing following receipt of the BIND includes the following:

- Obtain buffers for the session from BM. These buffers will be used by HS.
- Build a negotiated BIND response that specifies the agreed-to parameters for the session, and send the BIND response to the partner (primary) LU.
- Create and initialize the HS for this LU's side of the session.
- Notify SS that a new session is activated.
- Inform RM that a session has been activated at the request of the remote LU.

The parameters used for the session and carried in the BIND request and response have the following sources:

- Fixed parameters: These have fixed values for all BIND requests and responses for LU 6.2 sessions.
- Implementation-dependent parameters: These have values that are determined during the implementation of the node.
- Installation-specified parameters: These have values that are determined by the user at the node's installation.
- CINIT parameters: These have values taken from the CINIT_SIGNAL record and sent in the BIND.

OVERVIEW OF SESSION TERMINATION

RM directs the LU to deactivate a session by sending SM a DEACTIVATE_SESSION record across an internal protocol boundary. The two LUs deactivate their session by exchanging an UNBIND request and response and destroying their local half-sessions. SM's processing of the DEACTIVATE_SESSION record, which constitutes its part of the session termination, includes the following:

- Send an UNBIND to the partner LU and receive the RSP(UNBIND). SM deactivates the session upon sending the UNBIND, before getting the RSP(UNBIND).
- Notify SS that the session has been deactivated.
- Destroy the HS for this LU's side of the session.
- Inform BM that buffers previously reserved for the session are no longer needed.

The partner LU of the one terminating the session is directed to deactivate the session by means of the UNBIND. Its processing following receipt of the UNBIND is similar to the processing just outlined. SM at the UNBIND receiver informs RM that it has deactivated a session at the request of the remote LU.

SESSION OUTAGE AND SESSION REINITIATION

An active session between two LUs may be interrupted by a failure of one or both of the LUs, by an abort of one or both of their HSs, or by a failure of the path that connects the LUs. This interruption causes a session outage, and notification to the LU of the session outage is referred to as session-outage notification, or SON. When SM receives a SESSION_ROUTE_INOP record from ASM, it notifies RM, SS, and BM and destroys the HS for each session affected by the session outage.

When session outage occurs, RM may direct SM to reinitiate the sessions. See "Chapter 3. LU Resources Manager" and "Chapter 5.4. Presentation Services--Control-Operator Verbs" for more details.

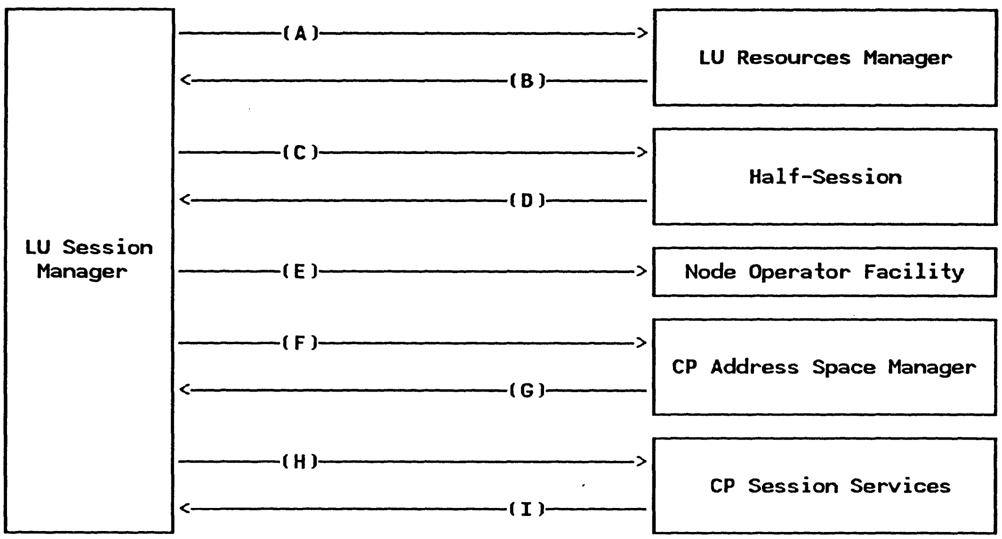
PLU AND SLU

PLU and SLU refer to the role of an LU in providing, respectively, primary or secondary half-session control for a session. The PLU sends the INIT_SIGNAL record, receives the CINIT_SIGNAL record, sends the BIND, and receives the RSP(BIND). Conversely, the SLU receives the BIND and sends the RSP(BIND).

SM PROTOCOL BOUNDARIES

This section describes the protocol boundaries (PBs) that SM has with various LU and other node components. SM interacts with them by exchanging records. Figure 4-2 shows the protocol boundaries and lists the record names associated with them. The procedures and finite-state machines (FSMs) of this chapter describe SM's protocols for sending and receiving these records. See "Appendix A. Node Data Structures" for a definition of the formats of these records.

In addition, SM interacts with the node's buffer manager. See "SM and Buffer Management" on page 4-28 for a detailed description of this protocol boundary.



Records that SM sends:

(A) ACTIVATE_SESSION_RSP
SESSION_ACTIVATED
SESSION_DEACTIVATED

(C) INIT_HS

(E) RM_CREATED

(F) MU (contains the following RUs)
BIND
UNBIND
±RSP(BIND)
±RSP(UNBIND)
PC_HS_DISCONNECT
ASSIGN_LFSID
FREE_LFSID
LFSID_IN_USE_RSP

(H) ASSIGN_PCID
INIT_SIGNAL
SESSST_SIGNAL
SESSEND_SIGNAL

Records that SM receives:

(B) ACTIVATE_SESSION
DEACTIVATE_SESSION
ABEND_NOTIFICATION

(D) INIT_HS_RSP
ABORT_HS
ABEND_NOTIFICATION

(G) MU (contains the following RUs)
BIND
UNBIND
±RSP(BIND)
SESSION_ROUTE_INOP
ASSIGN_LFSID_RSP
LFSID_IN_USE

(I) ASSIGN_PCID_RSP
INIT_SIGNAL_NEG_RSP
CINIT_SIGNAL

Figure 4-2. Records Exchanged between SM and Other Components

PB WITH RM

This section describes the records that SM exchanges with RM.

The following table lists each record and the page number of its description.

ACTIVATE_SESSION

Flow: From RM to SM

ACTIVATE_SESSION instructs SM to activate a session with a specified partner LU using a given mode name. RM expects a response (positive or negative) to this request.

DEACTIVATE_SESSION

Flow: From RM to SM

DEACTIVATE_SESSION instructs SM to deactivate a specific session. SM correlates this request to a specific session either by HS_ID (a half-session process identifier), if RM has already been informed that the session is activated, or by the CORRELATOR parameter, which matches that from an ACTIVATE_SESSION record.

ABEND_NOTIFICATION

Flow: From RM to SM

ABEND_NOTIFICATION informs SM that RM has abended. SM will bring down all of its sessions, and will inform all affected components in the node.

ACTIVATE_SESSION_RSP

Flow: From SM to RM

ACTIVATE_SESSION_RSP tells RM whether or not SM was able to satisfy RM's request to activate a session. If positive, ACTIVATE_SESSION_RSP contains a half-session process identifier, which will be used by RM to identify the session. If negative, ACTIVATE_SESSION_RSP uses an ERROR_TYPE parameter to inform RM whether another attempt to activate a session might succeed.

SESSION_ACTIVATED

Flow: From SM to RM

SESSION_ACTIVATED tells RM that SM has activated a session at the request of a partner LU.

SESSION_DEACTIVATED

Flow: From SM to RM

SESSION_DEACTIVATED tells RM that SM has deactivated a session either at the request of a partner LU, because of a detected protocol violation, or following a link failure.

PB WITH HS

This section describes the records that SM exchanges with HS.

The following table lists each record and the page number of its description.

INIT_HS

Flow: From SM to HS

INIT_HS gives HS all the session information it needs to begin to perform its functions. This information includes the values of the negotiated session parameters (see details in the later description of the BIND request and response RUs) and the buffer pool identifiers for the buffers obtained by SM on behalf of this HS. The buffer identifier is a pointer to a buffer manager control block for the buffer pool.

INIT_HS_RSP

Flow: From HS to SM

INIT_HS_RSP tells SM whether or not the HS is successfully initialized. The only reason an HS activation could fail is the failure of the cryptography verification.

ABORT_HS

Flow: From HS to SM

ABORT_HS informs SM that a HS has abnormally terminated because of a protocol violation. SM will bring down the session and inform the partner LU, and the RM and SS components in its own node of the condition.

ABEND_NOTIFICATION

Flow: From HS to SM

ABEND_NOTIFICATION informs SM that HS has abnormally terminated. SM will bring down all sessions associated with the HS, and will inform RM and SS in this node of the condition.

PB WITH NOF

This section describes the records that SM exchanges with NOF.

The following table lists each record and the page number of its description.

RM_CREATED

Flow: From SM to NOF

RM_CREATED informs NOF that SM has successfully created an RM process so that NOF can start its first transaction program on the LU. This record is sent by SM only once during it's initialization stage.

PB WITH SS

This section describes the records that flow between SM and the session services component of the CP.

The following table lists each record and the page number of its description.

ASSIGN_PCID

Flow: From SM to SS

ASSIGN_PCID record is sent by the session manager to the session services (SS) component of the node control point. ASSIGN_PCID asks SS to create a fully qualified procedure correlator (FQPCID) that will serve as a unique identifier for this session. This record is sent after SM receives a request to activate a session from RM or when it receives a BIND that does not contain an FQPCID control vector. ASSIGN_PCID asks SS to create an FQPCID that will serve as a local session identifier. In this latter case, FQPCID will never be sent to the partner LU. ASSIGN_PCID requires a response from SS. SM cannot accept any other signal related to any session until a response to a sent ASSIGN_PCID is received.

ASSIGN_PCID_RSP

Flow: From SS to SM

ASSIGN_PCID_RSP record is sent to the session manager by the session services component of the node control point. This record is sent after SS receives a request from the session manager to assign an FQPCID control vector for the session.

When SM receives the ASSIGN_PCID_RSP record, it compares the PCID portion of the received FQPCID with those of all of its active and pending-active sessions. If a duplicate PCID is found, SM sends another ASSIGN_PCID record to SS indicating that it has discovered a PCID collision. This will continue until SS returns an ASSIGN_PCID_RSP record with an FQPCID whose PCID portion is not duplicated.

INIT_SIGNAL

Flow: From SM to SS

INIT_SIGNAL requests that the CP assist in the initiation of a session between the LU sending the request (the PLU) and the LU named in the request (the SLU). The INIT_SIGNAL requires a response from the control point.

The INIT_SIGNAL request contains, among other parameters, the fully qualified network names of the PLU and the SLU, the mode name for the session, and the FQPCID for the session.

The CP returns either a CINIT_SIGNAL record or an INIT_SIGNAL_NEG_RSP record.

INIT_SIGNAL_NEG_RSP

Flow: From SS to SM

INIT_SIGNAL_NEG_RSP is sent by SS to SM in response to an INIT_SIGNAL record. INIT_SIGNAL_NEG_RSP tells the PLU that a request to activate a session is rejected. SM will then inform RM that it couldn't activate a session.

CINIT_SIGNAL

Flow: From SS to SM

CINIT_SIGNAL is sent by SS to SM in response to an INIT_SIGNAL record. CINIT_SIGNAL instructs the PLU to send a BIND to the SLU and provides the information that the PLU needs in order to generate the BIND RU. See the description of BIND in this chapter for the rules of how the PLU chooses the BIND parameters.

The PLU uses the FQPCID control vector field in the CINIT_SIGNAL record to correlate it to a previously sent INIT_SIGNAL. This field is always present in the CINIT_SIGNAL. The Class-of-Service/Transmission_Priority control vector (COS_TPF) may be present in the CINIT_SIGNAL record. If it is present, the PLU puts it in the BIND without modification.

SESSST_SIGNAL

Flow: From SM to SS

SESSST_SIGNAL notifies SS that a session has been successfully activated. This record is sent to SS after SM sends a positive response to BIND.

SESSST_SIGNAL is sent by SM only from the SLU side. On the PLU side, it is not necessary because SS assumes after sending a CINIT_SIGNAL record that the session will be activated. If the session activation fails after the CINIT_SIGNAL is received, the PLU's SM sends a SESSEND_SIGNAL record to SS.

SESSEND_SIGNAL

Flow: From SM to SS

SESSEND_SIGNAL notifies SS that a session has been successfully deactivated or that an attempt to activate a session has failed. The former case applies to both PLU and SLU; the latter case applies only to the PLU in the situation when CP sent a CINIT_SIGNAL record to the PLU but the PLU could not successfully complete the session activation.

SESSEND_SIGNAL carries the FQPCID control vector field and the sense data information.

PB WITH ASM

This section describes the records that flow between the SM and the Address Space Manager component of the CP.

The following table lists each record and the page number of its description.

MESSAGE UNIT (MU)

Flow: From SM to ASM and from ASM to SM

Message unit (MU) records carrying the TH, RH, and session-activation and session-deactivation RUs are exchanged between SM and the address space manager component of the control point in both directions.

SM receives a BIND MU in a session buffer that may be reused by the SM to send the response to the BIND, whether it be a ±RSP(BIND) or an UNBIND. All other MU types arrive at SM in a link buffer that SM frees immediately. For a description of buffer types, refer to "SM and Buffer Management" on page 4-28.

BIND, RSP(BIND), UNBIND, and RSP(UNBIND) RUs are described in greater detail later in this chapter. The format of MU records is defined in Appendix A.

PC_HS_DISCONNECT

Flow: From SM to ASM

PC_HS_DISCONNECT record is sent by SM to ASM after SM receives -RSP(BIND). PC_HS_DISCONNECT instructs ASM to free the LFSID (local-form session identifier) used for the session. The reason why this record is sent only after a -RSP(BIND) is received is that in all other cases SM sends either an UNBIND or a RSP(UNBIND) before the session is terminated. Both of these MUs contain a FREE_LFSID field, which SM sets to instruct ASM to free the LFSID.

SESSION_ROUTE_INOP

Flow: From ASM to SM

SESSION_ROUTE_INOP tells SM to terminate each session with an LULU_CB control block entry that has the PATH_CONTROL_ID parameter equal to the corresponding value received in the SESSION_ROUTE_INOP record.

ASSIGN_LFSID

Flow: From SM to ASM

ASSIGN_LFSID is sent by SM to the address space manager component of the control point after SM receives the CINIT_SIGNAL record. The purpose of this record is to request ASM to assign an LFSID for the session. ASSIGN_LFSID requires a response from ASM. SM cannot accept any other signal related to any session until a response to a sent ASSIGN_LFSID is received.

ASSIGN_LFSID_RSP

Flow: From ASM to SM

ASSIGN_LFSID_RSP is received by SM from the address space manager component of the control point in response to ASSIGN_LFSID. ASSIGN_LFSID_RSP carries the LFSID for the session, except when ASM was unable to assign it. In this latter case, ASM sets the sense data field in the record to a nonzero value.

FREE_LFSID

Flow: From SM to ASM

FREE_LFSID is sent by SM to the address space manager component of the control point when SM gets an ASSIGN_LFSID_RSP record from ASM but SM is unable to send a BIND because of an error encountered during initiation, such as lack of a BIND buffer.

LFSID_IN_USE

Flow: From ASM to SM

LFSID_IN_USE is received by SM from the address space manager component of the control point when ASM needs SM to check whether a certain (PATH_CONTROL_ID, LFSID) pair is in use by the session manager. ASM may need to check if SM is using an LFSID because of the following race condition. Upon sending an UNBIND, the UNBIND sender may reuse an LFSID for a subsequent BIND that arrives at the partner LU before the partner has fully processed the preceding UNBIND and freed the LFSID for re-use on its side. In this case, the receiving ASM checks on the LFSID status with its SM (using this signal queued behind the pending UNBIND) to accommodate the race condition. See SNA Type 2.1 Node Reference for further details.

LFSID_IN_USE_RSP

Flow: From SM to ASM

LFSID_IN_USE_RSP is sent by SM to the address space manager component of the control point in response to the LFSID_IN_USE record. It contains a parameter that indicates whether a (PATH_CONTROL_ID, LFSID) pair in question is used by SM.

See Figure 4-3 and Figure 4-4 on page 4-15 for a description of TH parameters and Figure 4-5 on page 4-16 and Figure 4-6 on page 4-17 for a description of RH parameters in MUs that SM sends and receives. The meaning of the individual TH and RH bits is described in SNA Formats.

Offset in TH		Field Name	Value	
Byte 0	Bits 0-3 Bit 4 Bit 5 Bit 6 Bit 7	FID BBIUI EBIUI ODAI EFI	(Note) BBIU EBIU 0 EXP	BIND|UNBIND
Byte 1	Bits 0-7	reserved	00000000	
Byte 2	Bits 0-7	DAF'	00000000	
Byte 3	Bits 0-7	OAF'	00000000	
Bytes 4-5		SNF	unique sequence number	
Byte 0	Bits 0-3 Bit 4 Bit 5 Bit 6 Bit 7	FID BBIUI EBIUI ODAI EFI	(Note) BBIU EBIU 0 EXP	RSP(BIND)|RSP(UNBIND)
Byte 1	Bits 0-7	reserved	00000000	
Byte 2	Bits 0-7	DAF'	00000000	
Byte 3	Bits 0-7	OAF'	00000000	
Bytes 4-5		SNF	sequence number from the request	

Note: The FID type is set by path control (PC).

Figure 4-3. TH Parmeters for MUs That SM Sends.

Offset in TH	Field Name	Value	
Byte 0 Bits 0-3	FID	(Note 1)	
Bit 4	BBIUI	(Note 1)	
Bit 5	EBIUI	(Note 1)	
Bit 6	ODAI	(Note 1)	
Bit 7	EFI	(Note 1)	
Byte 1 Bits 0-7	reserved	(Note 1)	BIND|UNBIND
Byte 2 Bits 0-7	DAF'	(Note 1)	
Byte 3 Bits 0-7	OAF'	(Note 1)	
Bytes 4-5	SNF	unique sequence number; copied in the RSP, if sent (Note 4)	
Byte 0 Bits 0-3	FID	(Note 1)	
Bit 4	BBIUI	(Note 1)	
Bit 5	EBIUI	(Note 2)	
Bit 6	ODAI	(Note 1)	
Bit 7	EFI	(Note 1)	
Byte 1 Bits 0-7	reserved	(Note 1)	RSP(BIND) (Note 4)
Byte 2 Bits 0-7	DAF'	(Note 1)	
Byte 3 Bits 0-7	OAF'	(Note 1)	
Bytes 4-5	SNF	(Note 3)	

Notes:
1. This parameter is checked by either PC or ASM not by SM.
2. If a +RSP(BIND) is received with EBIUI = ¬EBIU, SM sends an UNBIND except when SM has already lost its awareness of the session, in which case it merely discards the received response.
3. The SNF parameter correlates the response with the previously sent BIND.
4. The session clean-up is done when the UNBIND is sent by SM; SM does not wait for the RSP(UNBIND).

Figure 4-4. TH Parameters for MUs That SM Receives

BIND|UNBIND

Offset in RH		Field Name	Value
Byte 0	Bit 0	RRI	RQ
	Bits 1-2	RU Category	SC
	Bit 3	reserved	0
	Bit 4	FI	FMH
	Bit 5	SDI	¬SD
	Bit 6	BCI	BC
	Bit 7	ECI	EC
Byte 1	Bit 0	DR1I	DR1
	Bit 1	reserved	0
	Bit 2	DR2I	¬DR2
	Bit 3	ERI	¬ER
	Bit 4	reserved	0
	Bit 5	RLWI	¬RLW
	Bit 6	QRI	¬QR
	Bit 7	PI	¬PAC
Byte 2	Bit 0	BBI	¬BB
	Bit 1	EBI	¬EB
	Bit 2	CDI	¬CD
	Bits 3-6	reserved	0000
	Bit 7	CEBI	¬CEB

RSP(BIND)| RSP(UNBIND)

Offset in RH		Field Name	Value
Byte 0	Bit 0	RRI	RSP
	Bits 1-2	RU_CTGY	SC
		Category	SC
	Bit 3	reserved	0
	Bit 4	FI	FMH
	Bit 5	SDI	(Note)
	Bit 6	BCI	BC
	Bit 7	ECI	EC
Byte 1	Bit 0	DR1I	DR1
	Bit 1	reserved	0
	Bit 2	DR2I	¬DR2
	Bit 3	RTI	±
	Bits 4-5	reserved	00
	Bit 6	QRI	¬QR
	Bit 7	PI	¬PAC
Byte 2	Bits 0-7	reserved	00000000

Note: SDI is set to SD when a -RSP(BIND|UNBIND) is sent by SM; otherwise, it is set to ¬SD.

Figure 4-5. RH Parameters for MUs That SM Sends

Offset in RH		Field Name	Value	
Byte 0	Bit 0	RRI	(Note 2)	A
	Bits 1-2	RU	(Note 2)	
		Category	(Note 2)	
	Bit 3	reserved	(Note 2)	
	Bit 4	FI	(Note 2)	
	Bit 5	SDI	¬SD	
	Bit 6	BCI	(Note 2)	
	Bit 7	ECI	(Note 2)	
Byte 1	Bit 0	DR1I	(Note 2)	BIND|UNBIND
	Bit 1	reserved	(Note 2)	
	Bit 2	DR2I	(Note 2)	
	Bit 3	ERI	(Note 2)	
	Bit 4	reserved	(Note 2)	
	Bit 5	RLWI	(Note 2)	
	Bit 6	QRI	(Note 2)	
	Bit 7	PI	(Note 2)	
Byte 2	Bit 0	BBI	(Note 2)	
	Bit 1	EBI	(Note 2)	
	Bit 2	CDI	(Note 2)	
	Bits 3-6	reserved	(Note 2)	
	Bit 7	CEBI	(Note 2)	V
Byte 0	Bit 0	RRI	(Note 2)	A
	Bits 1-2	RU	(Note 2)	
		Category	(Note 2)	
	Bit 3	reserved	(Note 2)	
	Bit 4	FI	(Note 2)	
	Bit 5	SDI	(Note 1)	
	Bit 6	BCI	(Note 2)	
	Bit 7	ECI	(Note 2)	RSP(BIND) (Note 3)
Byte 1	Bit 0	DR1I	(Note 2)	
	Bit 1	reserved	(Note 2)	
	Bit 2	DR2I	(Note 2)	
	Bit 3	RTI	±	
	Bits 4-5	reserved	(Note 2)	
	Bit 6	QRI	(Note 2)	
	Bit 7	PI	(Note 2)	
Byte 2	Bits 0-7	reserved	(Note 2)	V

Notes:
1. SDI is set to SD when a -RSP(BIND) is received by SM; otherwise, it is set to ¬SD.
2. This parameter is checked by either PC or ASM, not by SM.
3. The session clean-up is done when the UNBIND is sent by SM; SM does not wait for the RSP(UNBIND).

Figure 4-6. RH Parameters for MUs That SM Receives

The following sections define some parameters that are common to many interprocess records and session-control field-formatted RUs.

NETWORK-QUALIFIED NAME

A network-qualified name is the name by which an LU is known throughout an interconnected SNA network. An interconnected network comprises one or more individual subnetworks. A network-qualified name consists of a network identifier (identifying the individual subnetwork) and a network LU name. Network-qualified names are unique throughout an interconnected network.

LOCAL NAME

A local name is any name by which a transaction program at one LU knows another LU. A local name requires interpretation (or transformation) by SM in order to yield the network qualified name of the partner LU. A local name may be the same as a network-qualified name.

MODE NAME

The CP has information about the LU partner that aids in the construction of the BIND parameters (carried in CINIT_SIGNAL). The CP and SM derive additional information from the mode name. The local LU supplies the mode name in the INIT_SIGNAL record.

LU-LU VERIFICATION DATA

Random data and enciphered data are used for LU-LU verification. Random data is randomly generated data of symbol-string type G. Enciphered data is the enciphered version of the random data.

If LU-LU verification is active, BIND and RSP(BIND) will contain 8 bytes of random data generated by the sender for LU-LU verification. RSP(BIND) will also contain 8 bytes of enciphered data for the same purpose. The secondary LU submits the random data received in the BIND along with the LU-LU password (refer to "Security" on page 2-9) to the Data Encryption Standard (DES) algorithm to obtain enciphered data. This enciphered data is inserted in the RSP(BIND) along with new random data. When the primary LU receives the RSP(BIND), it compares the received enciphered data with a copy of the same random data that it has also enciphered using its copy of the LU-LU password and the DES algorithm. If they are identical, the primary LU has verified that the SLU has the correct LU-LU password.
Up to this point in the LU-LU verification process, processing has been done by the SM of each LU. SM in the primary and secondary LUs send to their respective RMs a record that contains the random data from the RSP(BIND). The primary LU's RM enciphers the random data using the LU-LU password and the DES algorithm, inserts it in a Security FM header (refer to "Chapter 3. LU Resources Manager" in Chapter 3) and sends it to the secondary's RM. The received enciphered data is compared with the secondary LU's version of the enciphered data. If they are identical, the secondary LU has verified that the primary LU has the correct LU-LU password, which completes the process.

SPECIFICATION OF RU PARAMETERS

Throughout the descriptions of the RUs in this chapter, reference is made to the specification of a parameter. "Specification" refers to a specific value that is supplied for the parameter when the RU is being built, prior to its being sent.

Implementation-Dependent Parameters

Throughout the descriptions of the RUs in this chapter, reference is made to implementation-dependent parameters. "Implementation-dependent" means that the particular value, or values, that a parameter of an RU can take is determined by each implementation.

Installation-Specified Parameters

Throughout the descriptions of the RUs in this chapter, reference is made to installation-specified parameters. "Installation-specified" means that the particular value, or values, that a parameter of an RU can take is determined by the installation owner or manager. Installation-specified values can be established during system definition of a node, or later during its operation. The method for establishing values of installation-specified parameters is implementation-dependent.

SESSION-CONTROL RU'S

This section describes the session-control requests and responses that SM sends and receives.

Each RU description includes the RU flow and a discussion of the function and use of the RU. Refer to SNA Formats for specifications of the RU formats.

The table below lists each session control RU that pertains to session activation and deactivation, and the page number of its description.

BIND

Flow: From PLU to SLU (Expedited)

BIND is sent from a PLU to an SLU to activate a session between the LUs. The BIND indicates definite-response requested.

The BIND carries the PLU's suggested parameters for the session. The specifications of the BIND parameters are based on required LU 6.2 values, on the PLU's implementation-dependent support, on the installation-specified values currently in effect for the parameters, or on the CINIT, depending on the particular parameter.

The SLU uses the BIND parameters to help determine whether it will send back a +RSP(BIND), an UNBIND, or a -RSP(BIND). The SLU's use of UNBIND or -RSP(BIND) to reject a BIND is based on whether the SLU is a current-level or back-level node. Control information in either LU is updated only when a positive response is returned. A successful BIND causes a half-session to be created at both PLU and SLU.

If the LU receives a BIND after sending a BIND, and either (1) parallel sessions between the two LUs are not supported, or (2) the current number of active sessions within the mode-name group is 1 less than the session limit for that group, then a BIND race has occurred. The BIND race is resolved by comparing the network-qualified LU names in the BINDs. Network-qualified LU names are unique throughout a network; therefore, one will always compare greater than the other in the EBCDIC collating sequence of the two names. The LU that sent the BIND containing the greater of the two network-qualified LU names is the winner (its BIND is accepted). The other LU is the loser (its BIND is rejected).

The comparison is made by comparing the two names as EBCDIC character strings, left-justified, and filled on the right with space (X'40') characters, if necessary. Network-qualified LU names contain no leading or embedded space characters.

The LU winning the BIND race sends back an UNBIND or a -RSP(BIND). The other LU sends back a +RSP(BIND), unless the BIND is not acceptable for other reasons, such as invalid format.

The BIND and its response do not have an ERP type. The distinction between simple activation and resynchronizing reactivation following a failure is made after the session has been activated. In some cases, resync protocols are used; in others, end-user protocols are invoked, see "Chapter 5.3. Presentation Services--Sync Point Services Verbs" in Chapter 5.3.

The SLU does not necessarily reject the BIND because of any incompatibility it may have with the BIND parameters. Rather, if the BIND is otherwise acceptable (for example, there are no format errors and the session limit is not exceeded), the SLU returns a positive response with an extended format that carries the complete set of session parameters. The specifications for the parameters can match those received in the BIND, or they can differ, where the SLU chooses different options. The parameters for which the SLU may choose different options are referred to as negotiable parameters.

The PLU receives +RSP(BIND) and checks the parameter specifications. If acceptable, they are used for the activated session. Otherwise, the PLU sends UNBIND.

A description of the parameters in the BIND follows.

Format: This specifies the format of the BIND. Only one format is defined: Format 0.

Type: This specifies the type of BIND. The type is always specified as negotiable. The +RSP(BIND) has the same general format as the BIND. The negotiable type of BIND permits the SLU to return a positive response in which the negotiable parameters may differ from those in the request.

FM Profile: This specifies the FM profile to be used for the session. FM profile 19 is the only one defined for LU 6.2. The FM profile is supplemented by the FM usage parameters of the BIND.

TS Profile: This specifies the TS profile to be used for the session. TS profile 7 is the only one defined for LU 6.2. The TS profile is supplemented by the TS usage parameters of the BIND.

FM Usage (PLU)—Chaining Use: This specifies the PLU's use of chains that it sends to the SLU. Multiple-RU chains is the only use defined for LU 6.2. Chains may consist of one or more RUs. The maximum-size RU that the PLU sends and the verbs that the transaction program issues to the PLU determine the number of RUs that make up the chain.

FM Usage (PLU)—Request Control Mode: This specifies the PLU's protocol for sending chains. Immediate-request mode is the only protocol defined for LU 6.2. The PLU waits for a response to a definite-response chain before it sends another chain.

FM Usage (PLU)—Chain Response Protocol: This specifies the PLU's protocol for requesting responses to chains. Definite- or exception-response requested is the only protocol defined. A chain indicating definite-response requested requires a response from the SLU; the response may be positive or negative. A chain indicating exception-response requested requires a response from the SLU only when the response is negative; a positive response is not returned.

FM Usage (PLU)—Send End Bracket: This specifies that the PLU does not send EB chains.

FM Usage (SLU)—Chaining Use: This specifies the SLU's use of chains that it sends to the PLU. Multiple-RU chains is the only use defined for LU 6.2. Chains may consist of one or more RUs. The maximum-size RU that the SLU sends and the verbs that the transaction program issues to the SLU determine the number of RUs that make up the chain.

FM Usage (SLU)—Request Control Mode: This specifies the SLU's protocol for sending chains. Immediate-request mode is the only protocol defined for LU 6.2. The SLU waits for a response to a definite-response chain before it sends another chain.

FM Usage (SLU)—Chain Response Protocol: This specifies the SLU's protocol for requesting responses to chains. Definite- or exception-response requested is the only protocol defined. A chain indicating definite-response requested requires a response from the PLU; the response may be positive or negative. A chain indicating exception-response requested requires a response from the PLU only when the response is negative; a positive response is not returned.

FM Usage (SLU)—Send End Bracket: This specifies that the SLU does not send EB chains.

FM Usage (Common)—Whole BIU Required Indicator: The PLU specifies whether it supports receiving segmented BIUs on the session. BIU segmenting affects the specifications of the maximum-size RUs sent by the PLU and SLU. This field is set according to whether the segment reassembly support is specified in the node this LU belongs to. This support is installation-specified.

FM Usage (Common)—FM Header Usage: This specifies that FM headers are used on the session.

FM Usage (Common)—Bracket Usage and Reset State: This specifies that brackets are used on the session and that the bracket reset state for the session is in-bracket (INB); that is, the session is in the in-bracket state following successful activation.

FM Usage (Common)—Bracket Termination Rule: This specifies that rule 1, conditional termination, will be used on the session. The sender of the end-bracket (CEB) chain determines whether the bracket is to end conditionally or unconditionally. If conditional, the receiver is allowed to reject the end-bracket chain and thereby keep the session in the in-bracket state.

FM Usage (Common)--BIND Queuing: This specifies whether the SLU is permitted to queue (hold) the BIND for an indefinite period without sending a response. Whether the PLU permits the SLU to queue the BIND and delay its response is implementation-dependent. All sessions for which this LU is a PLU have the same specification for this parameter.

FM Usage (Common)—Normal-Flow Send/Receive Mode: This specifies that the send/receive protocol for FMD requests on the normal flow is half-duplex flip-flop.

FM Usage (Common)—Recovery Responsibility: This specifies the responsibility for recovery from an error within the session. Symmetric recovery is the only value defined for LU 6.2. The sender of a negative response is responsible for recovery, regardless of whether the sender is the PLU or SLU.

FM Usage (Common)—Contention Winner/Loser: This specifies whether the PLU or SLU will be the contention winner for the session. The contention winner is the brackets first speaker, and the contention loser is the bidder. The specification of contention winner or loser depends on whether the session is parallel or single session, as indicated by the PS usage parameter, Parallel Session Support, in the BIND.

For a parallel session, the PLU specifies that it is the contention winner if, for the mode name, the number of active sessions for which the PLU is the contention winner is less than its maximum; otherwise, the PLU specifies the SLU as the contention winner. The PLU's maximum number of contention-winner sessions is determined from the last change-number-of-sessions protocol executed by the two LUs.

For a single session, the PLU specifies that it is the contention winner if, for the mode name, the SLU is to be the contention loser; otherwise, the PLU specifies the SLU as the contention winner. For each mode name associated with a single-session LU, the contention winner (PLU or SLU) for the session is installation-specified. (Refer to Figure 5.4-14 on page 5.4-24).

FM Usage (Common)—Control Vectors Included Indicator: This specifies that at least one control vector is present in the BIND.

FM Usage (Common)—Half-Duplex Flip-Flop Reset States: This specifies the half-duplex flip-flop reset states for the PLU and SLU following successful activation of the session. The reset states are send for the PLU and receive for the SLU; that is, the PLU sends first.

TS Usage—Staging for Secondary TC to Primary TC: This specifies whether pacing of normal-flow requests from the SLU to the PLU occurs in one stage or more than one stage. One-stage pacing is always specified for LU 6.2. See "Chapter 6.2. Transmission Control" for details on session-level pacing.

TS Usage—Secondary TC's Send Window Size: This installation-specified value is set to the desired receive window size of the PLU.

TS Usage—Adaptive Session-Level Pacing: This specifies that the PLU supports adaptive session-level pacing for the session. See "Chapter 6.2. Transmission Control" for details on adaptive session-level pacing.

TS Usage—Secondary TC's Receive Window Size: This installation-specified value is set to the desired send window size of the PLU.

TS Usage—Maximum-Size RU Sent by SLU: If segment reassembly is supported, an installation-dependent value is put in the BIND for the mode name. Otherwise, this parameter in BIND is set by SM to a minimum of (1) that installation-dependent value and (2) path control's maximum receive size RU for the PLU node.

TS Usage—Maximum-Size RU Sent by PLU: If segment generation is supported, an installation-dependent value is put in the BIND for the mode name. Otherwise, this parameter in BIND is set by SM to a minimum of (1) that installation-dependent value and (2) path control's maximum send size RU for the PLU node.

TS Usage—Staging for Primary TC to Secondary TC: This specifies whether pacing of normal-flow requests from the PLU to the SLU occurs in one stage or more than one stage. One-stage pacing is always specified for LU 6.2.

TS Usage—Primary TC's Send Window Size: This installation-dependent value is set to the desired send window size of the PLU.

TS Usage—Primary TC's Receive Window Size: This installation-dependent value is set to the desired receive window size of the PLU.

PS Profile—PS Usage Format: This specifies the PS usage format. The basic format is the only PS usage format defined.

PS Profile—LU Type: This specifies type-6 as the LU type.

PS Usage—LU Type-6 Level: This specifies the level of LU type-6. Level 2 is the LU type-6 level defined for LU 6.2.

PS Usage—Security Manager Receive Function: This specifies whether the PLU supports a security manager for receiving a user-ID, password or already-verified indication, or profile-ID on FMH-5 Attach commands from the SLU.

PS Usage—Already Verified Indicator Acceptance: This specifies whether the PLU will accept the User-ID Already Verified indication on FMH-5 Attach commands from the SLU.

PS Usage—Synchronization Level: This specifies the level of synchronization support for the session. One of two levels of support may be specified:

1. Confirm
2. Confirm, Sync point, and Backout

The level of support specified for the session determines the synchronization levels that can be specified for a conversation allocated to the session. The synchronization level, "none" (not listed above), can be specified for a conversation allocated to any session; therefore, "none" is not explicitly specified for the session.

All LU implementations support the Confirm level; support for Sync point and Backout is implementation-dependent. If the PLU implementation supports Sync point and Backout, the specification of support-level 1 versus support-level 2 is installation-specified for each mode name. All sessions with the same mode name have the same specification for this parameter; however, the specification may differ for different mode names. See "Chapter 5.3. Presentation Services--Sync Point Services Verbs" for details about Sync point and Backout.

PS Usage—Responsibility for Session Reinitiation: This specifies the responsibility for reinitiation of a session following a session outage. This parameter applies only to sessions for which parallel sessions and change number of sessions (CNOS) are not supported. Four levels of responsibility are defined:

1. Operator controls.
2. Primary half-session reinitiates.
3. Secondary half-session reinitiates.
4. Either half-session may reinitiate.

Operator controlled reinitiation means neither LU will automatically attempt to reinitiate the session. The particular level of responsibility for reinitiation of the session--operator controlled or otherwise--can be either implementation-dependent or installation-specified.

Other events may cause a session to be activated, independent of the reinitiation responsibility. For example, if the resources manager has queued a request for allocation of a conversation, the resources manager will request activation of a session when the SM informs the resources manager that the current session has been deactivated.

PS Usage--Parallel-Session Support: This specifies that the PLU supports parallel sessions between the PLU and SLU.

PS Usage--Change-Number-Of-Sessions Support: This specifies whether the PLU and SLU support the change-number-of-sessions (CNOS) protocol, which includes exchange of the Change Number Of Sessions GDS variable. Support for CNOS is implementation-dependent; however, if parallel sessions are supported, CNOS is also supported. If the PLU implementation supports CNOS, the indication of support versus no support is installation-specified for each partner SLU. All sessions with the same SLU have the same specification for this parameter; however, the specification may differ for different SLUs.

Cryptography Options: This specifies whether session-level mandatory cryptography is supported for the session, and, if so, the cryptography options to be used. Support for session-level mandatory cryptography is implementation-dependent. If the mode name indicates support for session-level mandatory cryptography, then the PLU specifies in BIND that it is supported; otherwise, the PLU specifies it is not supported. All sessions with the same mode name have the same specification for this parameter; however, the specification may differ for different SLUs.

The cryptography options include a length parameter. The PLU indicates that session-level cryptography is not to be used for the session by specifying 0 for the length of the cryptography options. Session-level mandatory cryptography is the only session-level cryptography defined. See "Sessions with Cryptography" in "Chapter 6.2. Transmission Control" for additional information.

Primary LU Name: This specifies the name of the PLU for the session.

This parameter is not used by SM. Instead, SM uses the network-qualified PLU name carried in the user data to identify the PLU to the SLU.

User Data: This specifies, in a structured format, further parameters for the session.

Figure 4-7 shows the format of the user data and the preceding length. The user-data Key is always specified as X'00', which indicates structured subfields follow.

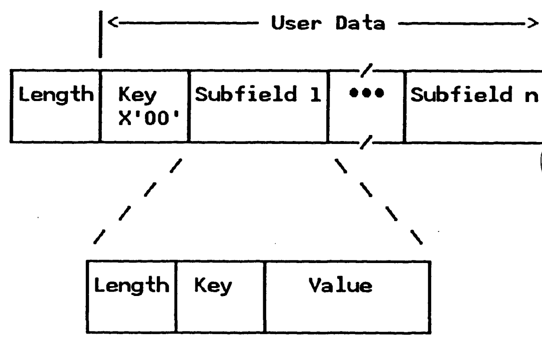


Figure 4-7. Format of User Data

Each subfield includes a length and is identified by a subfield key following the length. When more than one subfield are included, they appear in ascending order by subfield number.

The structured subfields that the PLU sends in BIND are:

Key	Name
X'00'	Unformatted Data
X'02'	Mode Name
X'03'	Session-Instance ID
X'04'	Network-Qualified PLU Name
X'11'	Random Data

A T2.1-node implementation that contains a single LU and a single link connection, does not support parallel sessions and CNOS, does not support the synchronization level for Sync point and Backout, and does not support LU-LU verification may omit all User-Data subfields. The PLU omits all User Data subfields either by specifying 0 for the length of the user data, or by specifying 1 for the length and specifying user data consisting only of the user-data key; the choice is implementation-dependent.

In general, the PLU may omit one or more subfields; see the descriptions of individual subfields for more information. If it does, the entire subfield, including its length, is omitted.

Details of each subfield follow.

- Subfield X'00'—Unformatted Data: This subfield carries installation-specified data. Support for this subfield is implementation-dependent.

- Subfield X'02'—Mode Name: Mode name specifies the type of service required for the session. Mode names are installation-specified. The same mode names are configured at both the PLU and SLU for all sessions between the two LUs. The installation-specified configuration for each mode name associates that mode name with the set of session properties to be used for all sessions for that mode name. The particular set of session properties associated with a mode name is implementation-dependent.

  A mode name may be null; that is, a null mode name is a valid mode name. When specifying a null mode name, the PLU may omit the Mode Name subfield entirely. Alternatively, the PLU may specify only the length and number for the null mode name, in which case the length is 1, or it may specify a mode name of all space (X'40') characters, which is equivalent to a null mode name. The particular form that the PLU uses to represent a null mode name is implementation-dependent.

  A T2.1-node implementation that contains a single LU and a single link connection, and that does not support parallel sessions and CNOS, may omit the Mode Name subfield entirely.

- Subfield X'03'—Session-Instance Identifier: The session-instance ID is used to uniquely identify the session from among the sessions where this LU is one of the partners. Using the session-instance ID, control operators at the PLU and SLU can coordinate the diagnostics (traces, for example) or clean-up procedures for a specific session. The session-instance ID is used also during resynchronization of a conversation after session outage.

  The LU that is the primary LU for a given session generates the session-instance ID. The first byte of the session-instance ID is set to X'01' to indicate that a PCID portion of the FQPCID control vector will be used for session identification.

  If the SLU does not use the FQPCID control vector (see below) for session identification, it will use the PLU-generated session-instance ID subfield to create a unique session identifier (see Subfield X'03' in the RSP(BIND) for more information).

- Subfield X'04'--Network-Qualified PLU Name: The network-qualified PLU name allows the PLU to identify itself to the SLU. The network-qualified PLU name is installation-specified at both the PLU and SLU.

  An LU resolves BIND-race conditions by comparing the network-qualified PLU name it sent in the BIND with the network-qualified PLU name it received in a BIND sent by the partner LU. BIND race conditions are discussed in more detail in the first part of this description of the BIND.

  A T2.1-node implementation that contains a single LU and a single link connection, does not support parallel sessions and CNOS, does not support the synchronization level for Sync point and Backout, and does not support LU-LU verification may have no network-qualified PLU name. In this case, the PLU omits the Network-Qualified PLU Name subfield from the BIND.

- Subfield X'11'—Random Data: This subfield is used when LU-LU verification is active. See "LU-LU Verification Data" on page 4-18 for more information on the function of random data.

User Request Correlation: Always omitted.

Secondary LU Name: This specifies the SLU name used to route the BIND to the intended SLU for the session.

A T2.1-node implementation that contains a single LU and a single link connection, does not support parallel sessions and CNOS, and is connected over the single link to another T2.1-node implementation containing a single LU and single link connection may omit the SLU name. The PLU omits the SLU name by specifying 0 for the length of the SLU name.

Control Vectors: The following control vectors may be included in the BIND:

- Fully Qualified PCID (FQPCID) control vector: This is a unique session identifier, always set in the BIND. Its value is given to SM by the session services component of the control point. A BIND that includes the FQPCID control vector is called an extended BIND, a BIND that does not include the FQPCID control vector is considered to have come from a back-level node.

- Class-of-Service/Transmission-Priority control vector (COS/TPF): This specifies the class of service for the session. It is included in the BIND if it is present in the CINIT.

Flow: From SLU to PLU (Expedited)

A (positive or negative) response to BIND is sent from an SLU to a PLU. A negative response is sent only when rejecting a BIND request that did not contain an FQPCID control vector. (When a BIND with an FQPCID control vector is rejected, an UNBIND is sent.) A -RSP(BIND) consists of sense data and the BIND request code; the remaining fields described below appear only on +RSP(BIND).

A back-level partner LU may send BIND or RSP(BIND) without control vectors and without indicating support for adaptive pacing. The receive support required for such back-level partners is shown in the procedural logic of the formal description. The rest of this section deals only with current-level support.

When the SLU receives a BIND that is acceptable (for example, there are no format errors and the SLU's session limit is not exceeded), the SLU sends back a +RSP(BIND) containing the complete set of session parameters. The specifications for the parameters can match those received in the BIND request, or they can differ, where the SLU chooses different options. The parameters for which the SLU may choose different options are referred to as negotiable parameters.

The specifications for the matching parameters are taken directly from the BIND. The specifications for the negotiable parameters are determined by the SLU based on its implementation-dependent support, on the installation-specified values currently in effect for the parameters, or on the BIND, depending on the particular parameter.

The following description of the RSP(BIND) parameters indicates the specifications that are used for the session and, where applicable, how they are determined. See the description of the corresponding parameters in the BIND for details of the function and use of the parameters.

Format: The SLU specifies format 0.

Type: The SLU specifies negotiable.

FM Profile: The SLU specifies FM profile 19.

TS Profile: The SLU specifies TS profile 7.

FM Usage (PLU)—Chaining Use: The SLU specifies multiple-RU chains.

FM Usage (PLU)—Request Control Mode: The SLU specifies immediate-request mode.

FM Usage (PLU)—Chain Response Protocol: The SLU specifies definite- or exception-response requested.

FM Usage (PLU)—Send End Bracket: The SLU specifies EB is not sent.

FM Usage (SLU)—Chaining Use: The SLU specifies multiple-RU chains.

FM Usage (SLU)—Request Control Mode: The SLU specifies immediate-request mode.

FM Usage (SLU)—Chain Response Protocol: The SLU specifies definite- or exception-response requested.

FM Usage (SLU)—Send End Bracket: The SLU specifies EB is not sent.

FM Usage (Common)—Whole BIU Required Indicator: The SLU specifies whether it supports receiving segmented RUs on the session. This support is installation-specified.

FM Usage (Common)—FM Header Usage: The SLU specifies FM headers are used.

FM Usage (Common)—Bracket Usage and Reset State: The SLU specifies brackets are used and the bracket reset state is in-bracket (INB).

FM Usage (Common)—Bracket Termination Rule: The SLU specifies rule 1, conditional termination.

FM Usage (Common)--BIND Queuing: This field is not used for RSP(BIND) since queuing of RSP(BIND) is not allowed.

FM Usage (Common)—Normal-Flow Send/Receive Mode: The SLU specifies half-duplex flip-flop.

FM Usage (Common)—Recovery Responsibility: The SLU specifies symmetric responsibility.

FM Usage (Common)—Contention Winner/Loser: This specification depends on whether the session is a parallel or single session, as indicated by the PS usage parameter, Parallel Session Support, in the RSP(BIND). For a parallel session, the specification is taken from the BIND--the SLU accepts, and does not change, the specification of the LU that is to be the contention winner for a parallel session.

For a single session, the SLU specifies that it is the contention winner if, for the mode name, the installation-defined specification is that the SLU is to be the contention winner; otherwise, the specification is taken from the BIND.

FM Usage (Common)—Control Vectors Included Indicator: This specifies whether at least one control vector is present in the RSP(BIND). An FQPCID control vector will be present if it was present in the BIND.

FM Usage (Common)—Half-Duplex Flip-Flop Reset States: The SLU specifies send for the PLU and receive for the SLU.

TS Usage—Staging for Secondary TC to Primary TC: Copied from the BIND request.

TS Usage—Secondary TC's Send Window Size: If adaptive pacing is supported by both this LU and the partner LU then this parameter is set to 0. The initial value of Secondary Send Window Size will be 1 when session traffic first starts flowing. The value of this parameter may change while the session is active.

If adaptive pacing is not supported by both ends of the session, then this value is taken from the BIND, as follows: If the BIND specifies one-stage pacing from the SLU to the PLU, this specification is taken from the Primary TC's Receive Window Size field; otherwise, this specification is taken directly from the Secondary TC's Send Window Size field.

TS Usage—Adaptive Session-Level Pacing: Copied from the BIND. That is, if adaptive pacing is requested, it will be used on this session. If not requested, fixed pacing will be used.

TS Usage—Secondary TC's Receive Window Size: If adaptive pacing is supported by both this LU and the partner LU, then this parameter is set to 0. The initial value of Secondary Receive Window Size will be 1 when session traffic first starts flowing. The value of this parameter may change while the session is active.

If adaptive pacing is not supported by both ends of the session, then this value is based on the BIND for the session and an installation-specified value associated with the mode name (this value is always greater than 0, as enforced by the control operator component of the LU), as follows:

- If the BIND for the session specifies a secondary TC's receive window size of 0, this specification is taken from the installation-specified value.

- If BIND specifies a window size other than 0, this specification is taken from the minimum of the value in BIND and the installation-specified value.

TS Usage—Maximum-Size RU Sent by SLU: The upper and lower bounds are set to installation-specified values. If the SLU's node does not support the segment generation or the PLU does not support segment regeneration, then SLU resets the upper bound to the minimum of the current upper bound and the path control's maximum RU size. The SLU specifies a value between a lower bound and the upper bound, as follows:

- If the value specified in the BIND is between the lower and upper bounds, the value in the RSP(BIND) is taken from the BIND.

- If the value specified in BIND is less than the lower bound, the SLU sets the value in the RSP(BIND) to the lower bound.

- If the value specified in BIND is greater than the upper bound, the SLU sets the value in the RSP(BIND) to the upper bound.

TS Usage—Maximum-Size RU Sent by PLU: The SLU determines a value between a lower bound and an upper bound, in the same manner described above for the maximum-size RU sent by the SLU, except that when modifying an upper bound, it takes into account only the PLU's capability to reassemble (rather than to generate) segments.

TS Usage—Staging for Primary TC to Secondary TC: Copied from the BIND.

TS Usage—Primary TC's Send Window Size: If adaptive pacing is supported by both this LU and the partner LU, then this parameter is set to 0. The initial value of Primary Send Window Size will be 1 when session traffic first starts flowing. The value of this parameter may change while the session is active.

If adaptive pacing is not supported by both ends of the session, then this value is taken from the BIND, as follows: If the BIND specifies one-stage pacing from the PLU to the SLU, this specification is taken from the Secondary TC's Receive Window Size field; otherwise, this specification is taken directly from the Primary TC's Send Window Size field.

TS Usage—Primary TC's Receive Window Size: If adaptive pacing is supported by both this LU and the partner LU, then this parameter is set to 0. The initial value of Primary Receive Window Size will be 1 when session traffic first starts flowing. The value of this parameter may change while the session is active.

If adaptive session pacing is not supported by both ends of the session, then this value is taken from the BIND.

PS Profile—PS Usage Format: The SLU specifies basic format.

PS Profile—LU Type: The SLU specifies LU type-6.

PS Usage—LU Type-6 Level: The SLU specifies level 2.

PS Usage—Security Manager Receive Function: The SLU specifies whether it supports a security manager for receiving a user-ID, password or already-verified indication, and profile-ID on FMH-5 Attach commands from the PLU.

PS Usage—Already Verified Indicator Acceptance: The SLU specifies whether it will accept the User-ID Already Verified indication on FMH-5 Attach commands from the PLU.

PS Usage—Synchronization Level: The SLU specifies the synchronization level for the session, as follows:

- If a session between the SLU and PLU is already active for the mode name, the SLU specifies the same level of support as specified for the active session.
- If no sessions between the SLU and PLU are active for the mode name and the BIND specifies Confirm, Sync point, and Backout, the SLU specifies the installation-specified value associated with the mode name for the session.
- If no sessions between the SLU and PLU are active for the mode name and the BIND specifies Confirm, the SLU specifies Confirm.

PS Usage—Responsibility for Session Reinitiation: The SLU specifies the responsibility for reinitiation based on the installation-specified responsibility and on the specification in the BIND. This parameter applies only to sessions for which parallel sessions and change number of sessions (CNOS) are not supported.

The matrix in Figure 4-8 shows how the SLU derives the specification for the RSP(BIND). The row headings of the matrix give the installation-specified responsibility and the column headings give the responsibility specified in the BIND. The cells of the matrix give the responsibility that the SLU specifies in the RSP(BIND).

Row headings indicate installation-specified responsibility.

Column headings indicate responsibility specified in BIND.

- Cells indicate responsibility specified in RSP(BIND).

	Operator	Primary	Secondary	Either
Operator	Operator	Operator	Operator	Operator
Primary	Operator	Primary	Either	Primary
Secondary	Operator	Either	Secondary	Secondary
Either	Operator	Primary	Secondary	Either

Figure 4-8. Reinitiation Responsibility

PS Usage—Parallel-Session Support: The SLU specifies parallel-session support for the session, as follows:

- If a session between the SLU and PLU is already active, the SLU specifies the same support as specified for the active session.
- If no sessions between the SLU and PLU are active and the BIND specifies parallel sessions are supported, the SLU specifies the installation-specified value associated with the PLU.
- If no sessions between the SLU and PLU are active and the BIND specifies parallel sessions are not supported, the SLU specifies parallel sessions are not supported.

PS Usage—Change-Number-Of-Sessions Support: The SLU specifies support for the use of change-number-of-sessions (CNOS) protocols, as follows:

- If a session between the SLU and PLU is already active, the SLU specifies the same support as specified for the active session.
- If no sessions between the SLU and PLU are active and the BIND specifies CNOS is supported, the SLU specifies the installation-specified value associated with the PLU.
- If no sessions between the SLU and PLU are active and the BIND specifies CNOS is not supported, the SLU specifies CNOS is not supported.

Cryptography Options: Taken from the BIND.

Primary LU Name: Always omitted.

User Data: The SLU specifies further parameters for the session by means of the User Data structured subfields. If the SLU receives a BIND containing a subfield it does not recognize, it ignores the subfield and does not send it in the RSP(BIND).

The User Data subfields that the SLU sends in the RSP(BIND) are:

Number	Name
X'00'	Unformatted Data
X'02'	Mode Name
X'03'	Session-Instance ID
X'05'	Network-Qualified SLU Name
X'11'	Random Data
X'12'	Enciphered Data

A T2.1-node implementation that contains a single LU and a single link connection, does not support parallel sessions and CNOS, does not support the synchronization level for Sync point and Backout, and does not support LU-LU verification may omit all User Data subfields.

In general, the SLU may omit one or more subfields; see the descriptions of individual subfields for more information. If it does, the entire subfield, including its length, is omitted.

Details of each subfield follow.

- Subfield X'00'—Unformatted Data: This subfield carries installation-specified data. Support for this subfield is implementation-dependent.

- Subfield X'02'—Mode Name: Taken from the BIND, if present there. Otherwise, a Mode Name subfield consisting of eight space (X'40) characters will be set in the RSP(BIND).

- Subfield X'03'--Session Instance Identifier: If this subfield was omitted from the BIND, it will also be omitted from the RSP(BIND). If it was present in the BIND then SLU sets it in the RSP(BIND) as follows:

  - If the first byte of this subfield in the BIND is set to X'01' the SLU returns the abbreviated session instance identifier (PCID portion of the FQPCID) with the first byte set to X'02'. That will indicate that both PLU and SLU will use the FQPCID control vector to uniquely identify the session.

  - Otherwise (the first byte of this subfield in the RSP(BIND) is set to X'00'), the value is taken from the BIND, except that the SLU changes the value of the first byte, if necessary, to make the session-instance ID unique. The SLU sets the first byte to X'F0' if the PLU's network qualified LU name is greater than its own. Otherwise, it sets the first byte to X'00'.

  Before sending a RSP(BIND), the SLU checks that the negotiated session identifier is different from that of all active sessions in which this SLU participates. If the identifier is not unique, the BIND is rejected.

- Subfield X'05'--Network-Qualified SLU Name: The network-qualified SLU name allows the SLU to confirm its identify to the PLU. The network-qualified SLU network name is installation-specified at both the SLU and PLU.

  All T2.1-node products can receive a BIND with the Network-Qualified PLU Name subfield omitted. If the SLU receives such a BIND, it uses a unique default network-qualified PLU name in order to locally identify the PLU.

  A T2.1-node implementation that contains a single LU and a single link connection, does not support parallel sessions and CNOS, does not support the synchronization level for Sync point and Backout, and does not support LU-LU verification may have no network-qualified SLU name. In this case, the SLU omits the Network-Qualified SLU Name subfield from the RSP(BIND).

- Subfield X'11'—Random Data: This subfield is used when LU-LU verification is active. See "LU-LU Verification Data" on page 4-18 for more information on the function of random data.

- Subfield X'12'—Enciphered Data: When the primary LU receives the RSP(BIND), it compares the received enciphered data with its copy of the enciphered data that it has enciphered using the same random data, its copy of the LU-LU password, and the DES algorithm. If they are identical, the primary LU has verified that the SLU has the correct LU-LU password.

User Request Correlation Field: Copied from the BIND.

Secondary LU Name: Always omitted.

Control Vectors: The following control vectors may be included in the RSP(BIND):

- Fully Qualified PCID (FQPCID) control vector: Copied from the BIND, if present; otherwise, not included in the RSP(BIND).

UNBIND

Flow: From LU to LU (Expedited)

UNBIND requests the partner LU to deactivate the session. The UNBIND indicates definite-response requested. After SM sends an UNBIND, it does not keep any information pertaining to the session.

A description of parameters in the UNBIND follows.

Type: This specifies the type of session deactivation requested. The LU specifies normal deactivation when it is deactivating the session normally, that is, not as a result of an error condition. In this case, the two LUs stop all activity on the session prior to deactivating it. Activity is stopped by exchanging BIS requests. See "Chapter 6.1. Data Flow Control" for a description of the BIS request, and "Chapter 3. LU Resources Manager" for details of its use.

UNBIND(Cleanup) is sent in response to a BIND that was not accepted.

The other types of session deactivation are associated with error conditions.

Sense Data: Identifies the reason for the UNBIND. This field is included in the UNBIND if the UNBIND type is X'FE' (Session Failure).

Control Vectors: The following control vectors may be included in the UNBIND:

- FQPCID control vector: When SM sends an UNBIND, it includes an FQPCID control vector except when it received a BIND or a RSP(BIND) for the session without it. SM is always prepared to receive an UNBIND without an FQPCID control vector if a sender is a back-level LU.

- Extended Sense Data (ESD) control vector: This control vector is included if and only if the FQPCID control vector is included and the UNBIND type is not X'01' (Normal End of Session).

NOTE 1: The general architecture allows for some other UNBIND types not to use the ESD control vector. However, the session manager generates only three different UNBIND types: Normal, Cleanup, and Invalid Session Paramenters. Out of those three, Normal is the only one that does not require an ESD control vector when an FQPCID control vector is included.

NOTE 2: Although SM generates only three UNBIND types, it is able to receive an UNBIND of any defined type. See SNA Formats for a list of all possible UNBIND types.

NOTE 3: RM and SS need to be informed of the reason for the session deactivation. Since only UNBIND type X'FE' carries sense data in the RU, SM sets a default value for the other UNBIND types. The default sense data are assumed, as follows:

UNBIND TYPE	SENSE DATA
X'07' VR_INOP	X'80200001'
X'08' ROUTE_EXT_INOP	X'80200004'
X'09' HIERARCHICAL_RESET	X'80030001'
X'0A' SSCP_GONE	X'08A00001'
X'0B' VR_DEACTIVATED	X'80200003'
X'0C' LU_FAIL_UNRECOV	X'80030003'
X'0E' LU_FAIL_RECOV	X'80030004'
X'0F' CLEANUP	X'08A00002'
X'11' GW_NODE_CLEANUP	X'08A00003'

RSP(UNBIND)

Flow: From LU to LU (Expedited)

The LU receiving the UNBIND tries to send back a ±RSP(UNBIND). If SM can correlate an UNBIND to one of its active or pending-active sessions, it uses a buffer it pre-reserved at session activation for the RSP(UNBIND). If SM cannot correlate an UNBIND to one of its active or pending-active sessions, it asks the BM for a demand buffer to be used for the RSP(UNBIND). If such a buffer cannot be obtained, RSP(UNBIND) is not sent.

The LU sends a -RSP(UNBIND) if the format of the UNBIND is in error. Otherwise, the LU sends back a +RSP(UNBIND), even if it has no HS to which it can correlate the UNBIND. A -RSP(UNBIND) includes sense data indicating the format error.

SM AND BUFFER MANAGEMENT

When SM gets a request to activate a session (from either RM or the partner LU), it asks the buffer manager (BM) to reserve the buffers needed for the session.

- When SM is initialized, it asks the BM to reserve a permanent buffer pool. This buffer pool is shared by all HS processes created by SM. HS uses the buffers in the pool to send responses, expedited requests, and IPM acknowledgments.

- When SM creates a new HS (part of session activation), it requests BM to increase the number of buffers in the pool; thus as the number of half-sessions using the buffer pool increases, the number of buffers available increases. When SM destroys a HS, it requests BM to decrease the number of buffers in the pool.

- When SM sends a BIND, it asks the BM for a demand buffer, which is used to build the BIND. When SM receives a BIND, it always gets it in a permanent buffer that can be reused to send the UNBIND or -RSP(BIND). When SM receives a RSP(BIND), an UNBIND, or a RSP(UNBIND), it gets it in a demand buffer, which is always freed and not reused by SM.

- After session parameters are negotiated, SM reserves dynamic and limited buffer pools that will be used by HS for the session. The dynamic buffer pool is used for receive pacing buffers, while the limited buffer pool is used to send normal-flow requests.

- At session activation, SM reserves a buffer that will be used when an UNBIND or a RSP(UNBIND) has to be sent. By pre-reserving this buffer, SM can always issue an UNBIND or RSP(UNBIND), even if BM runs out of other buffer space.

- At session deactivation time SM requests BM to decrease the number of buffers in the permanent buffer pool.

- When SM destroys a HS, the dynamic and limited buffer pools that were established at session activation time are automatically destroyed.

If any of the required buffers cannot be reserved, SM does not proceed with session activation and sends either an ACTIVATE_SESSION_RSP(negative) record to RM, or an UNBIND or -RSP(BIND) with type Cleanup to the partner LU, depending upon who requested the session. For more details on BM, see "Appendix B. Buffer Manager".

SM FLOWS

This section shows sequence flows that can occur between SM and other components of the LU, the node's control point, the node's buffer manager, and other LUs. These flows illustrate some examples of session initiation and termination. The flows illustrate the records listed in "SM Protocol Boundaries" on page 4-4.

See "Component Interactions and Sequence Flows" on page 2-48 for a description of the conventions used in the flows.

```
                 LU
       ┌─────────────────────────┐
       RM         SM        HS            BM            NOF
       ─────────────────────────────────────────────────────────────────────────────────────
       .          .       (create)        .             .             .             .            .
1      .          o<────────────────────────────────────o             .             .            .
       . (create) │           .           .             .             .             .            .
2      o<─────────┤           .           .             .             .             .            .
       .          │           .     RM_CREATED          .             .             .            .
3      .          └────────────────────────────────────>o             .             .            .
       .          │ RESERVE_BUFFER_POOL(permanent)      .             .             .            .
4      .          └──────────────────────>o             .             .             .            .
       .            +RSP(buffer pool ID)  │             .             .             .            .
       .          o<──────────────────────┘             .             .             .            .
       .          .           .           .             .             .             .            .
       .      (Receive and process signals)
       .          .           .           .             .             .             .            .
```

Figure 4-9. SM Initialization

The following notes correspond to the numbers in Figure 4-9.

1. NOF creates SM during node initialization.

2. SM creates RM.

3. SM sends RM_CREATED record to inform NOF that the RM has been created.

4. SM reserves a permanent buffer pool for its own use and to be shared by all HSs that it creates; see "SM and Buffer Management" on page 4-28 for details.

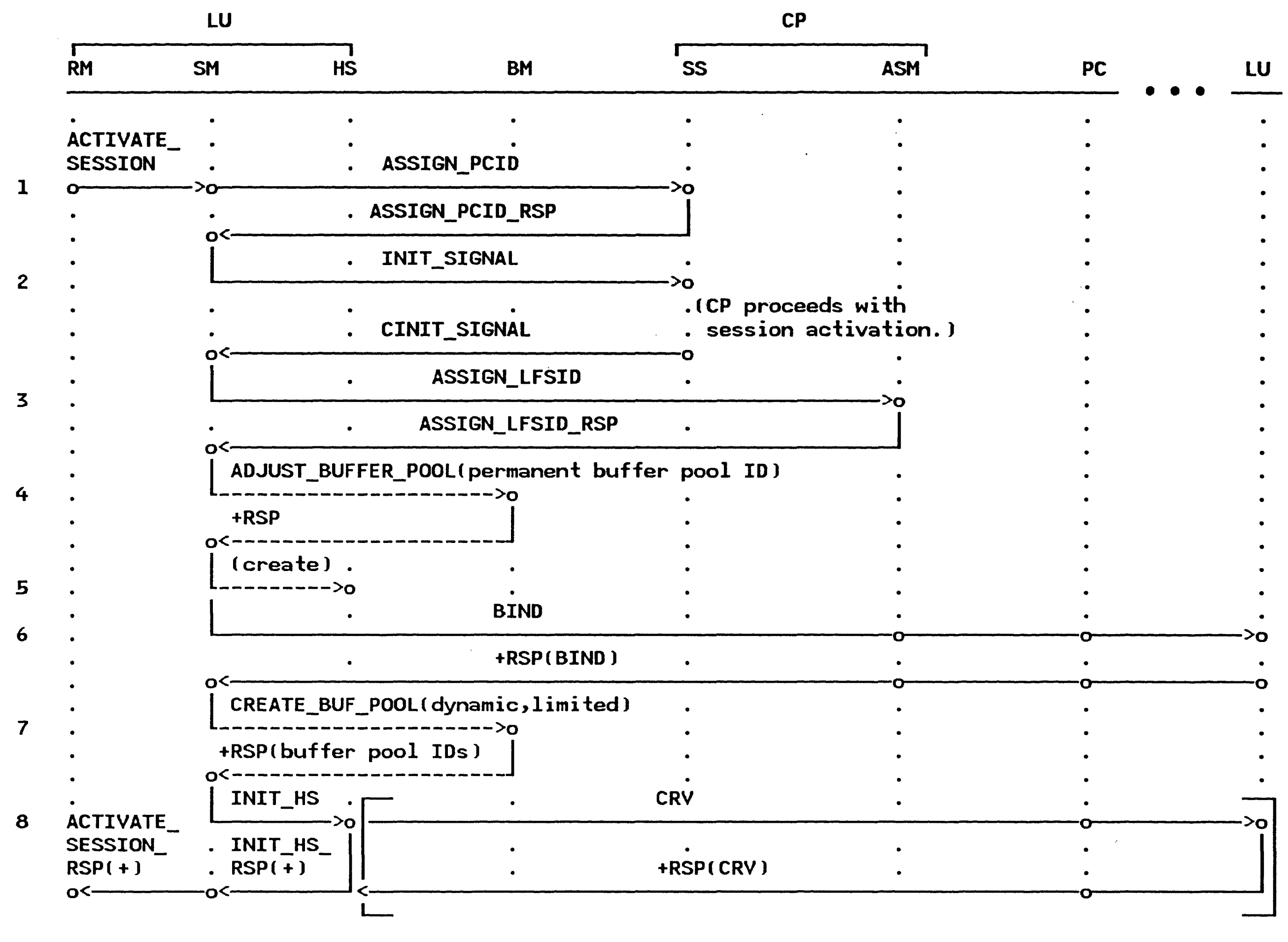


Figure 4-10. Session Initiation by Local LU

The following notes correspond to the numbers in Figure 4-10.

1. RM sends ACTIVATE_SESSION to SM requesting a session be activated with the specified partner LU; RM expects a response. SM requests SS to create an FQPCID, which will uniquely identify the session. SM will verify that this PCID does not collide with a PCID for any of its active or pending-active sessions. (See Figure 4-11 on page 4-33 for collision handling logic.)

2. SM sends an INIT_SIGNAL record to SS requesting the CP's assistance in activating the session. SS, if successful, returns a CINIT_SIGNAL, which contains a path control ID, the maximum BTU size, segmenting capabilities, and (if mode name to COS/TPF mapping is supported) a COS/TPF control vector. SS, if unsuccessful, returns an INIT_SIGNAL_NEG_RSP (not shown) indicating a failure (see SNA Type 2.1 Node Reference for details).

3. SM sends an ASSIGN_LFSID to ASM requesting an LFSID for the session. ASM returns an ASSIGN_LFSID_RSP record, which contains the requested LFSID.

4. SM increases the number of buffers in the permanent buffer pool. All HSs created by this SM share this buffer pool. HS uses these buffers for sending responses, expedited requests, and IPM acknowledgments.

5. SM creates an HS for the session.

6. SM sends the BIND to ASM, which ASM will send to the partner LU; a RSP(BIND) is returned to SM from ASM.

7. SM obtains a dynamic buffer pool, and a limited buffer pool for HS when it knows the maximum RU sizes from the RSP(BIND). HS uses the dynamic buffers for receive pacing buffers, while the limited buffers are used to send normal-flow requests.

8. SM sends an INIT_HS record to the HS it created. This record gives HS all the session information it needs to begin to

perform its functions. This information includes the values of the negotiated session parameters, and buffer pool IDs (returned by buffer manager on the CREATE_BUF_POOL calls). If cryptography is used, the LUs exchange CRV and its response. The HS sends an INIT_HS_RSP to SM to indicate successful initialization. SM then sends an ACTIVATE_SESSION_RSP in response to the original ACTIVATE_SESSION record. This record indicates whether SM was able to satisfy RM's request.

```
             LU                                           CP
   ┌─────────────────────────┐                ┌───────────────────────┐
   RM         SM           HS         BM        SS                ASM
   ──────────────────────────────────────────────────────────────────────────
   .          .            .          .         .                 .
   ACTIVATE_  .            .          .         .                 .
   SESSION    .            .  ASSIGN_PCID       .                 .
1  o─────────>o───────────────────────────────>o                 .
   .          .            . ASSIGN_PCID_RSP    │                 .
   .          o<────────────────────────────────┘                 .
   .      (PCID collision detected)   .         .                 .
   .          │            .  ASSIGN_PCID       .                 .
   .          └────────────────────────────────>o                 .
   .          .            . ASSIGN_PCID_RSP    │                 .
   .          o<────────────────────────────────┘                 .
   .          │            .   INIT_SIGNAL      .                 .
   .          └────────────────────────────────>o                 .
   .          .            .          .         .                 .
2  .          .            .          . (Session initiation proceeds as in
   .          .            .          .   Figure 4-10 on page 4-32.)
```

Figure 4-11. Session Initiation by Local LU: PCID Collision Detected

The following notes correspond to the numbers in Figure 4-11.

1. RM sends an ACTIVATE_SESSION record to SM. SM sends an ASSIGN_PCID to SS requesting a unique FQPCID be assigned for the session. SS sends an ASSIGN_PCID_RSP to SM, which contains the FQPCID that was assigned. SM compares the FQPCID with the FQPCIDs of all its active and pending-active sessions. In this case, a collision is found. SM sends another ASSIGN_PCID to SS, indicating that an FQPCID collision was found. This continues until SS returns an ASSIGN_PCID_RSP with an FQPCID that is unique.

2. Session initiation continues normally.

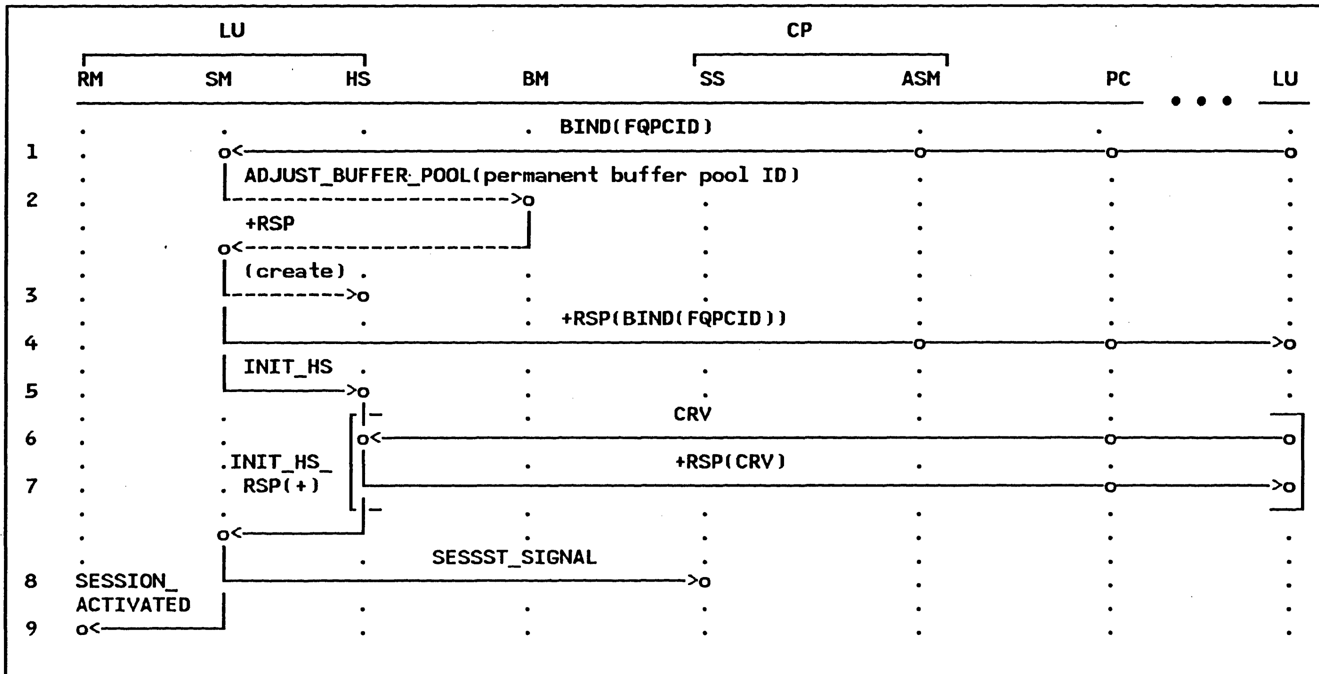


Figure 4-12. Session Activation by Partner LU: BIND(FQPCID) Is Received

The following notes correspond to the numbers in Figure 4-12.

1. SM receives BIND via ASM; the BIND includes an FQPCID control vector.

2. SM adjusts (increases) the number of buffers in the permanent buffer pool. The buffers in this pool are shared by all HSs created by SM.

3. SM creates the HS for the session.

4. SM send the RSP(BIND) to ASM. Since the BIND included the FQPCID control vector, the FQPCID will also be included in the RSP(BIND). The FQPCID is used by the partner LU to correlate the BIND and RSP(BIND).

5. SM sends an INIT_HS record to the HS it created. This record gives HS all the session information it needs to begin to perform its functions. This information includes the values of the negotiated session parameters, and buffer pool IDs of the buffers obtained from BM.

6. If cryptography is used, the LUs exchange CRV and its response.

7. SM receives a INIT_HS_RSP from HS indicating that HS was successfully initialized.

8. SM sends a SESSST_SIGNAL to notify SS that the session has been successfully activated at the request of the partner LU.

9. SM sends SESSION_ACTIVATED to notify RM that the session has been successfully activated at the request of the partner LU.

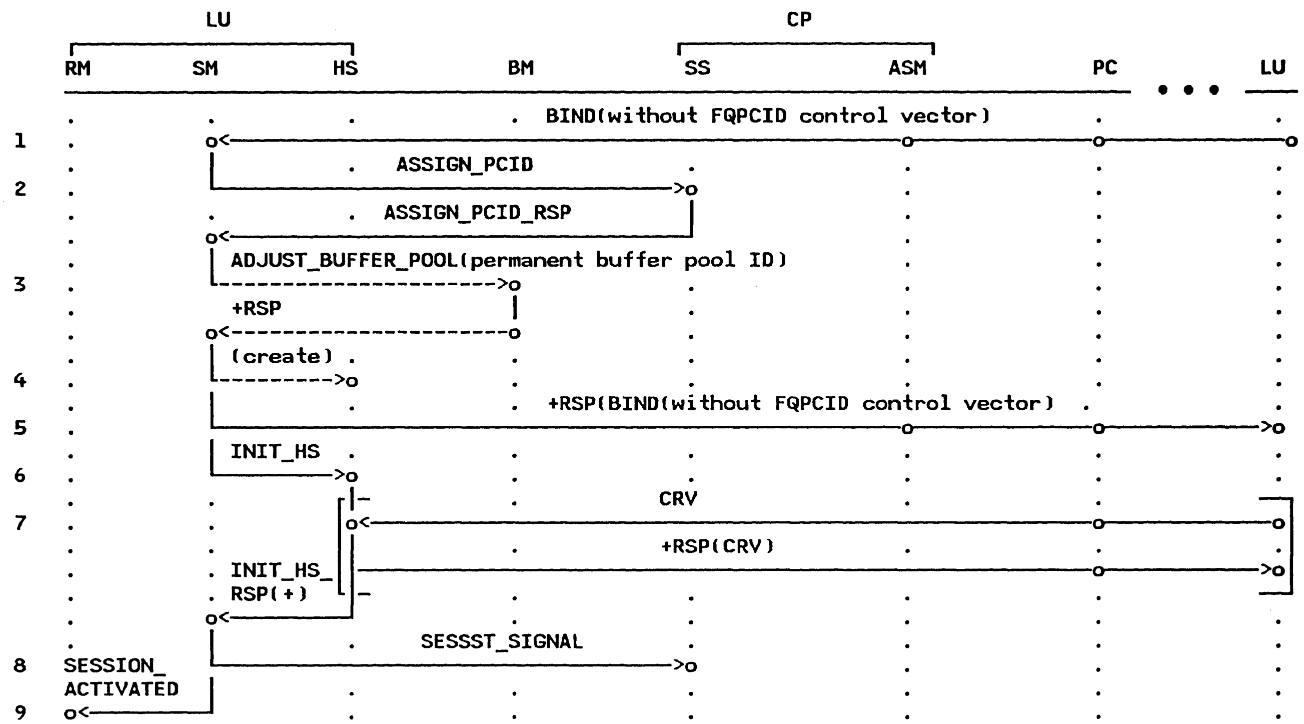


Figure 4-13. Session Activation by Partner LU: BIND Is Received

The following notes correspond to the numbers in Figure 4-13.

1. SM receives BIND from a back-level partner LU; the BIND does not include an FQPCID control vector.

2. Since no FQPCID was in the BIND, SM sends ASSIGN_PCID to SS requesting an FQPCID to be assigned to this session. SS returns the FQPCID in the ASSIGN_PCID_RSP record.

3. SM adjusts (increases) the number of buffers in the permanent buffer pool that is shared by all HSs created by SM.

4. SM creates the HS for the session.

5. SM sends the RSP(BIND) to the partner LU. Since the BIND did not contain an FQPCID, the FQPCID is not appended to the RSP(BIND).

6. SM sends an INIT_HS record to the HS it created. This record gives HS all the session information it needs to begin to perform its functions. This information includes the values of the negotiated session parameters, and buffer pool IDs of the buffers obtained from BM.

7. If cryptography is used, the LUs exchange CRV and its response.

8. SM sends a SESSST_SIGNAL to notify SS that the session has been successfully activated at the request of the partner LU.

9. SM sends SESSION_ACTIVATED to notify RM that the session has been successfully activated at the request of the partner LU.

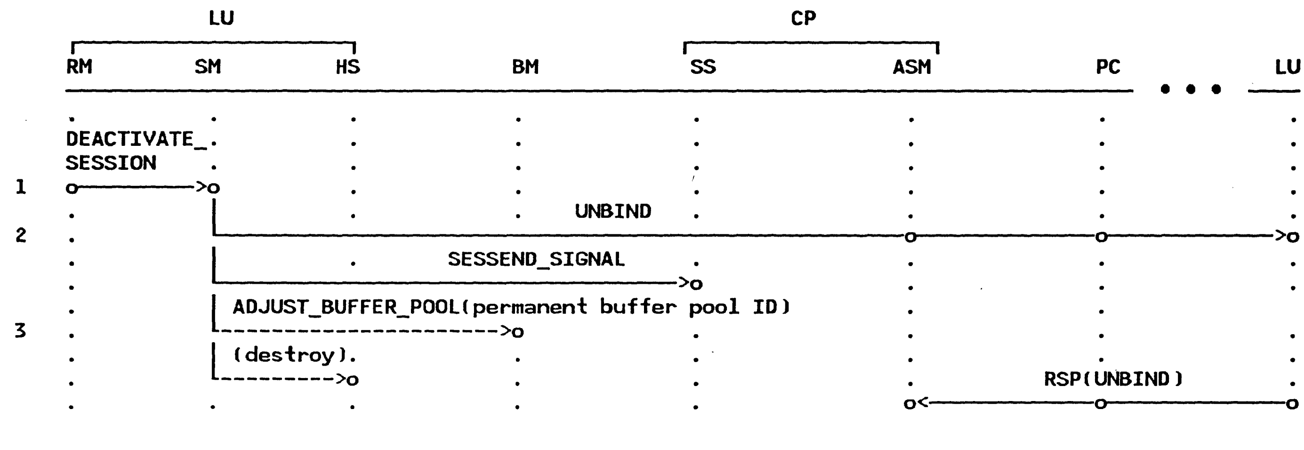


Figure 4-14. Session Deactivation by Local LU

The following notes correspond to the numbers in Figure 4-14.

1. RM sends DEACTIVATE_SESSION record to SM requesting that the session be deactivated.

2. SM sends UNBIND to the partner LU. At this point, SM deactivates the session; it does not wait for the RSP(UNBIND). SM sends a SESSEND_SIGNAL record to notify SS that the session is no longer active.

3. SM adjusts (decreases) the number of buffers in the permanent buffer pool; the dynamic and limited buffer pools are automatically destroyed when the HS is destroyed

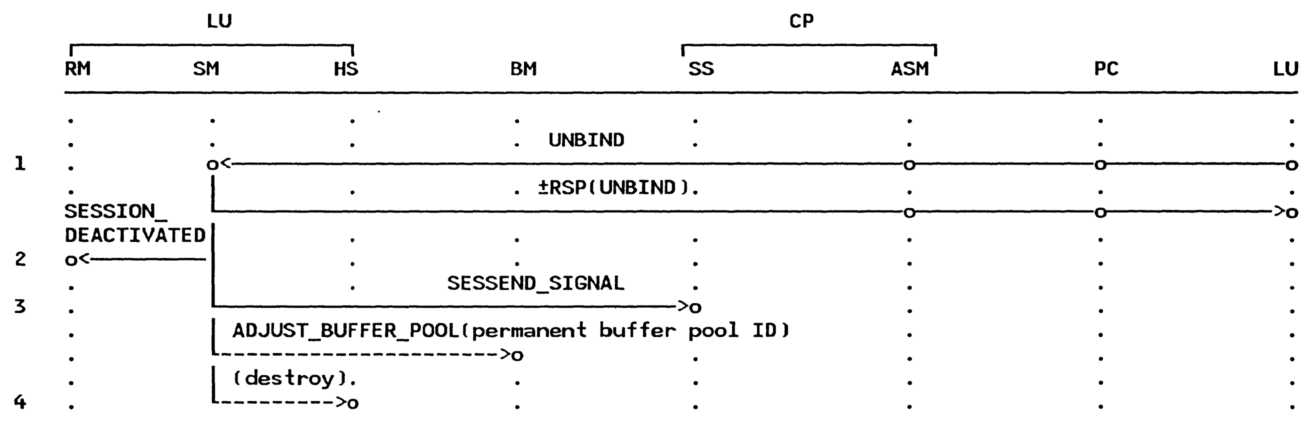


Figure 4-15. Session Deactivation by Partner LU

The following notes correspond to the numbers in Figure 4-15.

1. SM receives UNBIND from the partner LU and returns a RSP(UNBIND).

2. SM sends SESSION_DEACTIVATED to notify RM that the session is deactivated.

3. SM sends SESSEND_SIGNAL to notify SS that the session has ended.

4. SM adjusts (decreases) the number of buffers in the permanent buffer pool; the dynamic and limited buffer pools are automatically destroyed when the HS is destroyed.

```
               LU                                            CP
     ┌──────────────────────────┐                  ┌─────────────────────────┐
     RM          SM            HS              BM   SS                  ASM                PC
     ─────────────────────────────────────────────────────────────────────────────────────────────────────
     .           .             .               .    (session traffic)     .                  .
     .           .             o<────────────────────────────────────────────────────────────o──────────────────>
     .           .             .          SESSION_ROUTE_INOP   .          .
1    SESSION_    o<────────────────────────────────────────────────────────o                  .
     DEACTIVATED│              .               .               .          .                  .                 .
2    o<─────────┘              .               .               .          .                  .                 .
     .           │             .         SESSEND_SIGNAL        .          .                  .                 .
3    .           └─────────────────────────────────────────────>o          .                  .                 .
     .           │ ADJUST_BUFFER_POOL(permanent buffer pool ID)            .                  .                 .
4    .           └----------------------->o                    .          .                  .                 .
     .           │ (destroy).              .                   .          .                  .                 .
     .           └--------->o              .                   .          .                  .                 .
```

Figure 4-16. SM receives SESSION_ROUTE_INOP While a Session Is Active

The following notes correspond to the numbers in Figure 4-16.

1. ASM sends SESSION_ROUTE_INOP to SM indicating that a route is no longer active, and all sessions using this route need to be deactivated.

2. For each of this SM's active sessions that use the affected route, SM sends SESSION_DEACTIVATED to notify RM that the session is deactivated.

3. SM sends SESSEND_SIGNAL to notify SS that the session has ended.

4. SM adjusts (decreases) the number of buffers in the permanent buffer pool; the dynamic and limited buffer pools will be automatically destroyed when the HS is destroyed. SM destroys the HS for the session.

```
               LU                                            CP
     ┌──────────────────────────┐                  ┌─────────────────────────┐
     RM          SM            HS              BM   SS                  ASM                PC         . . .    LU
     ──────────────────────────────────────────────────────────────────────────────────────────────────         ────
     .           .             .               .    .                     .                  .                 .
     ACTIVATE_   .             .               .    .                     .                  .                 .
     SESSION     .             .               .    .                     .                  .                 .
     o──────────>o             .               .    .                     .                  .                 .
     .                         .                    .                     .                  .                 .
     .  (Initiation sequence proceeds as in Figure 4-10 on page 4-32.)
     .           .             .              BIND  .                     .                  .                 .
     .           o─────────────────────────────────────────────────────────o──────────────────o────────────────>o
     .           .             .          SESSION_ROUTE_INOP   .          .                  .                 .
1    ACTIVATE_   o<────────────────────────────────────────────────────────o                  .                 .
     SESSION_    │             .               .               .          .                  .                 .
     RSP(-)      │             .               .               .          .                  .                 .
2    o<──────────o             .               .               .          .                  .                 .
     .           │             . SESSEND_SIGNAL                .          .                  .                 .
     .           └─────────────────────────────────────────────>o          .                  .                 .
     .           │ ADJUST_BUFFER_POOL(permanent buffer pool ID)            .                  .                 .
3    .           └----------------------->o                    .          .                  .                 .
     .           │ (destroy).              .                   .          .                  .                 .
     .           └--------->o              .                   .          .                  .                 .
```

Figure 4-17. SM Receives SESSION_ROUTE_INOP While a Session Is Waiting Activation

The following notes correspond to the numbers in Figure 4-17.

1. After SM has sent out the BIND, and before a RSP(BIND) is returned, ASM sends SM a SESSION_ROUTE_INOP, informing SM that the session cannot be activated.

2. SM returns a negative ACTIVATE_SESSION_RSP informing RM that the requested session could not be activated.

SM then cleans up by sending a SESSEND_SIGNAL to inform SS that the session initiation is being stopped.

3. SM adjusts (decreases) the number of buffers in the permanent buffer pool.

```
             LU                                           CP
     ┌──────────────────────┐                    ┌────────────────┐
     RM        SM        HS         BM           SS            ASM          PC
     ─────────────────────────────────────────────────────────────────────────────────
     .         .         .          .            .              .            .            .
     ACTIVATE_ .         .          .    SESSION_ROUTE_INOP     .            .            .
1    SESSION   .         .          .   ┌───────────────────────o            .            .
2    o────────>o         .          .   │        .              .            .            .
     .         .         .          .   │        .              .            .            .
     .  (Initiation sequence procedes as in  Figure 4-10 on page 4-32.)      .            .
     .         .         .          .   │        .              .            .            .
     .         .         .        BIND  │        .              .            .            .
     .         o────────────────────────┼───────────────────────>o           .            .
     .         .         .          .   │        .              │            .            .
     .         .         .          .   │        .    ┌─────────┘            .            .
3    ACTIVATE_ o<───────────────────────┘        .    │         .            .            .
     SESSION_  │         .          .            .    │         .            .            .
     RSP(-)    │         .          .            .    │         .            .            .
     o<────────┘         .          .            .    │         .            .            .
     .         │         . SESSEND_SIGNAL        .    │         .            .            .
     .         └─────────────────────────────────>o   │         .            .            .
     .         │ ADJUST_BUFFER_POOL(permanent buffer pool ID)   .            .            .
     .         └--------------------->o          .    │         .            .            .
     .         │ (destroy).         .            .    │         .            .            .
     .         └--------->o         .            .    │         .            .            .
     .         .         .          .            .    │         .            .            .
     .         .       SESSION_ROUTE_INOP        .    │         .            .            .
4    .         o<─────────────────────────────────────┘         .            .            .
     .     (ignored)     .          .            .    .         .            .            .
     .         .         .          .            .    .         .            .            .
```

Figure 4-18. SM Receives SESSION_ROUTE_INOP While a Session Is Waiting Activation, Resulting in a BIND Crossing SESSION_ROUTE_INOP

The following notes correspond to the numbers in Figure 4-18.

1. ASM sends a SESSION_ROUTE_INOP to SM; this record is sent to all SMs in the node by ASM whenever a link is brought down.

2. SM receives an ACTIVATE_SESSION record and begins the initiation sequence. A BIND for the session is sent to ASM.

3. SM receives the SESSION_ROUTE_INOP that was sent in step 1. SM brings down all currently active and pending-active sessions that use the affected route. In this case, this includes the session that is currently being activated.

4. SM receives a SESSION_ROUTE_INOP in response to the BIND it sent. The corresponding session is no longer pending-active, since the previous SESSION_ROUTE_INOP forced the session to be deactivated; thus, the SESSION_ROUTE_INOP is ignored.

```
               LU                                            CP
    ┌────────────────────────┐                    ┌──────────────────────┐
    RM          SM          HS          BM        SS                    ASM
    ─────────────────────────────────────────────────────────────────────────
    .           .           .           .      SESSION_ROUTE_INOP       .
 1  ACTIVATE_   .           .           .     ┌─────────────────────────o
    SESSION     .           .  ASSIGN_PCID    │         .               .
 2  o──────────>o─────────────────────────────┼────────>o               .
    .           .           . ASSIGN_PCID_RSP │         │               .
    .           o<────────────────────────────┤─────────┘               .
    .           │           .   INIT_SIGNAL   │         .               .
    .           └─────────────────────────────┼────────>o               .
    .           .           .           .     │         .               .
 3  .           o<────────────────────────────┘         .(CP proceeds with
    .       (ignored)       .           .               . session activation, and
    .           .           .           .               . a route is successfully
    .           .           .  CINIT_SIGNAL             . reactivated.)  .
 4  .           o<──────────────────────────────────────o               .
    .           .           .           .               .               .
    .  (Session activation proceeds as in Figure 4-10 on page 4-32.)
    .           .           .           .               .               .
```

Figure 4-19. SM Receives SESSION_ROUTE_INOP While a Session Is Waiting Activation, Resulting in an INIT_SIGNAL Crossing SESSION_ROUTE_INOP

The following notes correspond to the numbers in Figure 4-19.

1. ASM sends a SESSION_ROUTE_INOP for a currently active session.

2. SM receives an ACTIVATE_SESSION record and begins the session initiation sequence. SM sends an INIT_SIGNAL to SS.

3. After SM sends the INIT_SIGNAL to SS, and before SS returns a CINIT_SIGNAL, SM receives the SESSION_ROUTE_INOP that was sent in step 1. In this case, no sessions are active or pending_active (a session is not pending-active until a BIND has been sent) that use the affected route, so the SESSION_ROUTE_INOP is ignored.

   As SM is processing the SESSION_ROUTE_INOP, the CP proceeds with the session activation. During the activation, the CP is able to reactivate the route that was down.

4. SS sends a CINIT_SIGNAL to SM in response to the preceding INIT_SIGNAL.

```
             LU                                          CP
    ┌──────────────────────┐                   ┌──────────────────────┐
    RM        SM        HS         BM          SS             ASM
    ─────────────────────────────────────────────────────────────────────
    .         .          .          .           .              .
    ACTIVATE_ .          .          .           .              .
    SESSION   .          .  ASSIGN_PCID         .              .
1   o-------->o------------------------------->o              .
    .         .          . ASSIGN_PCID_RSP      |              .
    .         o<--------------------------------┘              .
    .         |          .  INIT_SIGNAL         .              .
    .         └-------------------------------->o              .
    .         .          .          .           .              .
    .         .          .          .           .  (CP proceeds with
    .         .          .          .           .   session initiation.)
    .         .          .          .           .              .
    .         .          .   CINIT_SIGNAL       .              .
    .         o<--------------------------------o              .
    .         |          .    ASSIGN_LFSID      .              .
    .         └------------------------------------------------>o
    .         .      ASSIGN_LFSID_RSP(sense data)              |
2   ACTIVATE_ o<-----------------------------------------------┘
    SESSION_  |          .          .           .              .
    RSP(-)    |          .          .           .              .
3   o<--------┘          .          .           .              .
    .         |          . SESSEND_SIGNAL       .              .
4   .         └-------------------------------->o              .
```

Figure 4-20. Session Activation by Local LU: LFSID Assignment Failed

The following notes correspond to the numbers in Figure 4-20.

1. RM sends SM an ACTIVATE_SESSION. SM sends an ASSIGN_PCID to SS; SS returns an ASSIGN_PCID_RSP to SM that contains the requested PCID. SM then sends an INIT_SIGNAL to SS; the CP determines where the partner LU is and returns a CINIT_SIGNAL or an INIT_SIGNAL_NEG_RSP (not shown) as described for Figure 4-10 on page 4-32. SM then sends an ASSIGN_LFSID to SS requesting an LFSID for the session.

2. SM receives an ASSIGN_LFSID_RSP that contains sense data indicating why an LFSID could not be assigned.

3. SM sends a negative ASSIGN_SESSION_RSP to inform RM that the session could not be activated.

4. SM sends a SESSEND_SIGNAL to inform SS that the session initiation failed.

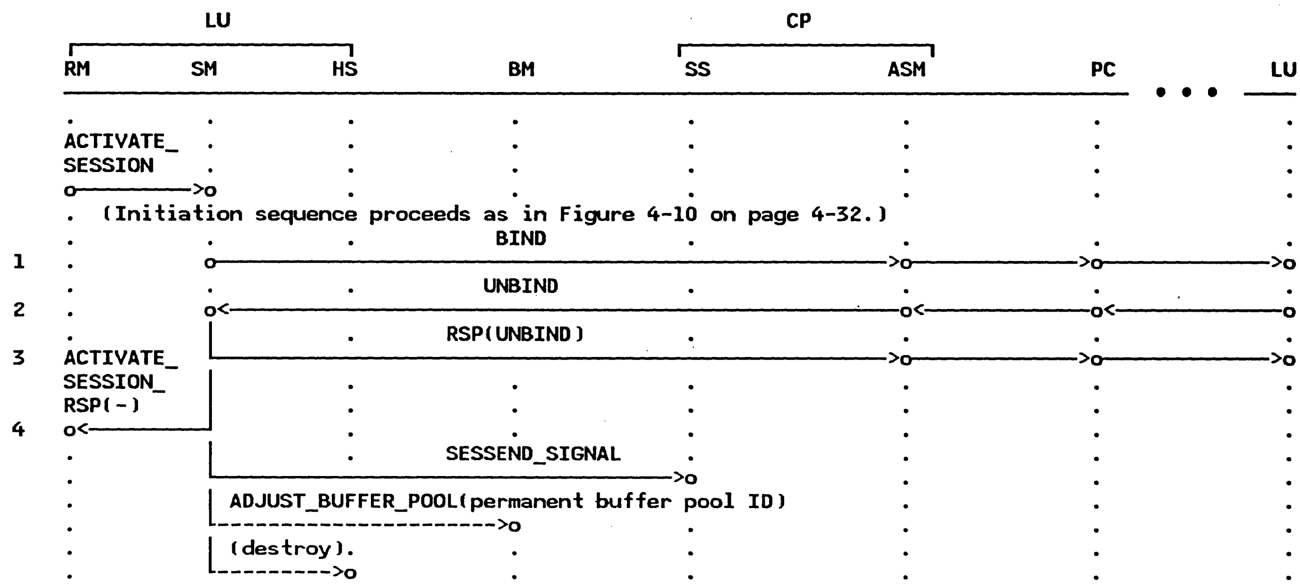


Figure 4-21. Session Activation by Local LU: BIND Is Rejected with UNBIND

The following notes correspond to the numbers in Figure 4-21.

1. SM receives an ACTIVATE_SESSION record and begins the initiation sequence by sending BIND to the partner LU.

2. SM receives an UNBIND from the partner LU rejecting the BIND.

3. SM sends a RSP(UNBIND) to the partner LU.

4. SM sends a negative ACTIVATE_SESSION_RSP to inform RM that the session could not be activated. SM then cleans up the session by sending a SESSEND_SIGNAL to inform SS that the session activation failed, adjusts (decreases) the number of buffers in the permanent buffer pool.

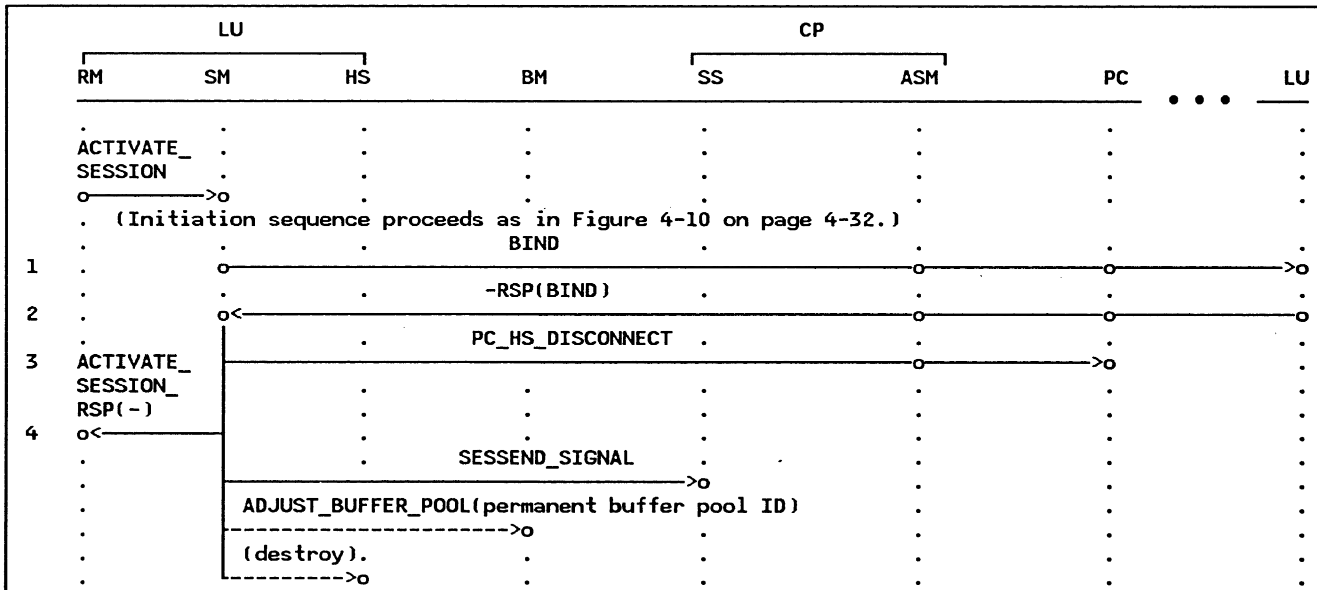


Figure 4-22. Session Activation by Local LU: BIND Is Rejected with -RSP(BIND).

The following notes correspond to the numbers in Figure 4-22.

1. SM receives an ACTIVATE_SESSION record and begins the initiation by sending BIND to the partner LU.

2. SM receives a -RSP(BIND) from the partner LU.

3. SM sends a PC_HS_DISCONNECT instructing ASM to free the LFSID.

4. SM sends a negative ACTIVATE_SESSION_RSP to inform RM that the session could not be activated. SM then cleans up the session by sending a SESSEND_SIGNAL to inform SS that the session activation failed, adjusts (decreases) the number of buffers in the permanent buffer pool. SM destroys the corresponding HS.

```
              LU                                          CP
      ┌──────────────────────┐                  ┌───────────────────┐
      RM        SM        HS           BM        SS                ASM
   ─────────────────────────────────────────────────────────────────────────
      .          .         .            .         .                  .
      ACTIVATE_  .         .            .         .                  .
      SESSION    .         .  ASSIGN_PCID         .                  .
      o─────────>o───────────────────────────────>o                  .
      .          .         . ASSIGN_PCID_RSP      │                  .
      .          o<───────────────────────────────┘                  .
      .          │         .   INIT_SIGNAL        .                  .
 1    .          └───────────────────────────────>o                  .
      .          .         .            .         .(CP cannot proceed
      .          .         .            .         . successfully with
      .          .         .            .         . session activation.)
      .          .         . INIT_SIGNAL_NEG_RSP  .                  .
 2    ACTIVATE_  o<───────────────────────────────o                  .
      SESSION_   │         .            .         .                  .
      RSP(-)     │         .            .         .                  .
 3    o<─────────┘         .            .         .                  .
```

Figure 4-23. Session Initiation by Local LU: INIT_SIGNAL Is Rejected

The following notes correspond to the numbers in Figure 4-23.

1. SM receives an ACTIVATE_SESSION record and begins the initiation by sending an INIT_SIGNAL record to SS.

2. SM receives an INIT_SIGNAL_NEG_RSP from SS informing SM that the CP was unsuccessful with the session activation. (See SNA Type 2.1 Node Reference for details.)

3. SM sends a negative ACTIVATE_SESSION_RSP to inform RM that the session could not be activated.

```
              LU                                          CP
      ┌──────────────────────┐                  ┌───────────────────┐
      RM        SM        HS           BM        SS                ASM
   ─────────────────────────────────────────────────────────────────────────
      .          .         .       LFSID_IN_USE   .                  .
 1    .          o<──────────────────────────────────────────────────o
      .          │         .     LFSID_IN_USE_RSP .                  .
 2    .          └──────────────────────────────────────────────────>o
```

Figure 4-24. ASM Checks Whether a Specific (PATH_CONTROL_ID, LFSID) Pair Is in Use by SM

The following notes correspond to the numbers in Figure 4-24.

1. ASM sends an LFSID_IN_USE record to check if the specified (PATH_CONTROL_ID, LFSID) pair is in use by SM.

2. SM returns an LFSID_IN_USE_RSP to ASM with the status of the pair. (See SNA Type 2.1 Node Reference for additional discussion of this exchange.)

```
           LU                                          CP
      ┌─────────────────────┐                    ┌──────────────┐
      RM        SM         HS          BM        SS            ASM
      ──────────────────────────────────────────────────────────────────────────
      .          .          .           .         .              .
      ACTIVATE_  .          .           .         .              .
      SESSION    .          .  ASSIGN_PCID        .              .
      o─────────>o────────────────────────────────>o              .
      .          .          . ASSIGN_PCID_RSP     |              .
      .          o<────────────────────────────────┘              .
      .          |          .   INIT_SIGNAL       .              .
 1    .          └────────────────────────────────>o              .
      .          .          .           .         .              .
      DEACTIVATE_.          .           .         . CP proceeds with
      SESSION    .          .           .         . session activation.)
 2    o─────────>o          .           .         .              .
      .          .          .  CINIT_SIGNAL       .              .
 3    .          o<────────────────────────────────o              .
      .          |          . SESSEND_SIGNAL      .              .
 4    .          └────────────────────────────────>o              .
```

Figure 4-25. Termination of a Pending LU-LU Session before CINIT_SIGNAL Is Received

The following notes correspond to the numbers in Figure 4-25.

1. SM receives an ACTIVATE_SESSION record and begins the initiation sequence by sending an INIT_SIGNAL to SS.

2. While SM is waiting for the CINIT_SIGNAL from SS in response to the INIT_SIGNAL, RM sends SM a DEACTIVATE_SESSION, which caused SM to change the state of the session to RESET.

3. SM receives the CINIT_SIGNAL for the session RM has since requested to be deactivated; thus, the FQPCID does not match the FQPCID of any active or pending-active session.

4. SM sends a SESSEND_SIGNAL to inform SS that the session activation was terminated.

```
                 LU                                      CP
     ┌───────────────────────────┐              ┌─────────────────────┐
     RM          SM            HS          BM       SS               ASM          PC     • • •     LU
     ────────────────────────────────────────────────────────────────────────────────────     ────
     .            .            .            .        .                .             .                .
     ACTIVATE_    .            .            .        .                .             .                .
     SESSION      .            .            .        .                .             .                .
     o---------->o             .            .        .                .             .                .
     .  (Initiation proceeds as in Figure 4-10 on page 4-32.)
     .            .            .          BIND       .                .             .                .
1    .            o-------------------------------------------------------o-------------o--------------->o
     DEACTIVATE_.              .            .        .                .             .                .
     SESSION      .            .            .        .                .             .                .
2    o---------->o             .            .        .                .             .                .
     .            |            .            .        .                .             .                .
     .            |            .            .        .                .             .                .
     .            |            .            .        .          ┌-----o-------------o----------------o
     .            |            .            .        .          |     .             .                .
     .            |            .          UNBIND     .          |     .             .                .
3    .            └─────────────────────────────────────────────   ───o-------------o--------------->o
     .            |            .    SESSEND_SIGNAL   .          |     .             .                .
4    .            └------------------------------------->o      |     .             .                .
     .            | ADJUST_BUFFER_POOL(permanent buffer pool ID)|     .             .                .
     .            └---------------------->o          .          |     .             .                .
     .            | (destroy).              .        .          |     .             .                .
     .            └--------->o              .        .          |     .             .                .
     .            .            .   ±RSP(BIND) or UNBIND         |     .             .                .
5    .            o<--------------------------------------------┘     .             .                .
     .        (ignored)        .            .        .                .             .                .
```

Figure 4-26. Termination of a Session Pending Activation After BIND Is Sent

The following notes correspond to the numbers in Figure 4-26.

1. SM receives an ACTIVATE_SESSION record and begins the initiation sequence with BIND.

2. While SM is waiting for the RSP(BIND) from ASM, RM sends SM a DEACTIVATE_SESSION, which causes SM to change the state of the session to RESET.

3. SM sends an UNBIND to ASM to stop the session SM is currently in the process of activating.

4. SM sends a SESSEND_SIGNAL to inform SS that the session activation was unsuccessful. SM also adjusts (decreases) the number of buffers in the permanent buffer pool. SM destroys the corresponding HS.

5. SM receives the RSP(BIND) from ASM. Since SM has already issued an UNBIND for the session and has released the FQPCID, SM does not have any active or pending-active session with an FQPCID that matches the FQPCID on the received UNBIND; thus it ignores UNBIND.

```
              LU                                    CP
        ┌─────────────────────┐                ┌─────────────┐
        RM        SM          HS         BM      SS          ASM          PC     • • •    LU
        ─────────────────────────────────────────────────────────────────────────   ──────
        .         .           .          .       .           .            .               .
        .         .           .          .    (session traffic)           .               .
        .         .           o<──────────────────────────────────────────o──────────────>o
        .         .           .          .       .           .            .               .
        .         . ABORT_HS  .          .       .           .            .               .
1       .         o<──────────o          .       .           .            .               .
        .         │           .          .     UNBIND        .            .               .
2       SESSION_  └──────────────────────────────────────────o────────────o──────────────>o
        DEACTIVATED│          .          .       .           .            .               .
3       o<─────────┘          .          .       .           .            .               .
        .         │           .   SESSEND_SIGNAL .           .            .               .
        .         └─────────────────────────────>o           .            .               .
        .         │ ADJUST_BUFFER_POOL(permanent buffer pool ID)          .               .
        .         └──────────────────────>o      .           .            .               .
        .         │ (destroy).           .       .           .     ±RSP(UNBIND)           .
        .         └──────────>o          .       .           o<───────────o──────────────o
        .         .           .          .       .    (not passed to SM)  .               .
        .         .           .          .       .           .            .               .
```

Figure 4-27. SM Receives ABORT_HS While a Session Is Active

The following notes correspond to the numbers in Figure 4-27.

1. During an active session, HS detects an error and sends an ABORT_HS record to SM.

2. SM sends an UNBIND to ASM. At this point, SM brings down the session, without waiting for the RSP(UNBIND).

3. SM sends a SESSION_DEACTIVATED to inform RM that the session is being deactivated. SM sends a SESSEND_SIGNAL to inform SS that the session is being deactivated. SM adjusts (decreases) the number of buffers in the permanent buffer pool. The dynamic and limited buffer pools will automatically be destroyed when the HS is destroyed; SM destroys the corresponding HS.

```
              LU                                          CP
       ┌───────────────────┐                    ┌─────────────────────┐
       RM        SM        HS         BM        SS                    ASM
       ──────────────────────────────────────────────────────────────────────
       .         .         .          .         .                     .
       ACTIVATE_ .         .          .         .                     .
       SESSION   .         .  ASSIGN_PCID       .                     .
1      o────────>o──────────────────────────────>o                    .
       .         .         . ASSIGN_PCID_RSP    .                     .
       .         o<──────────────────────────────o                    .
       .         │         .  INIT_SIGNAL       .                     .
       .         └──────────────────────────────>o                    .
       .         .         .          .         .                     .
       .         .         .          .         .(CP proceeds with
       .         .         .          .         . session initiation.)
       .         .         .          .         .                     .
       .         .         .  CINIT_SIGNAL      .                     .
       .         o<──────────────────────────────o                    .
       .         │         .    ASSIGN_LFSID    .                     .
       .         └──────────────────────────────────────────────────>o
       .         .         .   ASSIGN_LFSID_RSP .                     │
       .         o<───────────────────────────────────────────────────┘
       .         │ ADJUST_BUFFER_POOL(permanent buffer pool ID)       .
       .         └---------------------->o      .                     .
       .           -RSP                  │      .                     .
2      .         o<----------------------┘      .                     .
       .         │         .     FREE_LFSID     .                     .
3      ACTIVATE_ └──────────────────────────────────────────────────>o
       SESSION_  │         .          .         .                     .
       RSP(-)    │         .          .         .                     .
4      o<────────┘         .          .         .                     .
       .         │       SESSEND_SIGNAL.        .                     .
5      .         └──────────────────────────────>o                    .
```

Figure 4-28. A Request to Get a Buffer Is Rejected during Session Activation

The following notes correspond to the numbers in Figure 4-28.

1. RM sends an ACTIVATE_SESSION. The session initiation sequence proceeds as in Figure 4-10 on page 4-32.

2. SM receives a negative response to its request to increase the number of buffers in the permanent buffer pool. At this point session initiation cannot continue, so SM brings down the session.

3. SM sends a FREE_LFSID record to ASM. ASM normally would get this information from the UNBIND, but since a BIND was not issued, an UNBIND also is not issued.

4. SM sends a negative ACTIVATE_SESSION_RSP to inform RM that the session activation failed.

5. SM sends a SESSEND_SIGNAL to inform SS that the session initiation failed.

INTRODUCTION TO FORMAL DESCRIPTION

The remainder of this chapter contains the formal description of SM. This description consists of procedural logic, finite-state machines (FSMs), and data structures used only by SM. The highest level is the root procedure of the calling tree, named SM (same as the overall component). The SM root procedure calls the other procedures.

The procedures are arranged in the following order: first is the highest-level routine SM, followed by the routines called by SM--PROCESS_RECORD_FROM_RM, PROCESS_RECORD_FROM_HS, PROCESS_RECORD_FROM_SS, PROCESS_RECORD_FROM_ASM--followed by the remaining procedures in alphabetical order.

SM

FUNCTION:	LU session manager (SM) is responsible for creating the RM process and for activating and deactivating sessions between this LU and another LU. There is one SM process per LU in the node, and it is created and destroyed when the LU is created and destroyed. SM receives records from the resources manager (RM), the half-session (HS), the address space manager (ASM), and the session services (SS) processes. When the records are received, they are routed to the appropriate procedures where they are processed. SM uses process data (called LOCAL) that can be accessed by any procedure in the SM process.
INPUT:	At SM creation time: SM_CREATE_PARMS (contains information about the node and the LU associated with SM). At run time: records from RM, HS, ASM, or SS.
OUTPUT:	At SM creation time: RM process is created and the node operator facility (NOF) is informed of that, if successful. Otherwise, the SM process ends abnormally. A pool of buffers needed by SM is reserved. If buffers are not available, SM ends abnormally. At run time: received records are routed to appropriate procedures in SM. LOCAL.SENSE_CODE is initialized.

Referenced procedures, FSMs, and data structures:

RM	page 3-19
PROCESS_RECORD_FROM_RM	page 4-49
PROCESS_RECORD_FROM_HS	page 4-50
PROCESS_RECORD_FROM_ASM	page 4-51
PROCESS_RECORD_FROM_SS	page 4-50
SM_CREATE_PARMS	page A-27
RM_CREATE_PARMS	page A-27
RM_CREATED	page A-27
LOCAL	page 4-89
LUCB	page A-1
PARTNER_LU	page A-2
MODE	page A-3
LULU_CB	page 4-90

Creation-Time Logic

```
Set up addressability to the control blocks used by SM.  The SM
 process data (LOCAL) is a data area that may be referenced by any procedure
 or FSM in SM.  LOCAL is referenced only within SM.
 The LU control block (LUCB), partner-LU control block (PARTNER_LU in
 LUCB.PARTNER_LU_LIST), and mode control block (MODE in PARTNER_LU.MODE_LIST)
 are used but not created by SM.
 The LU-LU control block (LULU_CB in LOCAL.LULU_CB_LIST) is
 created and used only by SM.

Create RM_CREATE_PARMS record.
Set RM_CREATE_PARMS.LUCB_LIST_PTR to SM_CREATE_PARMS.LUCB_LIST_PTR.
Set RM_CREATE_PARMS.LU_ID to SM_CREATE_PARMS.LU_ID.
Create RM process while passing RM_CREATE_PARMS to it.
If RM process is created successfully then
   Create RM_CREATED record.
   Set RM_CREATED.LU_ID to SM_CREATE_PARMS.LU_ID.
   Send RM_CREATED to the NOF component of the node.
Else     (an attempt to create an RM process failed)
   Abend.
```

```
Call buffer manager(CREATE_BUF_POOL, permanent, buffer size, capacity
 of pool) to obtain a permanent buffer pool.  Buffer pool size is set
 to the maximum RU size, capacity of pool is set to 5 (number of
 buffers put in the pool).  Permanent buffer pool ID is
 returned by buffer manager.  This buffer pool will be shared by all
 HSs created by this SM, and is used for storing normal flow responses,
 IPM acknowledgments, and expedited flows.  The number of buffers
 is increased each time a new HS is created, and decreased each time
 a HS is destroyed (Appendix B).
If buffers are not available then
   Abend.
```

Run-Time Logic

```
Do until SM process is destroyed.
   Set LOCAL.SENSE_CODE to X'00000000' to indicate that there is no error
   when an incoming record is received.
   Select based on one of the following conditions:
      When record is received from RM
         Call PROCESS_RECORD_FROM_RM(RM_TO_SM_RECORD) (page 4-49).
      When record is received from HS
         Call PROCESS_RECORD_FROM_HS(HS_TO_SM_RECORD) (page 4-50).
      When record is received from ASM
         Call PROCESS_RECORD_FROM_ASM(ASM_TO_SM_RECORD) (page 4-51).
      When record is received from SS
         Call PROCESS_RECORD_FROM_SS(SS_TO_SM_RECORD) (page 4-50).
```

PROCESS_RECORD_FROM_RM

FUNCTION:	Route records received from RM to appropriate procedures.
INPUT:	Record from RM: ACTIVATE_SESSION, DEACTIVATE_SESSION, or ABEND_NOTIFICATION record
OUTPUT:	Record from RM forwarded to appropriate procedure

Referenced procedures, FSMs, and data structures:

PROCESS_ACTIVATE_SESSION	page 4-75
PROCESS_DEACTIVATE_SESSION	page 4-80
PROCESS_ABEND_NOTIFICATION	page 4-74
ACTIVATE_SESSION	page A-20
DEACTIVATE_SESSION	page A-21
ABEND_NOTIFICATION	page A-25

```
Select based on record from RM:
   When ACTIVATE_SESSION
     Call PROCESS_ACTIVATE_SESSION(ACTIVATE_SESSION)
                                (page 4-75).
   When DEACTIVATE_SESSION
     Call PROCESS_DEACTIVATE_SESSION(DEACTIVATE_SESSION)
                                (page 4-80).
   When ABEND_NOTIFICATION      (RM process has abended)
     Call PROCESS_ABEND_NOTIFICATION(ABEND_NOTIFICATION)
                                (page 4-74).
```

PROCESS_RECORD_FROM_HS

FUNCTION:	Route records received from the half-session (HS) process to the appropriate procedures.
INPUT:	Record from HS: INIT_HS_RSP record, ABORT_HS record, or ABEND_NOTIFICATION record
OUTPUT:	Record from HS forwarded to appropriate procedure

Referenced procedures, FSMs, and data structures:

PROCESS_INIT_HS_RSP	page 4-81
PROCESS_ABORT_HS	page 4-74
PROCESS_ABEND_NOTIFICATION	page 4-74
INIT_HS_RSP	page A-10
ABORT_HS	page A-9
ABEND_NOTIFICATION	page A-25

```
Select based on record from HS:
  When INIT_HS_RSP
    Call PROCESS_INIT_HS_RSP(INIT_HS_RSP)
                         (page 4-81).
  When ABORT_HS
    Call PROCESS_ABORT_HS(ABORT_HS)
                         (page 4-74).
  When ABEND_NOTIFICATION   (HS process has abended)
    Call PROCESS_ABEND_NOTIFICATION(ABEND_NOTIFICATION)
                         (page 4-74).
```

PROCESS_RECORD_FROM_SS

FUNCTION:	Route records received from session services (SS) to the appropriate procedures.
INPUT:	Record from SS: INIT_SIGNAL_NEG_RSP record, or CINIT_SIGNAL record
OUTPUT:	Record from HS forwarded to appropriate procedure

Referenced procedures, FSMs, and data structures:

PROCESS_INIT_SIGNAL_NEG_RSP	page 4-81
PROCESS_CINIT_SIGNAL	page 4-79
INIT_SIGNAL_NEG_RSP	page A-23
CINIT_SIGNAL	page A-23

```
Select based on record from SS:
  When INIT_SIGNAL_NEG_RSP
     Call PROCESS_INIT_SIGNAL_NEG_RSP(INIT_SIGNAL_NEG_RSP)
                                  (page 4-81).
  When CINIT_SIGNAL
     Call PROCESS_CINIT_SIGNAL(CINIT_SIGNAL)
                                  (page 4-79).
```

PROCESS_RECORD_FROM_ASM

FUNCTION:	Route records received from the address space manager (ASM) to the appropriate procedures.
INPUT:	MU, SESSION_ROUTE_INO, or LFSID_IN_USE
OUTPUT:	Record passed to appropriate procedure

Referenced procedures, FSMs, and data structures:


```
Select based on record from ASM:
   When MU
      Call PROCESS_MU(MU)
                                   (page 4-82).
   When SESSION_ROUTE_INOP
      Call PROCESS_SESSION_ROUTE_INOP(SESSION_ROUTE_INOP)
                                   (page 4-83).
   When LFSID_IN_USE
      Call PROCESS_LFSID_IN_USE(LFSID_IN_USE)
                                   (page 4-82).
```

BIND_RQ_STATE_ERROR

FUNCTION:	Determine if there is a state error on receipt of a BIND.
INPUT:	MU record containing BIND
OUTPUT:	TRUE if error detected; otherwise, FALSE. If TRUE, LOCAL.SENSE_CODE is set to the appropriate sense data value.

Referenced procedures, FSMs, and data structures:

BIND_SESSION_LIMIT_EXCEEDED	page 4-57
LOCAL	page 4-89
MU	page A-29
PARTNER_LU	page A-2
MODE	page A-3
BIND RU	SNA Formats
LUCB	page A-1

```
Locate the PARTNER_LU control block using the User Data PLU Name field in BIND.
If PARTNER_LU control block cannot be located then
   Set LOCAL.SENSE_CODE to X'0835xxxx' (xxxx is offset to PLU Name field).
   Return with a value of TRUE (error).

Check for FQPCID collisions.  The FQPCIDs of two sessions at this LU
 collide if their PCID parts are the same.  If such collision is found then
   Set LOCAL.SENSE_CODE to X'083B0001'.
   Return with a value of TRUE (error).

If the levels of session security between LUs do not match then
   Set LOCAL.SENSE_CODE to X'080F6051'  (Security violation).
   Return with a value of TRUE (error).

Locate the MODE control block using the User Data Mode Name field in BIND.
If MODE control block cannot be located then
   Set LOCAL.SENSE_CODE to X'0835xxxx' (xxxx is offset to Mode Name field).
   Return with a value of TRUE (error).

The following determines the session type for this LU so that the check for
 whether the session limit will be exceeded may be made.
If parallel sessions are not supported with the partner LU and
 MODE.MIN_CONWINNERS_LIMIT = 1 then
   Set local session_type to FIRST_SPEAKER.
Else (use value in BIND request)
   If BIND specifies the secondary as contention winner then
      Set local session_type to FIRST_SPEAKER.
   Else
      Set local session_type to BIDDER.

Call BIND_SESSION_LIMIT_EXCEEDED(PARTNER_LU.FULLY_QUALIFIED_LU_NAME, MODE,
 local session_type) (page 4-57).
If the session limit will be exceeded then
   Return with a value of TRUE (LOCAL.SENSE_CODE is set by BIND_SESSION_LIMIT_EXCEEDED).

If partner-LU does not support parallel sessions, and there is another
 session pending with the partner-LU a BIND race condition exists.
 BIND winner is determined by comparing LU names, the LU that sent the
 BIND containing the greater of the two network-qualified LU names is
 the BIND winner.

If BIND specifies that an alternate code set is to be used and
 the alternate code ID is other than ASCII_* then
   Set LOCAL.SENSE_CODE to X'0835007'.
   Return with a value of TRUE (error).

Do consistency checks (on PS usage fields) for parallel sessions using either the
 same partner-LU or the same (partner-LU, mode name) pair (see BIND request in SNA Formats).
If there is a consistency error then
   Set LOCAL.SENSE_CODE to X'0835xxxx' (xxxx is offset to inconsistent field).
   Return with a value of TRUE (error).
```

```
Do consistency checks on conversation-level security indicators
 for parallel sessions using the same PARTNER_LU.
If there is a consistency error then
   Set LOCAL.SENSE_CODE to X'080F6051' (Security violation).
   Return with a value of TRUE (error).

If this LU's cryptography support capability does not match that specified in BIND then
   Set LOCAL.SENSE_CODE to X'0835001A'.
   Return with a value of TRUE (error).

If cryptography is supported with the partner LU, but the cryptography
 component (that enciphers and deciphers) is not active then
   Set LOCAL.SENSE_CODE to X'08480000' (cryptography function inoperative).
   Return with a value of TRUE (error).

Check the SESSION_ID parameter:
If a PARTNER_LU indicates that an FQPCID control vector will be used for
 session identification instead of session instance ID User Data subfield
 (by setting the first byte to X'01' in the User Data SESSION_ID subfield) then
   Use a short (X'02') user data session ID subfield in the RSP(BIND).
   (This procedure checks that in this case the FQPCID control vector is indeed
    present in the BIND.)

Else   (the first byte of SESSION_ID is not equal to X'01'),
   Negotiate the SESSION_ID value:
   If the BIND sender's name is greater than the BIND receiver's name then
      Set the first byte of the SESSION_ID to X'F0'.
   Else (the BIND sender's name is not greater than that of the BIND receiver)
      Set the first byte of the SESSION_ID to X'00'.

In order to check the uniqueness of SESSION_ID,
 the negotiated SESSION_ID is compared to SESSION_IDs
 for all sessions where the BIND exchange has already occurred.
If the negotiated SESSION_ID is not unique then
   Set LOCAL.SENSE_CODE to X'08520001'.
   Return with a value of TRUE (error).

If the PLU does not support receiving of segments and the lower bound
 RU size for the specified mode name > the maximum send BTU size THEN
   Return with a value of TRUE (error).
   Set LOCAL.SENSE_CODE to '08350006'.

If this node does not support segment generation and the lower bound
 RU size for the specified mode name > the maximum send BTU size THEN
   Return with a value of TRUE (error).
   Set LOCAL.SENSE_CODE to '0877002A'.

If this node does not support segment reassembly and the lower bound
 RU size for the specified mode name > the maximum receive BTU size THEN
   Return with a value of TRUE (error).
   Set LOCAL.SENSE_CODE to '0877002B'.

Return with a value of FALSE (no error).
```

BIND_RSP_STATE_ERROR

FUNCTION:	Perform state error checking on a received +RSP(BIND).
INPUT:	MU containing a +RSP(BIND), LULU_CB control block
OUTPUT:	TRUE if error; otherwise, FALSE. If an error is found, LOCAL.SENSE_CODE is set.

Referenced procedures, FSMs, and data structures:

LU_MODE_SESSION_LIMIT_EXCEEDED	page 4-72
MU	page A-29
LULU_CB	page 4-90
LOCAL	page 4-89
PARTNER_LU	page A-2
BIND RU	SNA Formats
MODE	page A-3


```
If the BIND specified that an alternate code set will not be used and
 the RSP(BIND) specifies that an alternate code set may be used then
   Set LOCAL.SENSE_CODE to X'08350006'.
   Return with a value of TRUE (error).

If the RSP(BIND) specifies that an alternate code set may be used and that
 an alternate code process ID is anything but ASCII-8 then
   Set LOCAL.SENSE_CODE to X'08350006'.
   Return with a value of TRUE (error).
```

Pacing and maximum RU size checks

```
If the pacing staging indicators in the RSP(BIND) are not the same as
 those specified in the BIND then
   Set LOCAL.SENSE_CODE to X'08350008', if secondary to primary staging is not the same,
    or to X'0835000C', if primary to secondary staging is not the same.
   Return with a value of TRUE (error).

If the RSP(BIND) indicates that adaptive pacing will not be used on this session then
   If the secondary send window size in the RSP(BIND)
    is not the same as that specified in the BIND then
      Set LOCAL.SENSE_CODE to X'08350008'.
      Return with a value of TRUE (error).

   If the secondary receive window size in the RSP(BIND) is greater than that
    specified in the BIND then   (a window size of 0 is treated as
    infinitely large for these comparisons)
      Set LOCAL.SENSE_CODE to X'08350008'.
      Return with a value of TRUE (error).

   If the primary send window size in the RSP(BIND) is greater than that
    specified in the BIND then   (a window size of 0 is treated as
    infinitely large for these comparisons)
      Set LOCAL.SENSE_CODE to X'0835000C'.
      Return with a value of TRUE (error).

   If the primary receive window size in the RSP(BIND) is not the same as
    that specified in the BIND then
      Set LOCAL.SENSE_CODE to X'0835000D'.
      Return with a value of TRUE (error).

Determine if the secondary or primary send maximum RU sizes are within installation-
 defined bounds.  If path control for the PLU does not support the segment
 reassembly, then secondary send maximum RU size must not exceed the maximum size
 allowed on the link.
If the secondary or primary send maximum RU sizes are not within the installation-
 defined bounds then
   Set LOCAL.SENSE_CODE to X'0835000A', if the secondary send RU size is outside
    the bounds, or to X'0835000B', if that is true for the primary send RU size.
   Return with a value of TRUE (error).
```

PS usage checks

```
If there are other active sessions to this partner-LU and
 the levels of conversation security between sessions to this partner-LU
 do not match then
   Set LOCAL.SENSE_CODE to X'080F6051'.
   Return with a value of TRUE (error).
If there are other active sessions for this (partner-LU, mode name) pair and
 the values of the RSP(BIND) fields for synchronization level and session
 reinitiation do not equal those of the other active sessions then
   Set LOCAL.SENSE_CODE to X'08350018'.
   Return with a value of TRUE (consistency error).

Else (no other sessions active for this [partner-LU, mode name] pair)
   If the RSP(BIND) specifies a synchronization level of Confirm, Sync Point,
    and Backout and the BIND specified only Confirm then
      Set LOCAL.SENSE_CODE to X'08350018'.
      Return with a value of TRUE (error).

If the RSP(BIND) specifies parallel sessions not supported then
   If the RSP(BIND) specifies session reinitiation responsibility as not
    operator controlled and the BIND specified operator controlled then
      Set LOCAL.SENSE_CODE to X'08350018'.
      Return with a value of TRUE (error).

   If the RSP(BIND) specifies session reinitiation responsibility as
    secondary will reinitiate and the BIND specified primary will
    reinitiate then
      Set LOCAL.SENSE_CODE to X'08350018'.
      Return with a value of TRUE (error).

   If the RSP(BIND) specifies session reinitiation responsibility as
    primary will reinitiate and the BIND specified secondary will
    reinitiate then
      Set LOCAL.SENSE_CODE to X'08350018'.
      Return with a value of TRUE (error).

If the values of the RSP(BIND) fields for parallel sessions support and
 change number of sessions support are not the same as specified in the BIND then
   Set LOCAL.SENSE_CODE to X'08350018'.
   Return with a value of TRUE (error).
```

```
Contention winner checks

If the RSP(BIND) specifies parallel sessions supported then
   If the value of the RSP(BIND) contention winner field is not the
    same as that specified in the BIND then
      Set LOCAL.SENSE_CODE to X'08035007'.
      Return with a value of TRUE (error).

Else (parallel sessions not supported)
   If the RSP(BIND) contention winner is specified as the primary and the
    BIND was specified as the secondary then
      Set LOCAL.SENSE_CODE to X'08350007'.
      Return with a value of TRUE (error).

If the RSP(BIND) specifies the primary as the contention winner then
   Set local session_type to FIRST_SPEAKER.
Else
   Set local session_type to BIDDER.

Call LU_MODE_SESSION_LIMIT_EXCEEDED(PARTNER_LU.FULLY_QUALIFIED_LU_NAME,
 MODE, local session_type, active) (page 4-72).
If the session limit will be exceeded then
   Return with a value of TRUE (error). (LOCAL.SENSE_CODE is set by
   LU_MODE_SESSION_LIMIT_EXCEEDED).

Cryptography checks (required).

If the RSP(BIND) cryptography field values are not the same as those specified
 in the BIND then
   Set LOCAL.SENSE_CODE to X'0835xxxx' (xxxx is an offset to that cryptography field
    in the RSP(BIND) that is different from the corresponding value in the BIND).
   Return with a value of TRUE (error).

User data subfield checks

If the user-data mode name in the RSP(BIND) is not the same as that
 specified in the BIND then
   Set LOCAL.SENSE_CODE to X'0835xxxx' (xxxx is an offset to the Mode Name subfield).
   Return with a value of TRUE (error).
If LU-LU verification is active (LULU_CB.RANDOM is nonempty) then
   If enciphered data is absent or incorrect (see page 4-24) then
      Set LOCAL.SENSE_CODE to X'080F6051'.
      Return with a value of TRUE (error).

If the user-data session-instance identifier in the RSP(BIND) is not
 specified correctly or if the negotiated value of the session ID is not unique then
        (see page 4-24 and the SNA Formats).
   Set LOCAL.SENSE_CODE to X'0835xxxx' (xxxx is an offset to the session ID subfield).
   Return with a value of TRUE (error).

URC checks

If the URC in the RSP(BIND) is not the same as that specified
 in the BIND then
   Return with a value of TRUE (error).

Return with a value of FALSE (no error).
```

BIND_SESSION_LIMIT_EXCEEDED

FUNCTION:	Determine whether or not session limits are exceeded for a received BIND.
INPUT:	PARTNER_LU.FULLY_QUALIFIED_LUNAME, MODE, session_type (FIRST_SPEAKER or BIDDER)
OUTPUT:	TRUE if limits exceeded; otherwise, FALSE. If TRUE, LOCAL.SENSE_CODE is set to appropriate sense data value.

Referenced procedures, FSMs, and data structures:

LU_MODE_SESSION_LIMIT_EXCEEDED page 4-72
LOCAL page 4-89
MODE page A-3
PARTNER_LU page A-2


```
If session limit is being negotiated and the proposed
 session limit is > than the current session limit
 (MODE.CNOS_NEGOTIATION_IN_PROGRESS = TRUE)  then
   If active session count is ≥ the proposed limit then
      Set LOCAL.SENSE_CODE to X'08050000'.
      Set local return code to TRUE.

   Else
     If the sum of active session count and pending session
      count is >= the proposed session limit then
        Set LOCAL.SENSE_CODE to X'08050000'.
        Set local check_winner_flag to TRUE.

Else  (session limits not being negotiated)
   Call LU_MODE_SESSION_LIMIT_EXCEEDED(PARTNER_LU.FULLY_QUALIFIED_LU_NAME, MODE,
      inputted session_type, state_condition(ACTIVE))   (page 4-72).
   If the session limit is exceeded then
      Set local return code to TRUE (LOCAL.SENSE_CODE set by LU_MODE_SESSION_LIMIT_EXCEEDED)

   Else
      Call LU_MODE_SESSION_LIMIT_EXCEEDED(PARTNER_LU.FULLY_QUALIFIED_LU_NAME,
       MODE, inputted session_type,
       state_condition(AT_LEAST_BIND_SENT)) (page 4-72).
      If the session limit is exceeded then
         local check_winner_flag is set to TRUE.
```

Check for BIND race condition

```
If local check_winner_flag is true then
   Determine which LU is the BIND race winner.  A comparison is made between the
    SLU name and PLU name (PARTNER_LU.FULLY_QUALIFIED_LU_NAME)
    using the EBCDIC collating sequence.
    The "greater" one is the winner.  Before the comparison is made, the shorter
    name is padded with space (X'40') characters so that the lengths are equal.

   If this LU is the winner then
      Set local return code to TRUE.

   Else
      Reset LOCAL.SENSE_CODE to X'00000000'.
      Set local return code to FALSE.
Return with the value of local return code.
```

BUILD_AND_SEND_ACT_SESS_RSP_NEG

FUNCTION:	Build and send ACTIVATE_SESSION_RSP (negative) to RM.
INPUT:	Correlator (in LULU_CB or ACTIVATE_SESSION) to activate-session request and error type (retry or no retry)
OUTPUT:	ACTIVATE_SESSION_RSP to RM

Referenced procedures, FSMs, and data structures:

RM	page 3-19
ACTIVATE_SESSION_RSP	page A-13
ACTIVATE_SESSION	page A-20
LULU_CB	page 4-90

Create an ACTIVATE_SESSION_RSP record.
Set ACTIVATE_SESSION_RSP.CORRELATOR to passed correlator.
Set ACTIVATE_SESSION_RSP.TYPE to NEG.
Set ACTIVATE_SESSION_RSP.ERROR_TYPE to passed error type.

Send an ACTIVATE_SESSION_RSP record to RM.

BUILD_AND_SEND_ACT_SESS_RSP_POS

FUNCTION:	Build and send ACTIVATE_SESSION_RSP (positive) to RM. This completes (from the SM's standpoint) the session initiation activity triggered by the ACTIVATE_SESSION record received by SM from RM.
INPUT:	LULU_CB control block
OUTPUT:	ACTIVATE_SESSION_RSP created and sent to RM

Referenced procedures, FSMs, and data structures:

RM	page 3-19
LULU_CB	page 4-90
ACTIVATE_SESSION_RSP	page A-13

Create an ACTIVATE_SESSION_RSP record.
Set ACTIVATE_SESSION_RSP.CORRELATOR to LULU_CB.CORRELATOR.
Set ACTIVATE_SESSION_RSP.TYPE to POS (positive response).

Set ACTIVATE_SESSION_RSP.SESSION_INFORMATION.HS_ID to LULU_CB.HS_ID.
Set ACTIVATE_SESSION_RSP.SESSION_INFORMATION.HALF_SESSION_TYPE to PRI.
Set ACTIVATE_SESSION_RSP.SESSION_INFORMATION.BRACKET_TYPE to LULU_CB.SESSION_TYPE.

Set ACTIVATE_SESSION_RSP.SESSION_INFORMATION.SEND_RU_SIZE to the negotiated maximum send RU size.
Set ACTIVATE_SESSION_RSP.SESSION_INFORMATION.PERMANENT_BUFFER_POOL_ID to the ID of the permanent buffer pool.
Set ACTIVATE_SESSION_RSP.SESSION_INFORMATION.LIMITED_BUFFER_POOL_ID to the ID of the limited buffer pool.

Set ACTIVATE_SESSION_RSP.SESSION_INFORMATION.SESSION_IDENTIFIER to LULU_CB.SESSION_ID.

Send random data to RM. This is used to build the FMH-12.

Set ACTIVATE_SESSION_RSP.SESSION_INFORMATION.RANDOM_DATA to LULU_CB.RANDOM.
Send ACTIVATE_SESSION_RSP TO RM.

BUILD_AND_SEND_BIND_RQ

FUNCTION:	Build and send a BIND
INPUT:	LULU_CB control block
OUTPUT:	A BIND request to ASM

Referenced procedures, FSMs, and data structures:

LUCB	page A-1
LULU_CB	page 4-90
MU	page A-29
BIND RU	SNA Formats
ASM	T2.1 Node Reference

```
Call buffer manager(GET_BUFFER, demand, buffer size, no wait)
 to get a demand buffer to contain the BIND.  Buffer size is the
 length of BIND including control vectors plus length of MU overhead.
 (Appendix B).
Set MU.HEADER_TYPE to BIND_RQ_SEND.
Set MU.BIND_RQ_SEND.LU_ID to this LU's identifier.
Set MU.BIND_RQ_SEND.SENDER.TYPE to SM.
Set MU.BIND_RQ_SEND.LFSID to LULU_CB.LFSID.
Set MU.BIND_RQ_SEND.TRANSMISSION_PRIORITY to NETWORK.
Set MU.BIND_RQ_SEND.PATH_CONTROL_ID to LULU_CB.PATH_CONTROL_ID.
Set MU.BIND_RQ_SEND.HS_ID to this half-session's identifier.
Set an MU.TH.SNF field to a unique value.
Set the remaining TH and RH fields in MU
 to the specified values (see page 4-14 and SNA Formats).
Set BIND RU to the appropriate values (see page 4-19).
Insert the random data into the LUCB.PENDING_RANDOM_DATA_LIST.
Set MU.DCF to (RH + RU) length.

Send a BIND MU to ASM.
```

BUILD_AND_SEND_BIND_RSP_NEG

FUNCTION:	Build and send a -RSP(BIND).
INPUT:	Buffer where a -RSP(BIND) will be stored
OUTPUT:	-RSP(BIND) MU to ASM

Referenced procedures, FSMs, and data structures:

MU	page A-29
BIND_RSP_SEND, see MU	page A-29
LOCAL	page 4-89
ASM	T2.1 Node Reference

Set MU.HEADER_TYPE to BIND_RSP_SEND.

Set MU.BIND_RSP_SEND.SENDER.ID to this LU's identifier.
Set MU.BIND_RSP_SEND.SENDER.TYPE to SM.
Set MU.BIND_RSP_SEND.LFSID to the LFSID received in the BIND MU.
Set MU.BIND_RSP_SEND.PATH_CONTROL_ID to the PATH_CONTROL_ID received in the BIND MU.
Set MU.BIND_RSP_SEND.TRANSMISSION_PRIORITY to LOW.
Set MU.BIND_RSP_SEND.FREE_LFSID to YES.
Set MU.BIND_RSP_SEND.HS_ID to NULL.

Set TH and RH fields in the RSP(BIND) MU to appropriate values (see page 4-14 and SNA Formats).
Set the RU portion of the RSP(BIND) MU to LOCAL.SENSE_CODE followed by BIND request code.
Set MU.DCF to (RH + RU) length.

Send -RSP(BIND) MU to ASM.

BUILD_AND_SEND_FREE_LFSID

FUNCTION:	Build and send a FREE_LFSID record to the control point. This is necessary when SM asked ASM to give SM an LFSID for a session, and SM received ASSIGN_LFSID_RSP, but could not send a BIND (because, for example, SM cannot get a buffer for it). In this case, SM explicitly asks ASM to free the LFSID by sending the FREE_LFSID record to it. If SM sends a BIND successfully, it later sends an UNBIND or a RSP(UNBIND) to ASM and sets the FREE_LFSID variable to YES in them.
INPUT:	LULU_CB control block
OUTPUT:	FREE_LFSID record to the ASM component of the control point

Referenced procedures, FSMs, and data structures:

LULU_CB	page 4-90
FREE_LFSID	page A-25
ASM	T2.1 Node Reference

Create a FREE_LFSID record.

Set FREE_LFSID.PATH_CONTROL_ID to LULU_CB.PATH_CONTROL_ID.
Set FREE_LFSID.LFSID to LULU_CB.LFSID.

Send FREE_LFSID record to ASM.

BUILD_AND_SEND_INIT_HS

FUNCTION:	Build an INIT_HS (initialize half-session) record and send it to the half-session designated by the passed LULU_CB.
INPUT:	LULU_CB, first 26 bytes of the negotiated BIND image
OUTPUT:	INIT_HS record to HS (LU-LU half-session)

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
LULU_CB	page 4-90
INIT_HS	page A-13
LOCAL	page 4-89

Create INIT_HS record.
Set the following fields in the INIT_HS record: PATH_CONTROL_ID, LFSID, HALF_SESSION_TYPE, DYNAMIC_POOL_ID, DEM_LIM_POOL_ID, and TRANSMISSION_PRIORITY by copying the corresponding fields from the LULU_CB control block.
Set INIT_HS.SHORT_BIND_IMAGE to the passed 26 bytes of the BIND image.

If adaptive pacing was negotiated for the session then
   Reset all the window size parameters in INIT_HS.SHORT_BIND_IMAGE (SEC_SEND_WINDOW_SIZE, PRI_SEND_WINDOW_SIZE, SEC_RCV_WINDOW_SIZE, and PRI_RCV_WINDOW_SIZE) to 1.

Send INIT_HS record to HS (the LU-LU half-session identified by LULU_CB.HS_ID).
If send fails (because the HS has ABEND) then
   Destroy INIT_HS record.
   Set LOCAL.SENSE_CODE to X'0812000D' (use the same sense code as if there were insufficient buffers to activate a session).

BUILD_AND_SEND_INIT_SIG

FUNCTION:	Build and send an INIT_SIGNAL record to the control point.
INPUT:	LULU_CB control block
OUTPUT:	INIT_SIGNAL record to the SS component of the control point

Referenced procedures, FSMs, and data structures:

LULU_CB	page 4-90
LUCB	page A-1
INIT_SIGNAL	page A-23
SS	<u>T2.1 Node Reference</u>

Create an INIT_SIGNAL record.

Set INIT_SIGNAL.SM_PROCESS_ID to this LU's identifier.
Set INIT_SIGNAL.FQPCID to LULU_CB.FQPCID.
Set INIT_SIGNAL.SLU_NAME to LULU_CB.FQ_PARTNER_LU_NAME.
Set INIT_SIGNAL.PLU_NAME to LUCB.FULLY_QUALIFIED_LU_NAME.
Set INIT_SIGNAL.MODE_NAME to LULU_CB.MODENAME.

Send an INIT_SIGNAL record to SS.

BUILD_AND_SEND_PC_HS_DISCONNECT

FUNCTION:	Build and send a PC_HS_DISCONNECT record to ASM. This is done only after a PLU receives a -RSP(BIND). If, instead, SM receives an UNBIND, it sends a RSP(UNBIND), asking ASM to free LFSID, thus disconnecting PC and HS.
INPUT:	LULU_CB control block
OUTPUT:	PC_HS_DISCONNECT record to ASM

Referenced procedures, FSMs, and data structures:

PC_HS_DISCONNECT	page A-24
LULU_CB	page 4-90
ASM	T2.1 Node Reference

Create a PC_HS_DISCONNECT record.

Set PC_HS_DISCONNECT.PATH_CONTROL_ID to LULU_CB.PATH_CONTROL_ID.
Set PC_HS_DISCONNECT.LFSID to LULU_CB.LFSID.

Send a PC_HS_DISCONNECT record to ASM.

BUILD_AND_SEND_SESS_ACTIVATED

FUNCTION:	Build and send SESSION_ACTIVATED to RM to indicate that a new session has become active and to give RM the information about this session.
INPUT:	LULU_CB control block
OUTPUT:	SESSION_ACTIVATED to RM

Referenced procedures, FSMs, and data structures:

RM page 3-19
LULU_CB page 4-90
SESSION_ACTIVATED page A-14


```
Create a SESSION_ACTIVATED record.

Set SESSION_ACTIVATED.SESSION_INFORMATION.HS_ID to LULU_CB.HS_ID.
Set SESSION_ACTIVATED.SESSION_INFORMATION.HALF_SESSION_TYPE to SEC.
Set SESSION_ACTIVATED.SESSION_INFORMATION.BRACKET_TYPE to LULU_CB.SESSION_TYPE.

Set SESSION_ACTIVATED.SESSION_INFORMATION.SEND_RU_SIZE to the
     negotiated maximum send RU size.
Set SESSION_ACTIVATED.SESSION_INFORMATION.PERMANENT_BUFFER_POOL_ID to
     LULU_CB.PERM_POOL_ID.
Set SESSION_ACTIVATED.SESSION_INFORMATION.LIMITED_BUFFER_POOL_ID to
     LULU_CB.DEM_LIM_POOL_ID (ID of limited buffer pool).

Set SESSION_ACTIVATED.SESSION_INFORMATION.SESSION_IDENTIFIER to
     LULU_CB.SESSION_ID.
```

Send random data to RM to verify the FMH-12's enciphered data received by RM.

```
Set SESSION_ACTIVATED.SESSION_INFORMATION.RANDOM_DATA
     to LULU_CB.RANDOM.

Set SESSION_ACTIVATED.LU_NAME to LULU_CB.LOCAL_PARTNER_LU_NAME.
Set SESSION_ACTIVATED.MODE_NAME to LULU_CB.MODENAME.

Send a SESSION_ACTIVATED record to RM.
If send fails (RM has abended) then
   Destroy SESSION_ACTIVATED record.
```

BUILD_AND_SEND_SESS_DEACTIVATED

FUNCTION:	Build and send SESSION_DEACTIVATED to RM to indicate that an active session has been deactivated.
INPUT:	HS_ID (process identifier of half-session deactivated), REASON (reason for deactivation), SENSE_CODE
OUTPUT:	SESSION_DEACTIVATED record to RM

Referenced procedures, FSMs, and data structures:

RM	page 3-19
SESSION_DEACTIVATED	page A-14

```
Create a SESSION_DEACTIVATED record.

Set SESSION_DEACTIVATED.HS_ID to the value of HS_ID passed to this routine.
Set SESSION_DEACTIVATED.REASON to the value of REASON passed to this routine.
If REASON is not NORMAL then
   Set SESSION_DEACTIVATED.SENSE_CODE to value of SENSE_CODE passed to this routine.

Send a SESSION_DEACTIVATED record to RM.
If send fails (RM has abended) then
   Destroy SESSION_DEACTIVATED record.
```

BUILD_AND_SEND_SESSEND_SIG

FUNCTION:	Build and send a SESSEND_SIGNAL record to the control point. This record can be sent by both PLU and SLU when the session is brought down. The PLU sends it, however, only if it has previously received a CINIT_SIGNAL record. The SLU sends it only if it has already sent a SESSST_SIGNAL record to SS.
INPUT:	LULU_CB, LOCAL.SENSE_CODE
OUTPUT:	SESSEND_SIGNAL record to the SS component of the control point

Referenced procedures, FSMs, and data structures:

LULU_CB	page 4-90
LOCAL	page 4-89
SESSEND_SIGNAL	page A-24
SS	<u>T2.1 Node Reference</u>

```
Create a SESSEND_SIGNAL record.
Set SESSEND_SIGNAL.SENSE_CODE to LOCAL.SENSE_CODE.
Set SESSEND_SIGNAL.FQPCID to LULU_CB.FQPCID.
Set SESSEND_SIGNAL.PATH_CONTROL_ID to LULU_CB.PATH_CONTROL_ID.

Send a SESSEND_SIGNAL record to SS. There is no need to check to see
 if the send failed because if the send did fail SS has abended, and
 the whole node will be down.
```

BUILD_AND_SEND_SESSST_SIG

FUNCTION:	Build and send a SESSST_SIGNAL record to the control point. This record is sent by the SLU when it receives the INIT_HS_RSP record from the half-session process. The PLU does not need to send it, since its local SS sends a CINIT_SIGNAL to SM and assumes that the session will be activated.
INPUT:	LULU_CB
OUTPUT:	SESSST_SIGNAL record to the SS component of the control point

Referenced procedures, FSMs, and data structures:

LULU_CB	page 4-90
SESSST_SIGNAL	page A-24
SS	T2.1 Node Reference

Create a SESSST_SIGNAL record.
Set SESSST_SIGNAL.PATH_CONTROL_ID to LULU_CB.PATH_CONTROL_ID.
Send a SESSST_SIGNAL record to SS.

BUILD_AND_SEND_UNBIND_RQ

FUNCTION:	Build and send an UNBIND.
INPUT:	Buffer where UNBIND will be stored, CLEANUP type, sense code
OUTPUT:	UNBIND MU to ASM

Referenced procedures, FSMs, and data structures:

MU	page A-29
UNBIND_RQ_SEND, see MU	page A-29
LOCAL	page 4-89
LULU_CB	page 4-90
ASM	T2.1 Node Reference

Set MU.HEADER_TYPE to UNBIND_RQ_SEND.

Set MU.UNBIND_RQ_SEND.SENDER.ID to LOCAL.LU_ID.
Set MU.UNBIND_RQ_SEND.SENDER.TYPE to SM.
Set MU.UNBIND_RQ_SEND.LFSID to LULU_CB.LFSID.
Set MU.UNBIND_RQ_SEND.PATH_CONTROL_ID to the LULU_CB.PATH_CONTROL_ID.
Set MU.UNBIND_RQ_SEND.TRANSMISSION_PRIORITY to LULU_CB.TRANSMISSION_PRIORITY.
Set MU.UNBIND_RQ_SEND.FREE_LFSID to YES.
Set MU.UNBIND_RQ_SEND.HS_ID to LULU_CB.HS_ID.

Set TH and RH fields in the UNBIND MU to appropriate values (see page 4-14 and SNA Formats).
Set the RU portion of the UNBIND MU to appropriate values (see page 4-27 and SNA Formats). Use the inputted type and sense code values to set corresponding fields in the UNBIND RU.
Set MU.DCF to (RH + RU) length.

Send UNBIND MU to ASM.

BUILD_AND_SEND_UNBIND_RSP

FUNCTION:	Build and send a RSP(UNBIND).
INPUT:	MU record containing UNBIND
OUTPUT:	A RSP(UNBIND) MU to ASM

Referenced procedures, FSMs, and data structures:

LULU_CB	page 4-90
MU	page A-29
MU_NEW, see MU	page A-29
ASM	T2.1 Node Reference

```
If either UNBIND arrived as an EXR or a length error was detected locally then
   A -RSP(UNBIND) will be built.
Else
   A +RSP(UNBIND) will be built.

Call buffer manager(GET_BUFFER, demand, buffer size, no wait)
 to get a demand buffer to build RSP(UNBIND) (+ or -) in.  Set buffer
 size to the length of RSP(BIND) plus length of MU overhead.
 (Appendix B).

If buffer was gotten successfully then
   Set MU_NEW.HEADER_TYPE to UNBIND_RSP_SEND.
   Set MU_NEW.UNBIND_RSP_SEND.LU_ID to this LU's identifier.
   Set MU_NEW.UNBIND_RSP_SEND.SENDER.TYPE to SM.

   If UNBIND was correlated to a particular session then
      Set MU_NEW.UNBIND_RSP_SEND.HS_ID to LULU_CB.HS_ID.
      Set MU_NEW.UNBIND_RSP_SEND.TRANSMISSION_PRIORITY to
       LULU_CB.TRANSMISSION_PRIORITY (LOW, MEDIUM, or HIGH only).
   Else
      Set MU_NEW.UNBIND_RSP_SEND.HS_ID to NULL.
      Set MU_NEW.UNBIND_RSP_SEND.TRANSMISSION_PRIORITY to LOW.

   Set MU_NEW.UNBIND_RSP_SEND.FREE_LFSID to YES.
   Copy the PATH_CONTROL_ID, LFSID and TH.SNF fields from MU into MU_NEW.
   Set the remaining TH and RH fields in MU_NEW
    to the specified values (see page 4-14 and SNA Formats).

   If an RSP(UNBIND) is positive then
      Set the only byte present in the RSP(UNBIND) RU
       to the UNBIND request code.
   Else  (RSP(UNBIND) is negative)
      Set the RU portion of the RSP(UNBIND) MU to the sense data (that describes
       an UNBIND error) followed by the UNBIND request code.

   Set MU.DCF to (RH + RU) length.

   Send RSP(UNBIND) MU to ASM.

Else (Buffer was not obtained)
   Do nothing, RSP(UNBIND) will not be sent.
```

BUILD_BIND_RSP_POS

FUNCTION:	Build +RSP(BIND).
INPUT:	MU record containing the received BIND, LULU_CB control block
OUTPUT:	MU_NEW record containing a +RSP(BIND)

Referenced procedures, FSMs, and data structures:

MU	page A-29
MU_NEW, see MU	page A-29
LOCAL	page 4-89
LULU_CB	page 4-90

```
Call buffer manager(GET_BUFFER, demand, buffer size, no wait)
 to get a demand buffer to build the +RSP(BIND) in.  The buffer size is
 set to the length of the RSP(BIND) including control vectors plus length
 of MU overhead  (Appendix B).

If buffer request failed then
   Set LOCAL.SENSE_CODE to X'0812000D' to indicate insufficient buffers
    to activate a session.
Else
   Set MU_NEW.HEADER_TYPE to BIND_RSP_SEND.
   Set MU_NEW.BIND_RSP_SEND.LU_ID to LOCAL.LU_ID.
   Set MU_NEW.BIND_RSP_SEND.SENDER.TYPE to SM.
   Set MU_NEW.BIND_RSP_SEND.TRANSMISSION_PRIORITY to NETWORK.
   Set MU_NEW.BIND_RSP_SEND.FREE_LFSID to NO.
   Set MU_NEW.BIND_RSP_SEND.HS_ID to LULU_CB.HS_ID.
   Copy the PATH_CONTROL_ID, LFSID and TH.SNF fields from MU into MU_NEW.
   Set the remaining TH and RH fields in MU_NEW
    to the specified values (see page 4-14 and SNA Formats).
   Set RSP(BIND) RU to the appropriate values (see page 4-24).
   Insert the random data found in the received BIND RU
    into the LUCB.PENDING_RANDOM_DATA_LIST.
   Set MU.DCF to (RH + RU) length.
```

CLEANUP_LU_LU_SESSION

FUNCTION:	Clean up LU-LU session.
INPUT:	LULU_CB of session to be cleaned up
OUTPUT:	LU-LU session cleaned up: SESSEND_SIGNAL sent to SS, if appropriate; the buffers reserved for this session by SM released; the half-session process for this session destroyed, if it exists; an outstanding random data entry removed from the pending random data list, if it is there

Referenced procedures, FSMs, and data structures:

BUILD_AND_SEND_SESSEND_SIG	page 4-64
UNRESERVE_BUFFERS	page 4-85
LULU_CB	page 4-90

```
If a SESSST_SIGNAL was previously sent or a CINIT_SIGNAL was received
 on this session then
   Call BUILD_AND_SEND_SESSEND_SIG(LULU_CB) (page 4-64).

Call UNRESERVE_BUFFERS(LULU_CB) to unreserve the buffers reserved for
 this session  (page 4-85).
Destroy this session's half-session process if it exists.

Remove an entry from the list of pending random data, if the random data
 for this session is there.
Remove the LULU_CB from the list so the LU-LU awareness is gone.
```

CORRELATE_BIND_RSP

FUNCTION:	Check if the received RSP(BIND) correlates with a previously sent BIND.
INPUT:	MU containing the RSP(BIND)
OUTPUT:	TRUE, if RSP(BIND) correlates; FALSE, otherwise

Referenced procedures, FSMs, and data structures:

MU page A-29

A correlation of a RSP(BIND) to BIND is a complicated procedure, partially because a number of race conditions may occur. The PATH_CONTROL_ID and LFSID fields in the RSP(BIND) MU must match those in the sent BIND MU and the session in question must be in the state where a response to BIND is expected, but this is not enough. Only if no -RSP(BIND)s are sent and every +RSP(BIND) carries an FQPCID control vector can each RSP(BIND) be properly correlated to a previously sent BIND. In doing the correlation, the following problems occur:

- A partner LU may include the FQPCID control vectors in the +RSP(BIND)/and use an UNBIND to reject a BIND; or if it is a back-level LU, not use FQPCID and send -RSP(BIND) to reject a BIND. A back-level LU returns an SNF that is used for correlation; a current-level LU does not have to return the matched SNF.

- A length error may be found while checking whether or not the RSP(BIND) contains the FQPCID control vector. Since the presence of an FQPCID is in question, an SNF parameter cannot be used for correlation, because the RSP(BIND) could arrive from an LU that didn't use it.

- Unlike BIND pacing, RSP(BIND) pacing is not required; it is possible that the ASM could not reassemble the RSP(BIND) that arrived from another node. In this case, ASM sends only the first segment of the RSP(BIND) MU to the session manager. SM recognizes it by checking the End of BIU Indicator in the TH. In this case, SM also does not know whether an FQPCID control vector was present in the RSP(BIND).

In view of the above, the following rules are used to check for the RSP(BIND) correlation:

1. PATH_CONTROL_ID and LFSID in the received RSP(BIND) must match the PATH_CONTROL and LFSID values for a pending-active session and the activation process of that session must be in a state where a RSP(BIND) is expected. If such a pending-active session is not found, no other consideration is given to this RSP(BIND).

2. The SNF fields in the BIND and -RSP(BIND) must match.

3. If it is known that +RSP(BIND) lacks an FQPCID control vector, the SNF fields in the BIND and RSP(BIND) must match.

4. If it is known that +RSP(BIND) carries an FQPCID control vector, the FQPCID must match the one in the BIND. The SNF fields are not compared.

5. If it cannot be determined whether the FQPCID control vector is present in +RSP(BIND), a RSP(BIND) is accepted as correlated under the first rule above. The SNF fields are not compared.

CORRELATE_UNBIND_RQ

FUNCTION:	Check if the received RSP(UNBIND) correlates with a known session.
INPUT:	MU containing the UNBIND
OUTPUT:	TRUE, if UNBIND correlates; otherwise, FALSE

Referenced procedures, FSMs, and data structures:

MU page A-29

A correlation of an UNBIND to a known session is a complicated procedure, although not as complicated as a correlation of a RSP(BIND) to a BIND. The PATH_CONTROL_ID and LFSID fields in the UNBIND MU header must match those used to activate the session in question, but this is not enough to complete the correlation. Only if every UNBIND carries an FQPCID control vector can each UNBIND be properly correlated to the right session. In doing the correlation, the following problems occur:

- A partner LU may include the FQPCID control vector in the UNBIND or, if it is a back-level LU, not use FQPCID control vectors.
- A length error can be found while trying to find out whether or not the RSP(UNBIND) contains the FQPCID control vector.
- The local LU may already have sent an UNBIND of its own and the (LFSID, PATH_CONTROL_ID) pair is in use for another session.

In view of the above, the following rules are used to check for the UNBIND correlation:

1. PATH_CONTROL_ID and LFSID in the received UNBIND must match the PATH_CONTROL_ID and LFSID for a pending-active or active session. If such a session is not found, the UNBIND does not correlate.
2. If the UNBIND arrives as an EXR (RH.SDI=SD), or if it contains length errors, or if it is known that the UNBIND lacks an FQPCID control vector, then UNBIND is accepted as correlated under the first rule above.
3. If it is known that the UNBIND carries an FQPCID control vector, the FQPCID must match the FQPCID of the session.

GET_FQPCID

FUNCTION:	Get the fully-qualified procedure correlation identifier (FQPCID) from the session services (SS) component of the control point. Repeat requests if a duplicate FQ PCID was received. An FQ PCID is considered duplicate if its PCID matches that for another active or pending-active session at this LU.
INPUT:	LULU_CB
OUTPUT:	ASSIGN_PCID to SS, LULU_CB.FQPCID initialized

Referenced procedures, FSMs, and data structures:

LULU_CB	page 4-90
ASSIGN_PCID	page A-22
ASSIGN_PCID_RSP	page A-23
SS	T2.1 Node Reference

```
Do until a valid PCID is found.
   Create an ASSIGN_PCID record.
   Set ASSIGN_PCID.SM_PROCESS_ID to this LU_ID.
   Set ASSIGN_PCID.DUPLICATE_PCID to NO.

   Send ASSIGN_PCID to SS.

   Receive ASSIGN_PCID_RSP from SS.

   Find an LULU_CB with the FQPCID whose PCID field matches that of an FQPCID
    for another session at this LU.
   If found then
      Create another ASSIGN_PCID record.  Set all parameters in it to the same
       values as before except that ASSIGN_PCID.DUPLICATE_PCID is set to YES.
      Send a new ASSIGN_PCID record to SS.

      Receive ASSIGN_PCID_RSP from SS.

   When an acceptable FQPCID is received, save it in LULU_CB.FQPCID.
Destroy ASSIGN_PCID record.
```

INITIALIZE_LULU_CB_ACT_SESS

FUNCTION:	Initialize an LULU_CB for an LU-LU session being activated as a result of an ACTIVATE_SESSION received from RM.
INPUT:	ACTIVATE_SESSION record, LULU_CB
OUTPUT:	The following parameters in LULU_CB are initialized: LOCAL_PARTNER_LU_NAME, FQ_PARTNER_LU_NAME, MODENAME, and SESSION_TYPE

Referenced procedures, FSMs, and data structures:

ACTIVATE_SESSION	page A-20
LULU_CB	page 4-90
PARTNER_LU	page A-2

```
Locate the PARTNER_LU control block using ACTIVATE_SESSION.LU_NAME.

Set LULU_CB.FQ_PARTNER_LU_NAME to PARTNER_LU.FULLY_QUALIFIED_LU_NAME.
Set LULU_CB.LOCAL_PARTNER_LU_NAME to PARTNER_LU.LOCAL_LU_NAME.
Set LULU_CB.MODENAME to ACTIVATE_SESSION.MODE_NAME.
Set LULU_CB.SESSION_TYPE to ACTIVATE_SESSION.SESSION_TYPE.
```

INITIALIZE_LULU_CB_BIND

FUNCTION:	Initialize an LULU_CB for an LU-LU session being activated as a result of receiving a BIND.
INPUT:	MU (containing BIND), LULU_CB
OUTPUT:	LULU_CB (initialized)

Referenced procedures, FSMs, and data structures:

GET_FQPCID	page 4-70
MU	page A-29
PARTNER_LU	page A-2
LULU_CB	page 4-90
LOCAL	page 4-89
BIND RU	SNA Formats

```
Locate the PARTNER_LU control block using the user data PLU name in BIND.
Set LULU_CB.LOCAL_PARTNER_LU_NAME to PARTNER_LU.LOCAL_LU_NAME.
Set LULU_CB.FQ_PARTNER_LU_NAME to LOCAL.USER_DATA.PLUNAME.NAME.
Set LULU_CB.MODENAME to user data mode name in BIND.
Set LULU_CB.HALF_SESSION_TYPE = SEC  (BIND receiver is secondary).

If parallel sessions are not supported with the partner LU and
 MODE.MIN_CONWINNERS_LIMIT = 1 then
   Set LULU_CB.SESSION_TYPE to FIRST_SPEAKER.

Else (negotiation is not allowed in this case)
   If BIND specifies secondary as contention winner then
      Set LULU_CB.SESSION_TYPE to FIRST_SPEAKER.
   Else
      Set LULU_CB.SESSION_TYPE to BIDDER.

Set LULU_CB.PATH_CONTROL_ID to the PATH_CONTROL_ID from the BIND.
Set LULU_CB.PC_CHARACTERISTICS to MU.BIND_RU.PC_CHARACTERISTICS.
Set LULU_CB.LFSID to MU.BIND_RU.LFSID.

If the FQPCID control vector is present in the BIND then
   Save it in LULU_CB.FQPCID.
Else
   Call GET_FQPCID(LULU_CB) to get FQPCID. (page 4-70).
   Save FQPCID in LULU_CB.FQPCID.
```

LU_MODE_SESSION_LIMIT_EXCEEDED

FUNCTION:	Determine whether or not session limits associated with a given (LU, mode name) pair are exceeded for the given state condition (FSM_STATUS for this session).
INPUT:	PARTNER_LU.FULLY_QUALIFIED_LU_NAME, MODE, session type (FIRST_SPEAKER or BIDDER), state_condition (ACTIVE, AT_LEAST_BIND_SENT, or AT_LEAST_INIT_SENT)
OUTPUT:	TRUE if session limits exceeded; otherwise, FALSE. If TRUE, LOCAL.SENSE_CODE is set to appropriate sense data.
NOTE:	If parallel sessions are not supported with the partner LU and the total session limit will not be exceeded, a session-activation request specifying this LU as first speaker is accepted. For example, a BIND is received specifying the SLU as first speaker (contention winner). The SLU does not support parallel sessions with the BIND sender and SESSION_LIMIT=1, MIN_CONWINNERS_LIMIT=0, and MIN_CONLOSERS_LIMIT=1 (these values are associated with the mode name specified in the BIND). Even though the MIN_CONWINNERS_LIMIT of 0 will be exceeded, the BIND is accepted.

Referenced procedures, FSMs, and data structures:

LOCAL page 4-89
MODE page A-3


```
If state_condition = ACTIVE then
   Set BIDDER_SESSION_COUNT to the number of active bidder sessions.
   Set FSP_SESSION_COUNT to the number of active first-speaker sessions.
   (A session is considered active if either a RSP(BIND) was received
    if PLU; or a BIND was received, if SLU).
Else
   If state_condition = AT_LEAST_BIND_SENT then
      Set BIDDER_SESSION_COUNT and FSP_SESSION_COUNT
        to the number of bidder and first-speaker sessions, respectively,
        for which a BIND is either sent or received.
   Else    (state_condition = AT_LEAST_INIT_SENT)
      Set BIDDER_SESSION_COUNT and FSP_SESSION_COUNT
        to the number of bidder and first-speaker sessions, respectively,
        for which either an INIT_SIGNAL is sent, if PLU, or a BIND
        is received, if SLU.


Set TOTAL_LIMIT to MODE.SESSION_LIMIT.
Set FSP_LIMIT to MODE.MIN_CONWINNERS_LIMIT.
Set BIDDER_LIMIT to MODE.MIN_CONLOSERS_LIMIT.

Select based on one of the following conditions:
   When FSP_SESSION_COUNT + BIDDER_SESSION_COUNT ≥ TOTAL_LIMIT
      Set LOCAL.SENSE_CODE to X'08050000' (total session limit will be exceeded).
   When FSP_SESSION_COUNT ≥ TOTAL_LIMIT - BIDDER_LIMIT and
    session_type = FIRST_SPEAKER and parallel sessions are supported with
    the partner LU (see Note).
      Set LOCAL.SENSE_CODE to X'08050001' (first speaker session limit will be
       exceeded).
   When BIDDER_SESSION_COUNT - TOTAL_LIMIT - FSP_LIMIT and session_type = BIDDER
      Set LOCAL.SENSE_CODE to X'08050001' (bidder session limit will be exceeded).
   Otherwise
      Set LOCAL.SENSE_CODE to X'00000000' (session limit will not be exceeded).

If LOCAL.SENSE_CODE = X'00000000' then
   Return with a value of FALSE (session limit will not be exceeded).
Else
   Return with a value of TRUE (session limit will be exceeded).
```

PREPARE_TO_SEND_BIND

FUNCTION:	Get the address (LFSID structure) for the session. Create a half-session process. Reserve buffers for the session.
INPUT:	LULU_CB control block
OUTPUT:	ASSIGN_LFSID record sent to ASM; HS process created; if an error occurs, LOCAL.SENSE_CODE set

Referenced procedures, FSMs, and data structures:

RESERVE_CONSTANT_BUFFERS	page 4-84
LULU_CB	page 4-90
ASSIGN_LFSID	page A-25
ASSIGN_LFSID_RSP	page A-26
LOCAL	page 4-89
ASM	T2.1 Node Reference


```
Create an ASSIGN_LFSID record.
Set ASSIGN_LFSID.PATH_CONTROL_ID to LULU_CB.PATH_CONTROL_ID.
Set ASSIGN_LFSID.SM_PROCESS_ID to this LU's identifier.
Set ASSIGN_LFSID.PROCESS_ID_TYPE to SM.
Send ASSIGN_LFSID to ASM.

Receive ASSIGN_LFSID_RSP from ASM.
If ASSIGN_LFSID_RSP.SENSE_CODE is not  X'00000000' then (ASM couldn't assign LFSID)
  Set LOCAL.SENSE_CODE to ASSIGN_LFSID_RSP.SENSE_CODE.
Else   (LFSID is assigned)
   Set LULU_CB.LFSID to ASSIGN_LFSID_RSP.LFSID.
   Create this half-session's process.
   Call RESERVE_CONSTANT_BUFFERS(LULU_CB) to adjust the permanent
    buffer pool, and to get a demand buffer for the UNBIND
    (page 4-84).
   Destroy ASSIGN_LFSID_RSP record.
```

PROCESS_ABEND_NOTIFICATION

FUNCTION:	Process an abend notification record from a child process (RM or HS). If HS abends, the FSM is called to clean up the session for the HS (if that session still exists). If RM abends, the FSM is called once for each active or pending-active session known to SM, to clean up all of them; after that, SM itself abends.
INPUT:	ABEND_NOTIFICATION record
OUTPUT:	None

Referenced procedures, FSMs, and data structures:

LOCAL	page 4-89
LULU_CB	page 4-90
FSM_STATUS	page 4-86
ABEND_NOTIFICATION	page A-25

```
Select based on ABEND_NOTIFICATION.ABENDING_PROCESS parameter:
   When RM_PROCESS_VARIABLE   (RM process abends)
      For each active and pending session
       (i.e., for each LULU_CB in LOCAL.LULU_CB_LIST)
         Call FSM_STATUS(ABEND_NOTIFICATION, LULU_CB) (page 4-86).

   When HS_PROCESS_VARIABLE   (HS process abends)
      Determine which LU-LU session is to be terminated by searching through
       the LOCAL.LULU_CB_LIST control block list for an LULU_CB with a half-session
       identifier (HS_ID) matching that of the half-session identifier in the
       ABEND_NOTIFICATION record (ABEND_NOTIFICATION.PROCESS_ID).
      If the LULU_CB is located then
         Call FSM_STATUS(ABEND_NOTIFICATION, LULU_CB) (page 4-86).
```

PROCESS_ABORT_HS

FUNCTION:	Process an ABORT_HS record received from LU-LU half-session. If the ABORT_HS record points to a known session, call the FSM to perform the session deactivation. Otherwise (ABORT_HS does not correlate to any session), ABORT_HS is ignored. This situation can occur when SM has already destroyed HS, but ABORT_HS is still waiting on the queue.
INPUT:	ABORT_HS record
OUTPUT:	None
NOTE:	A half-session cannot send ABORT_HS until it is initialized.

Referenced procedures, FSMs, and data structures:

FSM_STATUS	page 4-86
LOCAL	page 4-89
ABORT_HS	page A-9
LULU_CB	page 4-90

```
Determine which LU-LU session is being aborted by searching through the
 LOCAL.LULU_CB_LIST control block list for an LULU_CB with a half-session
 identifier (HS_ID) matching that of the half-session that sent the ABORT_HS
 record (ABORT_HS.HS_ID).
If the LULU_CB is located then
   Call FSM_STATUS(ABORT_HS, LULU_CB) (page 4-86).
```

PROCESS_ACTIVATE_SESSION

FUNCTION:	Process an ACTIVATE_SESSION record received from RM. That includes checking for a session limit to be exceeded (since RM does not know whether the session limit is exceeded when it sends ACTIVATE_SESSION to SM), creating and initializing of the LULU_CB control block, getting an FQPCID for the session from SS, and sending an INIT_SIGNAL record to SS.
INPUT:	ACTIVATE_SESSION record
OUTPUT:	LULU_CB created and initialized, ACTIVATE_SESSION record and LULU_CB passed to the FSM

Referenced procedures, FSMs, and data structures:

LU_MODE_SESSION_LIMIT_EXCEEDED	page 4-72
BUILD_AND_SEND_ACT_SESS_RSP_NEG	page 4-58
INITIALIZE_LULU_CB_ACT_SESS	page 4-70
GET_FQPCID	page 4-70
FSM_STATUS	page 4-86
ACTIVATE_SESSION	page A-20
PARTNER_LU	page A-2
MODE	page A-3
LULU_CB	page 4-90
SS	T2.1 Node Reference

```
Locate the PARTNER_LU and MODE control blocks using the LU_NAME and MODE_NAME from
 the passed ACTIVATE_SESSION record.

If LU_MODE_SESSION_LIMIT_EXCEEDED(PARTNER_LU.FULLY_QUALIFIED_LU_NAME, MODE,
 ACTIVATE_SESSION.SESSION_TYPE,
 state_condition(AT_LEAST_INIT_SENT)) then  (page 4-72).
     (The number of active and pending-active sessions is already
      equal to the limit set for this [partner LU, mode] pair)
   Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(ACTIVATE_SESSION.CORRELATOR, RETRY)
        (page 4-58).

Else  (a session limit is not exceeded)
   Create an LU-LU control block (LULU_CB) and initialize its fields.
   Call GET_FQPCID(LULU_CB) (page 4-70);
    get FQPCID and save in LULU_CB.
   Call INITIALIZE_LULU_CB_ACT_SESS(ACTIVATE_SESSION, LULU_CB)
         (page 4-70).
   Call FSM_STATUS(ACTIVATE_SESSION, LULU_CB)    (page 4-86).
```

PROCESS_BIND_RQ

FUNCTION:	Check BIND for semantic and state errors, create a half-session process, reserve required buffers. If no errors occur, build and send a +RSP(BIND), update and save active session parameters, and initialize the half-session.
INPUT:	MU record containing BIND Before passing a BIND to SM, the address space manager checks that the length of the BIND RU corresponds to the lengths of all fields in the BIND. If not, the BIND is rejected with the appropriate sense data. ASM also checks that the total length of the Structured User Data subfields field in the BIND corresponds to the lengths of the individual subfields, that the lengths of the NS PLU and SLU Name fields do not exceed 17 bytes, the length of the URC field does not exceed 12 bytes, the length of the data portion of the FQPCID control vector is between 9 and 26 bytes, and that at least one control vector is present in the BIND if the Control Vector Included indicator in the BIND is set to 1.
OUTPUT:	If an error is found, an UNBIND or a -RSP(BIND) is sent to ASM; if no error is found, LULU_CB is created and initialized, the half-session process is created and initialized, the appropriate buffers are reserved, the SESSION_TYPE and SESSION_ID parameters in LULU_CB are updated as a result of the BIND negotiation, and a +RSP(BIND) is sent to ASM.

Referenced procedures, FSMs, and data structures:

BIND_RQ_STATE_ERROR	page 4-52
INITIALIZE_LULU_CB_BIND	page 4-71
RESERVE_VARIABLE_BUFFERS	page 4-84
RESERVE_CONSTANT_BUFFERS	page 4-84
CLEANUP_LU_LU_SESSION	page 4-67
BUILD_AND_SEND_INIT_HS	page 4-61
BUILD_AND_SEND_UNBIND_RQ	page 4-65
BUILD_BIND_RSP_POS	page 4-67
BUILD_AND_SEND_BIND_RSP_NEG	page 4-60
FSM_STATUS	page 4-86
LOCAL	page 4-89
MU	page A-29
BIND_RQ_RCV, see MU	page A-29
MU_NEW, see MU	page A-29
LULU_CB	page 4-90
PARTNER_LU	page A-2
ASM	T2.1 Node Reference

```
Check BIND request for semantic errors and if an error exists,
 set LOCAL.SENSE_CODE to the sense data reflecting the error.
 Semantic errors are field content errors (e.g., a field does not
 contain an allowable value). These errors are state-independent.

Call BIND_RQ_STATE_ERROR(MU) (page 4-52)
 to check for state errors. If an error is found, LOCAL.SENSE_CODE
 contains the sense data indicating the type of error.

If no errors are found then
   Set PARTNER_LU.ACTIVE_SESSION_PARAMETERS.PARALLEL_SESSIONS =
    BIND_RQ_RCV.PARALLEL_SESSIONS.
   Create an LULU_CB control block and initialize its fields.
   Call INITIALIZE_LULU_CB_BIND(MU, LULU_CB) (page 4-71).
   Create LU-LU half-session with unique identifier (save identifier in
    LULU_CB.HS_ID).
   Call BUILD_BIND_RSP_POS(MU, LULU_CB, MU_NEW_PTR) (page 4-67).
   Call RESERVE_CONSTANT_BUFFERS(LULU_CB) to adjust the permanent buffer pool
    and to get a demand buffer for an UNBIND (page 4-84).
   If buffers were gotten then
      Call RESERVE_VARIABLE_BUFFERS(LULU_CB, BIND_RQ_RCV)
           to reserve pacing buffers for the session (page 4-84).
   If all buffers were gotten then
      Save a negotiated 8-byte session identifier in LULU_CB.SESSION_ID.
      Call BUILD_AND_SEND_INIT_HS(LULU_CB, first 26 bytes of negotiated BIND image)
       (page 4-61).
```

```
If no errors are found during the BIND processing and all required buffers
 are available then
   Send MU_NEW containing a positive RSP(BIND) to ASM.
   Call FSM_STATUS(MU_NEW, LULU_CB) (page 4-86).

Else   (there are errors, session will not be brought up)
   If the FQPCID control vector is present in the BIND then
      Call BUILD_AND_SEND_UNBIND_RQ(MU, CLEANUP type, LOCAL.SENSE_CODE)
        (page 4-65).
   Else  (FQPCID is not present in BIND or
          errors in BIND do not allow checking whether it is present or not)
      If a demand buffer was gotten for the RSP(BIND) then
         Call buffer manager(FREE_BUFFER, buffer address) to free
           the demand buffer (Appendix B).
      Call BUILD_AND_SEND_BIND_RSP_NEG(MU) (page 4-60).
   If LULU_CB control block was created for the session then
      Call CLEANUP_LU_LU_SESSION(LULU_CB)  (page 4-67).
```

PROCESS_BIND_RSP

FUNCTION:	Check if the received RSP(BIND) correlates with the previously sent BIND. If it does, delete pending random data used in LU-LU verification for the session (if present) and after additional processing (in case of a positive response) call the FSM. If it does not correlate, the RSP(BIND) is considered to be a stray one and is ignored (no action taken).
INPUT:	MU record containing the RSP(BIND)
OUTPUT:	If the RSP(BIND) correlates and LU-LU verification is active for the session, the corresponding random data needed for the verification is removed from the list of pending random data.

Referenced procedures, FSMs, and data structures:


```
Call CORRELATE_BIND_RSP(MU) (page 4-68).
 to check whether a RSP(BIND) correlates to an outstanding BIND.
If it correlates then
   Set PARTNER_LU.ACTIVE_SESSION_PARAMETERS.PARALLEL_SESSIONS =
    BIND_RSP_RCV.PARALLEL_SESSIONS.
   Remove an entry from the list of pending random data, if the random data
    for this session is there.
   If the RSP(BIND) is positive and no length errors were found while
    correlating it to a previously sent BIND then
      Check RSP(BIND) for semantic errors and if an error exists,
       set LOCAL.SENSE_CODE with the sense data reflecting error.
       Semantic errors are field content errors (e.g., a field does not
       contain an allowable value).  These errors are state-independent.

      Call BIND_RSP_STATE_ERROR(MU, LULU_CB) (page 4-54)
       to check for state errors.  If an error is found, LOCAL.SENSE_CODE
       contains the sense data indicating the type of error.

      If no errors were found then
         Call RESERVE_VARIABLE_BUFFERS(LULU_CB, BIND_RSP_RCV) to reserve
          buffers for this session (page 4-84).
         Call BUILD_AND_SEND_INIT_HS(LULU_CB, first 26 bytes of negotiated BIND image)
           (page 4-61).
   Call FSM_STATUS(MU(RSP(BIND)), LULU_CB) (page 4-86).
```

PROCESS_CINIT_SIGNAL

FUNCTION:	Process a received CINIT_SIGNAL record. First, this signal must be correlated with a previously sent INIT_SIGNAL record. The correlation is based on the value of FQPCID. If the correlation fails, the session has already been brought down by RM and a SESSEND_SIGNAL record is built and sent to SS. Otherwise (i.e., CINIT_SIGNAL is correlated to a pending-active session), the session count is checked, the link buffer size is checked to be sufficiently large, LULU_CB is initialized with the additional parameters received in the CINIT_SIGNAL record, LFSID is obtained, the half-session process is created, and the buffers for the session are reserved. If no errors are found, the BIND is sent.
INPUT:	CINIT_SIGNAL record
OUTPUT:	LULU_CB updated, SESSEND_SIGNAL sent if the CINIT_SIGNAL record could not be correlated to a previously sent INIT_SIGNAL, a BIND sent if no errors are found
NOTE:	Some of the buffers for the session cannot be obtained before the RSP(BIND) is received because the lengths of these buffers depend upon the negotiated RU sizes and window sizes (see Figure 4-10 on page 4-32).

Referenced procedures, FSMs, and data structures:


```
Try to correlate CINIT_SIGNAL to a previously sent INIT_SIGNAL
 using the FQPCID parameter.

If a CINIT_SIGNAL does not correlate with any outstanding INIT_SIGNAL then
   Create a SESSEND_SIGNAL record.
   Set SESSEND_SIGNAL.SENSE_CODE to X'00000000' (sense data is immaterial in this case).
   Set SESSEND_SIGNAL.FQPCID to CINIT_SIGNAL.FQPCID.
   Set SESSEND_SIGNAL.PATH_CONTROL_ID to CINIT_SIGNAL.PATH_CONTROL_ID.
   Send a SESSEND_SIGNAL record to SS.
```

Else (CINIT_SIGNAL correlated, a pending session is identified)
  Call LU_MODE_SESSION_LIMIT_EXCEEDED(PARTNER_LU.FULLY_QUALIFIED_LU_NAME, MODE, LULU_CB.SESSION_TYPE, state_condition(AT_LEAST_BIND_SENT)) (page 4-72) to check whether session limit is exceeded. Count only active sessions and those pending-active sessions where BIND has already been sent. If error is found, the called routine sets LOCAL.SENSE_CODE. (BIND has a priority over CINIT_SIGNAL, which, in turn, has a priority over ACTIVATE_SESSION.)
  Check whether the link buffer size is sufficiently large to satisfy the lower bound value for the RU sizes specified in the MODE control block.
  Check if an active session already exists with this partner, and that the partner supports parallel sessions. If error is found, set LOCAL.SENSE_CODE to X'08050000'.

  If no errors were discovered when the above checks were made then
    Call PREPARE_TO_SEND_BIND(LULU_CB) (page 4-73) to initialize additional fields in LULU_CB, obtain LFSID, create a half-session process, and get buffers (see Note).

    If all is ready to send a BIND (i.e., no errors found) then
      Call BUILD_AND_SEND_BIND_RQ(LULU_CB) (page 4-59).

Call the FSM whether or not an error was found while processing a correlated CINIT_SIGNAL record. The FSM will take appropriate action depending upon whether or not LOCAL.SENSE_CODE is set to X'00000000'.

  Call FSM_STATUS(CINIT_SIGNAL, LULU_CB) (page 4-86).

PROCESS_DEACTIVATE_SESSION

FUNCTION:	Process a DEACTIVATE_SESSION record received from RM.
INPUT:	DEACTIVATE_SESSION record
OUTPUT:	If the DEACTIVATE_SESSION record points to a known session, call the FSM to deactivated that session.

Referenced procedures, FSMs, and data structures:

FSM_STATUS	page 4-86
DEACTIVATE_SESSION	page A-21
LULU_CB	page 4-90

If RM is deactivating a pending-active session (DEACTIVATE_SESSION.STATUS = PENDING) then
  Attempt to locate the LU-LU half-session control block (LULU_CB) using the DEACTIVATE_SESSION.CORRELATOR field.

Else (RM is deactivating an active session--from its perspective)
  Attempt to locate the LU-LU half-session control block (LULU_CB) using the DEACTIVATE_SESSION.HS_ID field.

If an LULU_CB has been located then
  Call FSM_STATUS(DEACTIVATE_SESSION, LULU_CB) (page 4-86).

PROCESS_INIT_HS_RSP

FUNCTION:	Process an INIT_HS_RSP record received from a half-session.
INPUT:	INIT_HS_RSP record
OUTPUT:	If the INIT_HS_RSP record points to a known session, call the FSM to complete the session activation

Referenced procedures, FSMs, and data structures:

FSM_STATUS	page 4-86
INIT_HS_RSP	page A-10
LULU_CB	page 4-90

Attempt to locate the LU-LU half-session control block (LULU_CB) associated with the half-session that sent the INIT_HS_RSP. Search the list of LULU_CBs for one with a half-session identifier (HS_ID) matching that of the half-session the INIT_HS_RSP was received from.
If an LULU_CB is located then
  Call FSM_STATUS(INIT_HS_RSP, LULU_CB) (page 4-86).

PROCESS_INIT_SIGNAL_NEG_RSP

FUNCTION:	Process a received INIT_SIGNAL_NEG_RSP record from the SS component of the control point. If an outstanding request to activate a session that corresponds to this record from SS is found, call FSM in order to terminate it.
INPUT:	INIT_SIGNAL_NEG_RSP record
OUTPUT:	If an outstanding request to activate a session can be found that correlates with the INIT_SIGNAL_NEG_RSP record, call the FSM to terminate the session.

Referenced procedures, FSMs, and data structures:

FSM_STATUS	page 4-86
INIT_SIGNAL_NEG_RSP	page A-23
INIT_SIGNAL	page A-23
LULU_CB	page 4-90
SS	T2.1 Node Reference

Attempt to correlate the INIT_SIGNAL_NEG_RSP with a sent INIT_SIGNAL.
Search for an LULU_CB control block where LULU_CB.FQPCID = INIT_SIGNAL_NEG_RSP.FQPCID.

If the received record is correlated successfully then
  Call FSM_STATUS(INIT_SIGNAL_NEG_RSP, LULU_CB) (page 4-86).

PROCESS_LFSID_IN_USE

FUNCTION:	Process a received LFSID_IN_USE record. This record is sent to SM by ASM so that ASM will know whether a given (LFSID, PATH_CONTROL_ID) pair is currently in use. ASM must know before it sends a BIND to an appropriate LU. If the pair is in use, ASM will hold the BIND in order to avoid certain race conditions.
INPUT:	LFSID_IN_USE record
OUTPUT:	LFSID_IN_USE_RSP record to ASM

Referenced procedures, FSMs, and data structures:

LFSID_IN_USE	page A-26
LFSID_IN_USE_RSP	page A-25
ASM	T2.1 Node Reference

```
Find an active or pending-active session with a given (LFSID, PATH_CONTROL_ID) pair.

Create an LFSID_IN_USE_RSP record.
Set LFSID_IN_USE_RSP.PATH_CONTROL_ID to LFSID_IN_USE.PATH_CONTROL_ID.
Set LFSID_IN_USE_RSP.LFSID to LFSID_IN_USE.LFSID.

If a session with a given (LFSID, PATH_CONTROL_ID) pair was found then
   Set LFSID_IN_USE_RSP.ANSWER to YES.
Else
   Set LFSID_IN_USE_RSP.ANSWER to NO.

Send the LFSID_IN_USE_RSP record to ASM.
```

PROCESS_MU

FUNCTION:	Process an MU record.
INPUT:	MU containing BIND, RSP(BIND), or UNBIND
OUTPUT:	MU is forwarded to the appropriate procedure. If the buffer holding the MU is not reused by SM, the buffer is freed.

Referenced procedures, FSMs, and data structures:

PROCESS_BIND_RQ	page 4-76
PROCESS_BIND_RSP	page 4-78
PROCESS_UNBIND_RQ	page 4-83
MU	page A-29

```
Select based on MU.HEADER_TYPE
   When BIND_RQ_RCV
      Call PROCESS_BIND_RQ(MU)    (page 4-76).
   When BIND_RSP_RCV
      Call PROCESS_BIND_RSP(MU)    (page 4-78).
   When UNBIND_RQ_RCV
      Call PROCESS_UNBIND_RQ(MU)    (page 4-83).

If buffer holding the MU was not reused while processed then
   Call buffer manager(FREE_BUFFER, MU pointer) to free the MU buffer.
```

PROCESS_SESSION_ROUTE_INOP

FUNCTION:	Process a SESSION_ROUTE_INOP record received from ASM.
INPUT:	SESSION_ROUTE_INOP record
OUTPUT:	The FSM is called for each session using the path control that has failed.

Referenced procedures, FSMs, and data structures:

FSM_STATUS	page 4-86
SESSION_ROUTE_INOP	page A-24
LULU_CB	page 4-90

```
Reset all LU-LU sessions that are using the path control process that failed.
 This is done by locating all the LU-LU session control blocks (LULU_CBs)
 that have a path control identifier (PC_ID) matching that of the path
 control process that failed.  For each LULU_CB located,
 Call FSM_STATUS(SESSION_ROUTE_INOP, LULU_CB) (page 4-86)
 to reset that session.
```

PROCESS_UNBIND_RQ

FUNCTION:	Process a received UNBIND. SM always receives the entire UNBIND MU, since the PIU is not longer than 99 bytes and thus no reassembly by the ASM is needed. If a received UNBIND correlates to one of the active or pending-active sessions, the FSM is called to clean up the session.
INPUT:	MU record containing UNBIND
OUTPUT:	Whether or not UNBIND correlates, a RSP(UNBIND) is sent.

Referenced procedures, FSMs, and data structures:

BUILD_AND_SEND_UNBIND_RSP	page 4-66
CORRELATE_UNBIND_RQ	page 4-69
FSM_STATUS	page 4-86
MU	page A-29
UNBIND_RQ_RCV, see MU	page A-29
LULU_CB	page 4-90

```
Call CORRELATE_UNBIND_RQ(MU) (page 4-69).
 to check whether an UNBIND correlates with an existing session.

Call BUILD_AND_SEND_UNBIND_RSP(MU) (page 4-66) to send
 a RSP(UNBIND) regardless of whether UNBIND correlated or not.
 A negative response will be sent if UNBIND was received as an EXR
   or contained length errors.
 Otherwise, +RSP(UNBIND) will be sent.

If UNBIND correlated to an active or pending-active session then
   Call FSM_STATUS(MU(UNBIND_RQ_RCV), LULU_CB)
    (page 4-86) to terminate that session.
```

RESERVE_CONSTANT_BUFFERS

FUNCTION:	Increment the size of the permanent buffer pool. Get a demand buffer for an UNBIND.
INPUT:	LULU_CB control block
OUTPUT:	The buffers are reserved.

Referenced procedures, FSMs, and data structures:

LULU_CB	page 4-90
LOCAL	page 4-89

```
Call buffer manager(ADJUST_BUF_POOL, permanent buffer pool ID, change amount)
 to adjust (increase) the number of buffers in the permanent buffer
 pool.  Change amount is set to 1, which means that the size is
 incremented by a value determined by the buffer manager
 (Appendix B).

If additional buffers are available then
   Set LULU_CB.PERM_POOL_ADJUSTED_UP to YES.
   Call buffer manager(GET_BUFFER, demand, buffer size, no wait)
    to get a demand buffer to build an UNBIND.  Buffer size is set
    to the maximum size of an UNBIND RU plus length of MU overhead
    (Appendix B).

If one of the buffer requests was unsuccessful then
   Set LOCAL.SENSE_CODE to X'0812000D' (insufficient buffers exist
    to activate a session).
Return.
```

RESERVE_VARIABLE_BUFFERS

FUNCTION:	Reserve a dynamic buffer pool, and a limited buffer pool.
INPUT:	LULU_CB, bind image (either BIND_RQ_RCV or BIND_RSP_RCV)
OUTPUT:	Reserve pacing buffers for the session.

Referenced procedures, FSMs, and data structures:

MU	page A-29
BIND_RQ_RCV, see MU	page A-29
BIND_RSP_RCV, see MU	page A-29
LULU_CB	page 4-90
LOCAL	page 4-89

```
If BIND_RSP_RCV.ADAPTIVE_PACING = SUPPORTED then
   Call buffer manager(CREATE_BUF_POOL, varying dynamic, pool owner,
    capacity of pool, buffer size, initial number of buffers) to reserve
    the dynamic buffer pool for the receive pacing buffers.  Pool owner
    is set to LULU_CB.HS_ID, capacity of pool and buffer size are set to
    the negotiated values from the BIND, dynamic pool ID is returned
    by the buffer manager (Appendix B).
   If buffers were reserved then
      Call buffer manager(CREATE_BUF_POOL, limited, pool owner, capacity
       of pool, buffer size) to reserve the limited buffer pool for
       the send pacing buffers.  Pool owner is set to LULU_CB.HS_ID,
       capacity of pool and buffer size are set to the negotiated
       values from the BIND, limited pool ID is returned by the buffer
       manager (Appendix B).
Else
   Call buffer manager(CREATE_BUF_POOL, fixed dynamic, pool owner,
    capacity of pool, buffer size) to reserve the dynamic buffer pool
    for the receive pacing buffers.  Pool owner is set to LULU_CB.HS_ID,
    capacity of pool and buffer size are set to the negotiated values
    from the BIND, dynamic pool ID is returned by the buffer manager
    (Appendix B).
   If buffers were reserved then
      Call buffer manager(CREATE_BUF_POOL, limited, pool owner, capacity
       of pool, buffer size) to reserve the limited buffer pool for the
       send pacing buffers.  Pool owner is set to LULU_CB.HS_ID, capacity
       of pool and buffer size are set to the negotiated values from the
       BIND, limited  pool ID is returned by the buffer manager
       (Appendix B).
If any of the buffer requests were unsuccessful then
   Set LOCAL.SENSE_CODE to X'0812000D' (insufficient buffers exist to
    activate session).
```

UNRESERVE_BUFFERS

FUNCTION:	Unreserve (i.e., releases previously reserved buffers) appropriate buffers for the session
INPUT:	LULU_CB, permanent buffer pool ID
OUTPUT:	Buffers are unreserved

```
If the size of the permanent buffer pool was increased when this
 session was activated then
   Call buffer manager(ADJUST_BUF_POOL, permanent buffer pool ID, change
    amount) to reduce the number of buffers in the permanent buffer pool.  Set
    change amount to the value (negative) the permanent buffer pool was
    increased by when this session was activated (Appendix B).

If a demand buffer for UNBIND was previously reserved then
   Call buffer manager(FREE_BUFFER, UNBIND buffer address) to free the
    demand buffer gotten for the UNBIND (Appendix B).

The limited buffer pool, and the dynamic buffer pool, will be destroyed
 when the owning HS is destroyed.
```

FSM_STATUS

FUNCTION: This FSM maintains the state of an LU-LU session from initiation through termination. State name abbreviations and their meanings are as follows:

- RESET = reset
- PEND CINIT = pending receipt of CINIT_SIGNAL record
- PEND BIND RSP = pending receipt of a RSP(BIND)
- PEND INIT HS RSP PLU = pending receipt of INIT_HS_RSP when this LU is a PLU
- PEND INIT HS RSP SLU = pending receipt of INIT_HS_RSP when this LU is an SLU
- ACTIVE = active

INPUT: The record to be processed and the LU-LU half-session control block (LULU_CB). These inputs denote RUs, interprocess records (i.e., from HS, RM, SS, or ASM [see Appendix A]), results of earlier sense data settings (OK, if LOCAL.SENSE_CODE was set to 00000000; NG, otherwise), and session roles (PLU or SLU) of the local LU.

OUTPUT: The output depends upon the state of the FSM and upon the type of the input record. LULU_CB can be updated. An MU can be created and sent to ASM. See particular output code for the details.

NOTE: Error type is "retry" if LOCAL.SENSE_CODE has one of the following sense data values (an asterisk means any hexadecimal digit allowed):

- 0801****
- 0805****
- 0812****
- 0837****
- 0839****
- 0842****
- 0845****
- 084B****
- 0856****
- 0857****
- 8001****
- 8002****
- 8003****
- 8013**00
- 8013**03
- 8013**04
- 8013**05
- 8013**06

For any other value of LOCAL.SENSE_CODE error type is "no retry."

Referenced procedures, FSMs, and data structures:

CLEANUP_LU_LU_SESSION	page 4-67
BUILD_AND_SEND_UNBIND_RQ	page 4-65
BUILD_AND_SEND_INIT_SIG	page 4-61
BUILD_AND_SEND_ACT_SESS_RSP_NEG	page 4-58
BUILD_AND_SEND_PC_HS_DISCONNECT	page 4-62
BUILD_AND_SEND_SESSST_SIG	page 4-65
BUILD_AND_SEND_SESS_ACTIVATED	page 4-63
BUILD_AND_SEND_FREE_LFSID	page 4-60
BUILD_AND_SEND_SESS_DEACTIVATED	page 4-64
BUILD_AND_SEND_ACT_SESS_RSP_POS	page 4-58
LULU_CB	page 4-90
LOCAL	page 4-89
ACTIVATE_SESSION	page A-20
DEACTIVATE_SESSION	page A-21
ABORT_HS	page A-9
INIT_HS_RSP	page A-10
ABEND_NOTIFICATION	page A-25
MU	page A-29
BIND_RQ_RCV, see MU	page A-29
BIND_RSP_RCV, see MU	page A-29
UNBIND_RQ_RCV, see MU	page A-29
UNBIND_RSP_RCV, see MU	page A-29

All MUs sent from this FSM will be in the MU_NEW buffer to distinguish them from the MU, which may contain the input signal and will be freed after the processing is done.

MU_NEW, see MU	page A-29
INIT_SIGNAL_NEG_RSP	page A-23
CINIT_SIGNAL	page A-23
SESSION_ROUTE_INOP	page A-24

STATE NAMES---->	RESET	PEND CINIT	PEND BIND RSP	PEND INIT HS RSP PLU	PEND INIT HS RSP SLU	ACTIVE
INPUTS STATE NUMBERS-->	01	02	03	04	05	06
ACTIVATE_SESSION	2A	/	/	/	/	/
INIT_SIGNAL_NEG_RSP	/	1B	/	/	/	/
CINIT_SIGNAL,OK	/	3	/	/	/	/
CINIT_SIGNAL,NG	/	1L	/	/	/	/
+RSP(BIND),OK	/	/	4	/	/	/
+RSP(BIND),NG	/	/	1R	/	/	/
-RSP(BIND)	/	/	1E	/	/	/
BIND	5	/	/	/	/	/
+INIT_HS_RSP	/	/	/	6S	6J	/
-INIT_HS_RSP	/	/	/	1H	1N	/
DEACTIVATE_SESSION	/	1C	1P	1P	/	1P
UNBIND	/	/	1G	1G	1C	1I
SESSION_ROUTE_INOP	/	/	1K	1K	1C	1M
ABORT_HS	/	/	/	1U	1V	1Q
RM_ABEND	/	1C	1D	1D	1D	1D
HS_ABEND	/	/	1T	1T	1D	1F

OUTPUT CODE	FUNCTION
A	Call BUILD_AND_SEND_INIT_SIG(LULU_CB) (page 4-61).

B	Determine the error type by examining the sense data in the INIT_SIGNAL_NEG_RSP record (see Note). Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, error type) (page 4-58). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
C	Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
D	Call BUILD_AND_SEND_UNBIND_RQ(MU_NEW, CLEANUP, X'08120000') (page 4-65). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
E	Call BUILD_AND_SEND_PC_HS_DISCONNECT(LULU_CB) (page 4-62). Determine the error type by examining the sense data in the RSP(BIND) record (see Note). Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, error type) (page 4-58). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
F	Call BUILD_AND_SEND_UNBIND_RQ(MU_NEW, CLEANUP, X'08120000') (page 4-65). Call BUILD_AND_SEND_SESS_DEACTIVATED(LULU_CB.HS_ID, ABNORMAL_RETRY, X'08120000') (page 4-64). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
G	Determine the error type by examining the sense data in the UNBIND record (see Note). Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, error type) (page 4-58). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
H	Call BUILD_AND_SEND_UNBIND_RQ(MU_NEW, FORMAT_OR_PROTOCOL_ERROR, INIT_HS_RSP.SENSE_CODE) (page 4-65). Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, NO_RETRY) (page 4-58). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
I	Determine the reason for session deactivation. If the UNBIND type is Normal or BIND Forthcoming, the reason is NORMAL. If the UNBIND type is Invalid Session Parameters, or LU Failure Unrecoverable, or Format or Protocol Error, the reason is ABNORMAL_NO_RETRY. For all other UNBIND types, the reason is ABNORMAL_RETRY. Call BUILD_AND_SEND_SESS_DEACTIVATED(LULU_CB.HS_ID, REASON, sense data from UNBIND) (page 4-64). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
J	Call BUILD_AND_SEND_SESSST_SIG(LULU_CB) (page 4-65). Call BUILD_AND_SEND_SESS_ACTIVATED(LULU_CB) (page 4-63).
K	Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, RETRY) (page 4-58). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
L	If LFSID for the session was received but the session activation stopped because of an error then Call BUILD_AND_SEND_FREE_LFSID(LULU_CB) (page 4-60). (This is the only place where it has to be done by sending a FREE_LFSID record to ASM. If a BIND has already been sent then an instruction to free LFSID goes to ASM on an UNBIND or a RSP(UNBIND) MU header.) Determine the error type by examining the sense data that describes the condition that prevents session activation (see Note). Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, error type) (page 4-58). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
M	Call BUILD_AND_SEND_SESS_DEACTIVATED(LULU_CB.HS_ID, ABNORMAL_RETRY, X'80020000') (page 4-64). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).

N	Call BUILD_AND_SEND_UNBIND_RQ(MU_NEW, FORMAT_OR_PROTOCOL_ERROR, INIT_HS_RSP.SENSE_CODE) (page 4-65). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
P	Determine an UNBIND type based on the DEACTIVATE_SESSION.TYPE parameter. UNBIND type is NORMAL, CLEANUP, or FORMAT_OR_PROTOCOL_ERROR when DEACTIVATE_SESSION.TYPE is NORMAL, CLEANUP, or ABNORMAL; correspondingly. Call BUILD_AND_SEND_UNBIND_RQ(MU_NEW, UNBIND type, DEACTIVATE_SESSION.SENSE_CODE) (page 4-65). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
Q	Call BUILD_AND_SEND_UNBIND_RQ(MU_NEW, FORMAT_OR_PROTOCOL_ERROR, ABORT_HS.SENSE_CODE) (page 4-65). Call BUILD_AND_SEND_SESS_DEACTIVATED(LULU_CB.HS_ID, ABNORMAL_NO_RETRY, ABORT_HS.SENSE_CODE) (page 4-64). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
R	Call BUILD_AND_SEND_UNBIND_RQ(MU_NEW, INVALID_PARMS, LOCAL.SENSE_CODE) (page 4-65, LOCAL.SENSE_CODE describes the error that was discovered while processing the RSP(BIND)). Determine the error type based on LOCAL.SENSE_CODE(see Note). Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, error type) (page 4-58). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
S	Call BUILD_AND_SEND_ACT_SESS_RSP_POS(LULU_CB) (page 4-58).
T	Call BUILD_AND_SEND_UNBIND_RQ(MU_NEW, CLEANUP, X'08120000') (page 4-65). Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, RETRY) (page 4-58). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
U	Call BUILD_AND_SEND_UNBIND_RQ(MU_NEW, FORMAT_OR_PROTOCOL_ERROR, ABORT_HS.SENSE_CODE) (page 4-65). Determine the error type by examining the sense data in the ABORT_HS record (see Note). Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, error type) (page 4-58). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).
V	Call BUILD_AND_SEND_UNBIND_RQ(MU_NEW, FORMAT_OR_PROTOCOL_ERROR, ABORT_HS.SENSE_CODE) (page 4-65). Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-67).

LOCAL DATA STRUCTURES

LOCAL

LOCAL (this control block is accessible by any procedure in SM)
  LULU_CB_LIST list of LU-LU half-session control blocks (page 4-90)
  SENSE_CODE (this field is set with a sense data value whenever an error is found)
  LU_ID (SM process ID)
  PLUNAME
    NAME(primary LU name)

LULU_CB
The LU-LU session control block is used by session manager (SM) to keep information about an LU-LU session. One LULU_CB exists for each LU-LU session.

LULU_CB

> The following fields are always set to the correct value when the LULU_CB is created and initialized (independent of what caused it to be created).

LOCAL_PARTNER_LU_NAME: locally known name for the partner LU
FQ_PARTNER_LU_NAME: partner LU's network qualified name
MODENAME: mode name for this LU-LU session
HALF_SESSION_TYPE: possible values: PRI, SEC
SESSION_TYPE: possible values: FIRST_SPEAKER, BIDDER

> CORRELATOR field is set when an ACTIVATE_SESSION (from RM) causes the creation of the LULU_CB. It is used by RM to correlate ACTIVATE_SESSION_RSP to ACTIVATE_SESSION.

CORRELATOR

PATH_CONTROL_ID: path control ID for this session
LFSID: local address for this session
TRANSMISSION_PRIORITY: this session's transmission priority

> HS_ID—this field contains the process identifier for the LU-LU half-session process (HS). When the half-session process does not exist, this field is set to a null value.

HS_ID

FQPCID: this session's Fully Qualified PCID control vector
PC_CHARACTERISTICS: see page A-32
SESSION_ID: session instance identifier
PERM_POOL_ADJUSTED_UP: indicates whether the permanent buffer pool was adjusted for this HS.
PERM_POOL_ID: permanent buffer pool ID.
DEM_LIM_POOL_ID: limited buffer pool ID.
DYNAMIC_POOL_ID: dynamic buffer pool ID.

> SENT_BIND_RQ fields are set when a BIND request is sent. A copy of the sent BIND RU is saved because it is needed to perform error checking on the received RSP(BIND).

SENT_BIND_RQ
   SNF: TH sequence number of sent BIND (used to correlate RSP(BIND)
   BIND_RQ_RU: saved BIND RU

> RANDOM holds the random data used for LU-LU verification sent to a partner LU in BIND or RSP(BIND), and the random data received in a RSP(BIND).

RANDOM

CHAPTER 5.0. OVERVIEW OF PRESENTATION SERVICES

GENERAL DESCRIPTION

Presentation services (PS) is the component of the LU with which transaction programs interact directly. Each execution instance of a transaction program at the LU is served by its own PS process. This PS process is responsible for processing the transaction program's requests for LU services. The transaction program requests these services by issuing verbs.

The verbs, along with their supplied (by the user) and returned (by the LU) parameters, are fully described in SNA Transaction Programmer's Reference Manual for LU Type 6.2, which defines both the services that the LU provides and a syntax for transaction program requests for those services. The basic services are SNA-defined and are provided by all LU implementations, but the syntax of requests for the services within individual implementations are implementation-defined.

The services requested by verbs usually involve communication over a conversation with a transaction program at a remote LU. The supplied parameters of a verb therefore usually include an identifier of the conversation for which the verb is being issued. The data exchanged by conversing transaction programs is carried on a session assigned to the conversation.

PS interacts with various other LU components. The LU resources manager (RM) creates PS after receiving an Attach or START_TP record. In addition, RM assigns the half-sessions to PS for conversation traffic, and destroys PS after PS informs RM (via the TERMINATE_PS record) that the PS instance can be destroyed. To carry out transaction program verb requests, PS exchanges data with the half-session that RM has previously assigned for conversation traffic. PS also interacts with the transaction program; in this book, the transaction program and PS are modeled within the same process, which may not be the case in actual implementations. The PS instance is driven by the verbs issued by the transaction program (TP).

Throughout the PS chapters, a number of references are made to LU 6.2 verbs, and LU 6.2 verb parameters. See SNA Transaction Programmer's Reference Manual for LU Type 6.2 for detailed information about the verbs and verb parameters.

PS COMPONENT FUNCTIONS

Figure 5.0-1 on page 5.0-2 shows the components of PS. PS.INITIALIZE loads and calls the TP. The TP then issues verbs, which are processed by the other PS components. The TP ends by returning to PS.INITIALIZE. The functions and interactions of the PS components are further described below.

TP:

- Interacts directly with local end users and resources.
- Requests LU services (for interaction with remote resources) by issuing verbs.

PS.INITIALIZE:

- Receives program initialization parameters (PIP data).
- Loads and calls the TP.
- Instructs RM (after the TP completes and returns) to destroy this PS process.

PS.VERB_ROUTER:

- Checks every verb for compatibility with the type of the conversation on which it was issued.
- Routes valid verb-issuances to the appropriate verb-processing component.

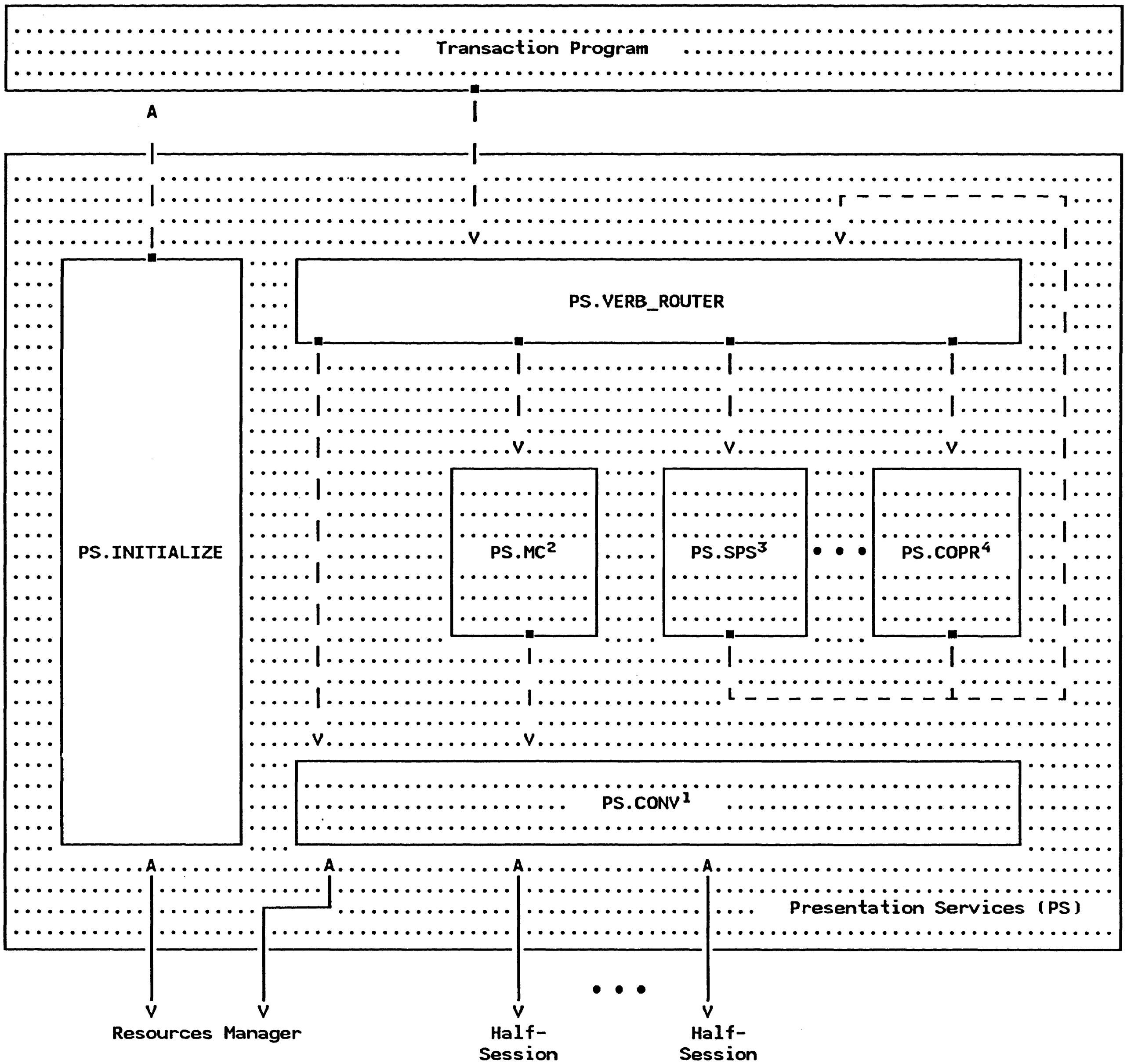
PS.MC, PS.SPS, ..., PS.COPR:

- Process non-basic verbs that request optional special services (these components and their associated services are described in separate chapters of this book).
- Translate non-basic verbs into basic verbs.

PS.CONV:

- Processes basic conversation verbs.
- Checks each basic conversation verb for compatibility with the state of the conversation on which it was issued.
- Performs (in co-operation with, or at the request of, other verb-processing components) all basic conversation services.

All the components of the PS process (including the transaction program execution instance) interact synchronously (using call/return logic). PS may exchange information with other LU components by means of



[1] See "Chapter 5.1. Presentation Services--Conversation Verbs"
[2] See "Chapter 5.2. Presentation Services--Mapped Conversation Verbs"
[3] See "Chapter 5.3. Presentation Services--Sync Point Services Verbs"
[4] See "Chapter 5.4. Presentation Services--Control-Operator Verbs"

Note: A dashed line denotes a synchronous (i.e., a Call) protocol boundary between components, while a solid line denotes an asynchronous (i.e., a Send) protocol boundary.

Figure 5.0-1. Overview of Presentation Services, Emphasizing PS.INITIALIZE and PS.VERB_ROUTER

asynchronous interprocess communication (using send/receive logic).

DATA BASE STRUCTURE

PS uses several data structures to record information needed to provide services to the transaction program. These data structures include PS_PROCESS_DATA, the transaction control block (TCB), and the resource control block list (RCB_LIST). This chapter describes the use of these data structures by the PS.INITIALIZE and PS.VERB_ROUTER components. Use of data structures by other PS components is described in detail in the corresponding chapters.

PS_PROCESS_DATA on page 5.0-24 contains data that is accessible by all components of the PS process. This data includes pointers to lists of shared control blocks, and to single control blocks describing the local LU and this PS process. These pointers are initialized using information contained in the PS_CREATE_PARMS data structure passed from RM when it creates the PS process, and they remain unchanged thereafter.

The transaction control block (TCB, page A-9) contains information specific to the transaction program instance, such as the list of resources allocated to it, the security user ID (see SNA Formats for additional information) carried in the Attach, and the security profile (see SNA Formats for additional information) optionally carried in the Attach. The TCB also contains the CONTROLLING_COMPONENT field, which is maintained by PS.VERB_ROUTER, and records whether the verb was issued by the TP or by a verb-processing component (on behalf of the TP). The TCB is created by RM when the PS process is created and destroyed by RM when the PS process is destroyed.

The resource control block (RCB, page A-6) contains information specific to a particular resource, such as the state of a conversation or the conversation type. One RCB exists for each active resource (e.g., for each active conversation). The RCB is created and destroyed by RM at the request of PS as part of the processing of the ALLOCATE and DEALLOCATE verbs. Certain fields of the RCB are shared between PS and RM, while other fields are used exclusively by PS.

```
      Half-               Resources                               Transaction
      Session             Manager          PS.INITIALIZE            Program

          .                   .                    .                   .
          .     Attach        .                    .                   .
 1        o------------------>o                    .                   .
          .                   . (create PS process) .                  .
 2        .                   o- - - - - - - - - ->o                   .
          .                   MU containing FMH-5(Attach)
          .                   . and, optionally, data .                .
 3        .                   o------------------->o                   .
          .                   .                    .                   .
          .                   MU                   .                   .
 4     [  o--------------------------------------->o ]                 .
          .                   .                    .                   .
          .                   .                    .                   .
          .                   .                  Call TP(RCB_ID, PIP1, ..., PIPn)
 5        .                   .                    o- - - - - - - - - ->o
          .                   .                    .                   .
          .                   .                    .                   .
          .                   .                    .             (TP executes)
          .                   .                    .                   .
          .                   .                    .     (return)      .
 6        .                   .                    o<- - - - - - - - - o
          .                   .                    .                   .
          .                [  .  DEALLOCATE_RCB    .  ]                .
          .                   o<-------------------o                   .
          .                   .                    .                   .
          .                   .  RCB_DEALLOCATED   .                   .
          .                   o------------------->o                   .
          .                [                          ]                .
          .                   .                    .                   .
          .                   .   TERMINATE_PS     .                   .
 7        .                   o<-------------------o                   .
          .                   .                    .                   .
          .                   . (destroy PS process) .                 .
          .                   o- - - - - - - - - ->o                   .
```

Figure 5.0-2. Attach Initialization and Termination of Presentation Services and Transaction Program

INITIALIZATION AND TERMINATION (PS.INITIALIZE)

The PS.INITIALIZE component performs initialization and termination of PS and the TP. Initialization occurs in response to receipt of an Attach from another LU, or to a locally initiated start-up request. These are discussed individually in the following sections.

Processing an FMH-5(Attach) Request

Figure 5.0-2 on page 5.0-3 shows the protocol boundary flows that are used by PS.INITIALIZE for initialization and termination of the PS process when the TP is invoked because of receipt of an FMH-5(Attach). The steps below correspond to the numbers in the figure.

1. Resources manager receives an Attach for a transaction program known locally by the LU.

2. RM creates the PS process, passing it initialization parameters contained within the PS_CREATE_PARMS (see page A-27) structure, including the LUCB_LIST_PTR, the TCB_LIST_PTR and the RCB_LIST_PTR. These parameters are used to initialize the PS_PROCESS_DATA structure.

3. PS next receives from RM an FMH-5 (Attach), accompanied by the TCB ID of this instance of PS, the RCB ID of the initial conversation (the conversation on which the Attach flowed), and sense data containing the result of RM's checking of the Attach. If the sense data indicates no error was found by RM, PS.INITIALIZE performs additional checking of the Attach. This includes a check of the transaction program's support of the conversation type (e.g., basic or mapped) and program initialization parameters (PIP data). If the Attach is valid and no additional data is contained in the MU, PS.INITIALIZE calls the buffer manager to free the MU buffer. If the Attach is in error (as determined by RM or PS.INITIALIZE) the conversation requested in the Attach is terminated. Depending on the error detected, the session may be deactivated, or the conversation ended with DEALLOCATE TYPE(ABEND_PROG).

4. The Attach indicates whether PIP data follows. If the Attach is correct, the PIP data (if any) is received as a single GDS variable, and is then separated into a list of individual PIP subfields. This flow will occur if PIP data is present and the data cannot be contained in the MU containing the FMH-5(Attach).

5. An execution instance of the transaction program named in the Attach is then created. This TP is called with arguments of the RCB ID of the initial conversation and the list of PIP subfields (if present).
6. When the TP completes processing (normally or abnormally), it returns to PS.INITIALIZE. PS.INITIALIZE terminates and deallocates (in an implementation-dependent way) the TP's remaining active conversations (if any; the list of conversations that are still active is found in the RESOURCES_LIST of the TCB).

7. Finally, PS.INITIALIZE sends a TERMINATE_PS (see page A-17) record to the resources manager and waits to be terminated. On receipt of the TERMINATE_PS record, RM destroys the PS process.

```
 START_TP              Resources                                     Transaction
  Sender                Manager           PS.INITIALIZE                Program

      .                    .                    .                          .
      .                    .                    .                          .
      .     START_TP       .                    .                          .
1     o------------------->o                    .                          .
      .                    .                    .                          .
      .                    .  (create PS process)                          .
2     .                    o- - - - - - - - - ->o                          .
      .                    .                    .                          .
      .                    START_TP(optional PIP data)                     .
3     .                    o------------------->o                          .
      .                    .                    .                          .
      .                    .                    .   Call TP(null RCB_ID,   .
      .                    .                    .   PIP1, PIP2, ..., PIPn) .
4     .                    .                    o- - - - - - - - - - - - ->o
      .                    .                    .                          .
      .                    .                    .                          .
      .                    .                    .                   (TP executes)
      .                    .                    .                          .
      .                    .                    .         (return)         .
5     .                    .                    o<- - - - - - - - - - - - -o
      .                    .                    .                          .
      .                  [ .   DEALLOCATE_RCB   . ]                        .
      .                  | o<-------------------o |                        .
      .                  | .                    . |                        .
      .                  | .   RCB_DEALLOCATED  . |                        .
      .                  | o------------------->o |                        .
      .                  [ .                    . ]                        .
      .                    .                    .                          .
      .                    .    TERMINATE_PS    .                          .
6     .                    o<-------------------.                          .
      .                    .                    .                          .
      .                    .  (destroy PS process)                         .
      .                    o- - - - - - - - - ->o                          .
```

Figure 5.0-3. START_TP Initialization and Termination of Presentation Services and Transaction Program

Processing a START_TP request

Figure 5.0-3 shows the protocol boundary flows that are used by PS.INITIALIZE for initialization and termination of the PS process when the TP is invoked because of receipt of a START_TP request. The steps below correspond to the numbers in the figure.

1. Resources manager receives a START_TP request and begins processing the record. This processing includes validating the START_TP record, and verifying that the correct number of PIP parameters have been included.

2. RM creates the PS process, passing it initialization parameters contained within the PS_CREATE_PARMS structure, including the LUCB_LIST_PTR, the TCB_LIST_PTR and the RCB_LIST_PTR. These parameters are used to initialize the PS_PROCESS_DATA structure.

3. PS receives the START_TP from RM, accompanied by the TCB ID of this instance of PS, and sense data containing the result of RM's checking of the START_TP. All checking of the START_TP is completed in the resources manager; PS does no additional checking. The START_TP includes the PIP data to be passed to the transaction program, if any is required.

4. PS.INITIALIZE creates an execution instance of the transaction program named in the START_TP record, calls the transaction program, and passes the list of PIP subfields (if present). PS.INITIALIZE passes a null RCB ID to the transaction program because the START_TP request does not have a conversation associated with it.

5. When the TP completes processing (normally or abnormally), it returns to PS.INITIALIZE. PS.INITIALIZE terminates and deallocates (in an implementation-dependent way) the TP's remaining active conversations (if any; the list of conversations that are still

active is found in the RESOURCES_LIST of the TCB).

6. Finally, PS.INITIALIZE sends a TERMINATE_PS (see page A-17) record to the resources manager and waits to be terminated. On receipt of the TERMINATE_PS record, RM destroys the PS process.

```
     Any Node             Resources                                          Transaction
     Process               Manager              PS.INITIALIZE                  Program

        .                     .                       .                           .
        .                     .                       .                           .
        . Attach or START_TP(1) .                     .                           .
1       o-------------------->o                       .                           .
        .                     .                       .                           .
        .         •           . (create PS process[1]) .                          .
2       .         •           o- - - - - - - - - - - ->o                          .
        .         •           .                       .                           .
        .        (2)          .   Attach or START_TP  .                           .
3       .         |           o---------------------->o                           .
        .       (n-1)         .                       .                           .
        .         •           .                       . Call TP(RCB_ID, PIP data) .
4       .         •           .                       o- - - - - - - - - - - - - ->o
        .         •           .                       .           •               .
        .                     .                       .           •        (TP executes)
        . Attach or START_TP(n) .                     .           •
5       o-------------------->o                       .                           .
        .                     .                       .                           .
        .                     . (create PS process[n]) .                          .
6       .                     o- - - - - - - - - - - ->o                          .
        .                     .                       .                           .
        .                     Attach (optional PIP data)                          .
        .                     o---------------------->o                           .
        .                     .                       .                           .
        .                     .                       . Call TP(RCB_ID, PIP data) .
        .                     .                       o- - - - - - - - - - - - - ->o
        . Attach or START_TP(n+1)                     .           •               .
7       o-------------------->o (Instance limit reached;          •        (TP executes)
        .                     .    request queued by  .           •
        .                     .    resources manager) .                           .
```

Figure 5.0-4. Limited-Instance Transaction Program Processing in Resources Manager

LIMITED-INSTANCE TP PROCESSING

Figure 5.0-4 shows the processing that occurs in the resources manager for initialization of the PS processes when a limited-instance TP (one having a limit of n concurrent instances) is invoked because of receipt of an Attach request. The steps below correspond to the numbers in the figure.

1. Resources manager receives an Attach or START_TP request and begins processing the record.

2. RM creates the PS process, passing it initialization parameters contained within the PS_CREATE_PARMS structure, including the LUCB_LIST_PTR, the TCB_LIST_PTR and the RCB_LIST_PTR. These parameters are used to initialize the PS_PROCESS_DATA structure.

3. PS receives an Attach or START_TP record from RM, accompanied by the TCB ID of this instance of PS, and sense data containing the result of RM's checks of the Attach or START_TP. PS continues processing the Attach or START_TP.

4. PS.INITIALIZE creates an execution instance of the transaction program named in the Attach or START_TP record, calls the transaction program, and passes the list of PIP subfields (if present). PS.INITIALIZE calls this TP with the appropriate parameters from the Attach or START_TP.

5. Resources manager may receive additional Attaches or START_TP records for the same transaction program. Processing of additional requests for the transaction program will continue until the instance limit n is reached.

6. RM creates additional instances of PS until the instance limit n has been reached. PS creates and calls (similar to the processing above) additional instance of the transaction program.

7. RM queues additional requests for a transaction program once the instance limit n has been reached. When an executing instance of the transaction program completes its processing, resources manager initiates the oldest queued request.

VERB PROCESSING (PS.VERB_ROUTER)

PS.VERB_ROUTER routes verbs to the appropriate PS verb-processing component. It also processes type-independent conversation verbs such as WAIT and GET_TYPE. The supplied RESOURCE parameter of most verbs identifies the conversation for which the verb is being issued. The value in the RESOURCE parameter matches one in TCB.RESOURCES_LIST, the list of resources allocated to the TP.

PS.VERB_ROUTER also maintains the CONTROLLING_COMPONENT field of the TCB. The value of CONTROLLING_COMPONENT is TP if the verb has been issued directly by the transaction program. The value is SERVICE_COMPONENT if the verb has been issued by another PS component as part of its verb processing.

WAIT Verb Processing

The WAIT verb, unlike most verbs, can be issued for multiple conversations in addition to being issued for a single conversation. It allows a TP to wait until specified conditions are satisfied ("posted") for any of several conversations. WAIT processing includes:

- Checking that all the resource IDs are valid and that at least one resource is eligible for posting
- Determining whether a resource is already posted (and, if one is, returning immediately)
- Awaiting, if no posting condition has been satisfied, the arrival of data that will cause a resource to be posted

GET TYPE Verb Processing

GET_TYPE processing is handled locally in PS.VERB_ROUTER by copying the conversation type from the appropriate RCB into a returned parameter of the verb.

GET TP PROPERTIES Verb Processing

GET_TP_PROPERTIES processing is handled locally in PS.VERB_ROUTER by copying the requested information from the appropriate control blocks into the returned parameters of the verb.

PS

FUNCTION:	Presentation services (PS) provides verb-processing services to a transaction program execution instance (TP). PS invokes, terminates, and is driven by the TP; PS and the TP are parts of the same process.
	This procedure receives an initialization record (MU or START_TP) from the resources manager (RM) and, based on record type, invokes the appropriate procedure. If the initialization record is valid, PS invokes the transaction program named in the record. When the TP returns to PS, PS informs the resources manager that the TP has completed (by calling DEALLOCATION_CLEANUP_PROC), and waits for a subsequent MU or START_TP.
INPUT:	PS_CREATE_PARMS record, and an MU or START_TP from RM
OUTPUT:	Process data is initialized and the appropriate procedure is called to initialize the transaction program.
NOTES:	1. If no additional initiation requests for the same TP (that just terminated) can be passed to a PS instance by RM, RM will destroy the instance upon receiving its TERMINATE_PS record.
	2. If no record is present, PS will be suspended until a record is received.
	3. DEALLOCATION_CLEANUP_PROC sends a TERMINATE_PS record to RM, alerting it to the termination of its TP and the reusability of this PS instance.

Referenced procedures, FSMs, and data structures:

PS_PROCESS_DATA	page 5.0-24
DEALLOCATION_CLEANUP_PROC	page 5.0-18
PROCESS_FMH5	page 5.0-10
PROCESS_START_TP	page 5.0-11
LUCB	page A-1
MU	page A-29
PS_CREATE_PARMS	page A-27
START_TP	page A-19
TCB	page A-9

Establish the PS environment.

```
Initialize the fields of PS_PROCESS_DATA with the values contained in PS_CREATE_PARMS and
 LUCB_PTR so it points to the LUCB for this LU (identified by PS_CREATE_PARMS.LU_ID),
 TCB_PTR so it points to the TCB for this TP (identified by PS_CREATE_PARMS.TCB_ID).
```

PS_INITIALIZE is imbedded in the root procedure in the PS calling tree.

```
Do this processing until destroyed by RM (see Note 1).
   Receive record from RM (FMH-5 or START_TP; see Note 2).
   Select based on record from RM:
      When MU
         Call PROCESS_FMH5(MU) (page 5.0-10).
      When START_TP
         Call PROCESS_START_TP(START_TP) (page 5.0-11).
   Call DEALLOCATION_CLEANUP_PROC (page 5.0-18; see Note 3).
```

During the processing in this chapter, a number of error conditions may be encountered. The following logic executes only if one of the detectable errors listed have been recognized. The following error condition may be detected: • A Cannot-Occur condition (>) that does occur in an FSM

RM page 3-19
ABEND_NOTIFICATION page A-25

Create and initialize an ABEND_NOTIFICATION record indicating PS abended.
Send the ABEND_NOTIFICATION record to RM.

PROCESS_FMH5

FUNCTION: This procedure loads and calls an instance of the transaction program named in a received FMH-5(Attach).

An incoming FMH-5(Attach) request is routed to this procedure to initialize the transaction program. As shown in SNA Formats, the FMH-5(Attach) contains the name of the transaction program to be invoked and an indicator of whether program initialization parameters (PIP data) will accompany the Attach. PROCESS_FMH5 receives the PIP data (if any) from the half-session, and validates fields of the FMH-5(Attach).

If the FMH-5(Attach) is valid, PS invokes the transaction program named in the FMH-5(Attach).

If the FMH-5(Attach) contains an error, ATTACH_ERROR_PROC is called.

INPUT: MU containing an FMH-5(Attach), SENSE_DATA from RM's Attach checks, and possibly PIP data following the Attach

OUTPUT: The attached (loaded) TP is passed the RCB_ID representing the conversation between the attached program and the attaching program, and PIP data, if present.

NOTES:

1. If RM finds the Attach invalid, the Attach is accompanied by sense data (in the MU_WITH_ATTACH) with one of the following values:

X'080F6051'	Security not valid
X'10086000'	FMH length not correct
X'10086005'	Access Security Information field length invalid
X'10086009'	Invalid parameter length
X'1008600B'	Unrecognized FMH command
X'10086011'	LUW length invalid
X'10086021'	TPN not recognized
X'10086040'	Invalid Attach parameter
X'084B6031'	Transaction program not available—retry
X'084C0000'	Transaction program not available—no retry
X'10086040'	Sync level not supported by LU
X'10086041'	Sync level not supported by TP

Otherwise, RM's sense data in the MU_WITH_ATTACH = X'00000000'.

2. As an alternative to invoking the transaction program immediately upon receipt of the FMH-5(Attach), PS may optionally await the receipt of data indicating end-of-chain before dispatching the transaction program. If the end-of-chain indicator cannot be received in a single pacing window, then buffer management and pacing would force the TP to be started. Once started, an additional pacing window can be used to send more data.

Referenced procedures, FSMs, and data structures:

ATTACH_ERROR_PROC	page 5.0-15
INITIALIZE_ATTACHED_RCB	page 5.0-20
PS_ATTACH_CHECK	page 5.0-12
PS_PIP_CHECKS	page 5.0-13
RECEIVE_PIP_FIELD_FROM_HS	page 5.0-12
UPM_EXECUTE	page 5.0-22
TCB	page A-9
RCB	page A-6
MU_WITH_ATTACH, see MU	page A-29
CODE, see SENSE_DATA	page 5.0-25

Set CODE to the Attach check CODE passed up from RM.

Find the RCB for the conversation identified by the RCB_ID parameter.
Call INITIALIZE_ATTACHED_RCB(RCB, MU_WITH_ATTACH) (page 5.0-20).

Put the MU_WITH_ATTACH in the front of the RCB.HS_TO_PS_BUFFER_LIST.

```
If CODE indicates a valid Attach then (continue with PS Attach checks)
   Call PS_ATTACH_CHECK(RCB, CODE) (page 5.0-12).
   If PIP data is expected for the TP then (PIP data follows Attach)
      Call RECEIVE_PIP_FIELD_FROM_HS(RCB, Attach PIP data, CODE) (page 5.0-12).
   Call PS_PIP_CHECKS(Attach PIP data, CODE) (page 5.0-13).

If CODE indicates a valid Attach then
   Call UPM_EXECUTE(TCB.TRANSACTION_PROGRAM_NAME, RCB.RCB_ID, Attach PIP data)
    (page 5.0-22; see Note 2).
Else (error with the Attach)
   Call ATTACH_ERROR_PROC(RCB, CODE) (page 5.0-15).
```

PROCESS_START_TP

FUNCTION:	This procedure loads and calls an instance of the transaction program named in a received START_TP. A START_TP request is routed to this procedure to complete the necessary processing to initialize the transaction program. The START_TP contains the name of the transaction program to be invoked, and any program initialization parameters (PIP) data.
INPUT:	START_TP information and the TCB.TRANSACTION_PROGRAM_NAME (from PS_PROCESS_DATA)
OUTPUT:	The TP is passed the PIP data, if present. The START_TP record is destroyed.

Referenced procedures, FSMs, and data structures:

UPM_EXECUTE	page 5.0-22
START_TP	page A-19
TCB	page A-9

Save PIP data from START_TP to pass to UPM_EXECUTE.
Destroy the START_TP record.
Call UPM_EXECUTE(TCB.TRANSACTION_PROGRAM_NAME, null RCB_ID, saved PIP data) (page 5.0-22).

RECEIVE_PIP_FIELD_FROM_HS

FUNCTION:	During invocation of the transaction program, this procedure receives a program initialization parameters (PIP data) by issuing a RECEIVE_AND_WAIT verb. If this verb issuance succeeds in receiving a complete logical record containing a PIP Data GDS variable, the received PIP field is returned. If it fails, a protocol violation has been committed by the partner LU; the session is deactivated and the transaction program is not invoked.
INPUT:	The RCB for the TP's initial conversation, the PIP Data GDS variable from the half-session, and the current status of the Attach processing contained in CODE
OUTPUT:	PIP data and the value of CODE indicating if a protocol error has occurred
NOTE:	This error occurs if the partner indicates in the Attach that PIP data follows, but no data follows, or the data that follows is not PIP data, or the PIP data field was truncated.

Referenced procedures, FSMs, and data structures:

RECEIVE_AND_TEST_POSTING	page 5.1-50
RCB	page A-6
CODE, see SENSE_DATA	page 5.0-25

Create, initialize, and issue a RECEIVE_AND_WAIT on this conversation with:
RECEIVE_AND_WAIT.POST_CONDITIONS.FILL set to LL.
RECEIVE_AND_WAIT.POST_CONDITIONS.MAX_LENGTH set to X'7FFF'.

Call RECEIVE_AND_TEST_POSTING(RCB, RECEIVE_AND_WAIT verb parameters) (page 5.1-50) to get the PIP data to pass to the TP.

If the DATA parameter of the RECEIVE_AND_WAIT verb contains the complete PIP data (see SNA Formats for format) then
  Return the Attach PIP data.
Else (Error with PIP data; see Note)
  Set CODE to X'1008201D'.

PS_ATTACH_CHECK

FUNCTION:	This procedure validates additional fields of the received Attach. These additional checks are performed only if the Attach checks in RM did not detect an error.
INPUT:	Attach information (from RM), the current value of CODE (X'00000000'), and program initialization parameter (PIP) data from HS.
OUTPUT:	CODE remains X'00000000' if no invalid fields are found; otherwise, the appropriate sense data. If all the data is exhausted from the MU containing the Attach, this procedure calls the buffer manager to free the MU buffer.

Referenced procedures, FSMs, and data structures:

RCB	page A-6
TCB	page A-9
MU	page A-29
CODE, see SENSE_DATA	page 5.0-25

Get the first MU from the RCB.HS_TO_PS_BUFFER_LIST.
Set the TRANSACTION_PROGRAM_NAME to the entry in the TCB that is indicated on the Attach.
Select, in order, based on the following conditions (of Attach fields):

Errors that cause the session to be deactivated

When the Logical Unit of Work Identifier fields are incorrectly specified (see SNA Formats for proper format)
  Set CODE to X'10086011'.

Errors that cause an FMH-7 to be generated

When the transaction program does not support the conversation type (Basic or Mapped) specified in the Attach
  Set CODE to X'10086034'.

Otherwise (no problems detected).
  Do nothing.

All data in the MU has been processed, so free the MU buffer.

If all the data in the MU has been processed then
  Save the type field (end-of-chain type) from the MU buffer.
  Call buffer manager (FREE_BUFFER, buffer address).

PS_PIP_CHECKS

FUNCTION:	This procedure checks if PIP data was required with the Attach and whether the number of PIP parameters expected matches the number sent, or if PIP data was sent and should not have been sent. If a previous error was detected with the Attach (CODE ¬= X'00000000'), that value is returned and the PIP checks are not performed.
INPUT:	The received PIP data and the current value of code
OUTPUT:	If the current value of CODE is not X'00000000', the error code is returned without making the PIP checks; otherwise, the PIP checks are performed with CODE remaining X'00000000' if no errors are found, or updated to the appropriate sense data in the case of error.

Referenced procedures, FSMs, and data structures:

TCB	page A-9
TRANSACTION_PROGRAM	page A-5
CODE, see SENSE_DATA	page 5.0-25

Set the TRANSACTION_PROGRAM_NAME to the entry in the TCB that is indicated on the Attach.
Select, in order, based on the following conditions:

Errors that cause the session to be deactivated

```
When CODE indicates a previously detected error
   Keep the value of CODE unchanged.
```

Errors that cause an FMH-7 to be generated

```
When the TP is configured to require checking the number of PIP fields
   If TRANSACTION_PROGRAM.NUMBER_OF_PIP_SUBFIELDS is 0 and Attach
    indicates PIP data is present then
      Set CODE to X'10086031' (PIP not allowed).
   Else (specific number of PIP fields expected)
      If TRANSACTION_PROGRAM.NUMBER_OF_PIP_SUBFIELDS differs from the number in
       received PIP data, or the Attach indicates PIP data not present then
         Set CODE to X'10086032' (wrong number of PIP fields).
      Else
         If the format of PIP data is invalid then (see SNA Formats for format)
            Set CODE to X'1008201D' (PIP format invalid--protocol violation).

When the TP is not configured to require checking the number of PIP fields
   If the format of PIP data is invalid then (see SNA Formats for format)
      Set CODE to X'1008201D' (PIP format invalid--protocol violation).

Otherwise (no problems detected).
   Do nothing.
```

ATTACH_ERROR_PROC

FUNCTION:	This procedure handles the processing required when an invalid FMH-5 (Attach) is received. Depending upon the type of Attach error (as reflected in the passed SENSE_CODE parameter), PS either generates an (FMH-7, CEB) or causes the session over which the Attach flowed to be deactivated. When the Attach contains an error that violates defined protocols, PS sends a request to RM indicating that the session is to be deactivated. For all other Attach errors, PS first issues a SEND_ERROR record to the half-session. PS then creates an FMH-7 error message that contains sense data identifying the type of Attach error encountered. END_CONVERSATION_PROC is called to instruct RM to terminate the conversation and the PS process.
INPUT:	The RCB corresponding to the conversation over which the invalid Attach was received, and the sense data specifying the type of Attach error
OUTPUT:	The session is deactivated or an FMH-7 error message is sent to the half-session and the conversation is ended. (Error data is optionally logged and sent with the FMH-7.)

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
PS_PROTOCOL_ERROR	page 5.0-20
GET_END_CHAIN_FROM_HS	page 5.1-36
SEND_ERROR_TO_HS_PROC	page 5.1-58
UPM_ATTACH_LOG	page 5.0-22
END_CONVERSATION_PROC	page 5.1-34
SEND_DATA_BUFFER_MANAGEMENT	page 5.1-54
RCB	page A-6
MU	page A-29
CODE, see SENSE_DATA	page 5.0-25

```
Select based on the value of CODE:
   When X'1008200E', X'10086000', X'10086005', X'10086009', X'10086011', X'10086040',
    X'1008201D' (Deactivate the session)
      Call PS_PROTOCOL_ERROR(RCB.HS_ID, CODE) (page 5.0-20).
      Call END_CONVERSATION_PROC(RCB) (page 5.1-34).
   Otherwise (Generate an FMH-7)
      Call SEND_ERROR_TO_HS_PROC(RCB) (page 5.1-58).
      Call GET_END_CHAIN_FROM_HS(RCB) (page 5.1-36).
      Select based on the end-of-chain type received:
         When DEALLOCATE_FLUSH
            Log the error in system error log.
         When DEALLOCATE_CONFIRM, CONFIRM, PREPARE_TO_RCV_CONFIRM, PREPARE_TO_RCV_FLUSH
            Call UPM_ATTACH_LOG(CODE, LOG_DATA) (page 5.0-22) to generate log data
             describing the detected Attach error.
            If the log data is non-null then
               Log error in local system error log.
               Put into the send MU an FMH-7 (see SNA Formats for format) indicating that
                log data follows and sense data (from CODE) is included.
               Call SEND_DATA_BUFFER_MANAGEMENT(Error log GDS variable, RCB) (page 5.1-54).
                (See SNA Formats for Error log GDS format.)
            Else
               Put into the SEND_MU an FMH-7 (see SNA Formats for format) indicating that
                no log data follows and sense data (from CODE) is included.
            Set MU.PS_TO_HS.TYPE to DEALLOCATE_FLUSH and send the MU record to HS.
      Call END_CONVERSATION_PROC(RCB) (page 5.1-34).
```

PS_VERB_ROUTER

FUNCTION:	This procedure receives all verbs issued by the TP and routes them to the appropriate PS component (e.g., basic conversation verbs to PS.CONV, and control-operator verbs to PS.COPR) for processing.
INPUT:	The current transaction program verb
OUTPUT:	The RESOURCE parameter of the verb is checked to see if it is valid before proceeding with additional processing. The return code of the TRANSACTION_PROGRAM_VERB may be updated to OK or to indicate a detected error. Also, the CONTROLLING_COMPONENT is updated to indicate TP or SERVICE_COMPONENT. Refer to the PS components that are called from this process for the specific outputs.
NOTES:	1. As a general rule, basic verbs must be issued on basic conversations. This check enforces that rule; however, there are some exceptions. Non-basic verb-processing components reside above PS.CONV (see Figure 5.0-1 on page 5.0-2), and typically issue basic conversation verbs in carrying out the functions of non-basic verbs. When the TP issues a mapped conversation verb, PS.VERB_ROUTER routes the verb to PS.MC. PS.MC begins processing the verb, and then, in general, issues one or more basic conversation verbs, which are processed by PS.CONV. As an alternative, for performance reasons, mapped verbs can be routed directly to the proper PS.CONV procedure to avoid additional procedure calls.
	2. If the TP issues a verb that is incompatible with the specified resource, such as a mapped conversation verb specifying a basic conversation, the TP has committed a programming error. PS_VERB_ROUTER informs the TP of the error by means of a return code.

Referenced procedures, FSMs, and data structures:

PS_CONV	page 5.1-10
GET_TP_PROPERTIES_PROC	page 5.0-18
WAIT_PROC	page 5.0-19
PS_MC	page 5.2-20
PS_COPR	page 5.4-32
PS_SPS	page 5.3-35
RCB	page A-6
TCB	page A-9

Select based on type of the TP verb issued:

Verbs Processed by Presentation Services for Conversations

```
When ALLOCATE
   Call PS_CONV(verb parameters)   (page 5.1-10).
When CONFIRM, CONFIRMED, DEALLOCATE, FLUSH, GET_ATTRIBUTES, POST_ON_RECEIPT,
 PREPARE_TO_RECEIVE, RECEIVE_AND_WAIT, RECEIVE_IMMEDIATE, REQUEST_TO_SEND,
 SEND_DATA, SEND_ERROR, or TEST
   If the supplied RESOURCE parameter of the verb identifies a conversation assigned
    to this transaction (i.e., occurs in TCB.RESOURCES_LIST), then
      Find the RCB for the conversation identified by the supplied RESOURCE parameter.
      If the RCB.CONVERSATION_TYPE is BASIC_CONVERSATION or
       (the RCB.CONVERSATION_TYPE is MAPPED_CONVERSATION and
       TCB.CONTROLLING_COMPONENT is SERVICE_COMPONENT) (see Note 1) then
         Call PS_CONV(verb parameters) (page 5.1-10).
      Else (see Note 2)
         Set the RETURN_CODE of the verb to PROGRAM_PARAMETER_CHECK.
```

Verbs Processed by Presentation Services for Mapped Conversations

```
When MC_ALLOCATE
   Call PS_MC(verb parameters) (page 5.2-20).
When MC_CONFIRM, MC_CONFIRMED, MC_DEALLOCATE, MC_FLUSH, MC_GET_ATTRIBUTES,
 MC_POST_ON_RECEIPT, MC_PREPARE_TO_RECEIVE, MC_RECEIVE_AND_WAIT,
 MC_REQUEST_TO_SEND, MC_SEND_DATA, MC_SEND_ERROR, or MC_TEST
   If the verb's supplied RESOURCE parameter identifies a conversation assigned
    to this transaction (i.e., occurs on TCB.RESOURCES_LIST), then
      Find the RCB for the conversation identified by RESOURCE.
      If RCB.CONVERSATION_TYPE is MAPPED_CONVERSATION,
       then (it should be, because this verb is a mapped verb)
         Set TCB.CONTROLLING_COMPONENT to SERVICE_COMPONENT.
         Call PS_MC(verb parameters) (page 5.2-20).
         Set TCB.CONTROLLING_COMPONENT back to TP.
      Else (see Note 2)
         Set the RETURN_CODE of the verb to PROGRAM_PARAMETER_CHECK.
   Else
      Set the RETURN_CODE of the verb to PROGRAM_PARAMETER_CHECK.
```

Verbs Processed by Presentation Services for the Control Operator

```
When INITIALIZE_SESSION_LIMIT, CHANGE_SESSION_LIMIT, RESET_SESSION_LIMIT, SET_LUCB,
 SET_PARTNER_LU, SET_MODE, SET_MODE_OPTION, SET_TRANSACTION_PROGRAM,
 SET_PRIVILEGED_FUNCTION, SET_RESOURCE_SUPPORTED, SET_SYNC_LEVEL_SUPPORTED,
 SET_MC_FUNCTION_SUPPORTED_TP, GET_LUCB, GET_PARTNER_LU, GET_MODE, GET_LU_OPTION,
 GET_MODE_OPTION, GET_TRANSACTION_PROGRAM, GET_PRIVILEGED_FUNCTION, GET_RESOURCE_SUPPORTED,
 GET_SYNC_LEVEL_SUPPORTED, GET_MC_FUNCTION_SUPPORTED_LU, GET_MC_FUNCTION_SUPPORTED_TP,
 LIST_PARTNER_LU, LIST_MODE, LIST_LU_OPTION, LIST_MODE_OPTION, LIST_TRANSACTION_PROGRAM,
 LIST_PRIVILEGED_FUNCTION, LIST_RESOURCE_SUPPORTED, LIST_SYNC_LEVEL_SUPPORTED,
 LIST_MC_FUNCTION_SUPPORTED_LU, LIST_MC_FUNCTION_SUPPORTED_TP, PROCESS_SESSION_LIMIT,
 ACTIVATE_SESSION, or DEACTIVATE_SESSION
   Set TCB.CONTROLLING_COMPONENT to SERVICE_COMPONENT.
   Call PS_COPR(verb parameters) (page 5.4-32).
   Set TCB.CONTROLLING_COMPONENT  back to TP.
```

Type-Independent Conversation Verbs

```
When SYNCPT or BACKOUT
   Set TCB.CONTROLLING_COMPONENT to SERVICE_COMPONENT.
   Call PS_SPS (page 5.3-35).
   Set TCB.CONTROLLING_COMPONENT to TP.

When GET_TP_PROPERTIES
   Call GET_TP_PROPERTIES_PROC(verb parameters) (page 5.0-18).

When GET_TYPE
   Set the RETURN_CODE of the GET_TYPE verb to OK.
   If the verb's supplied RESOURCE parameter identifies a conversation
    assigned to this transaction then
      Find the RCB for the conversation identified by RESOURCE.
      Copy RCB.CONVERSATION_TYPE into the verb's returned TYPE parameter.
   Else
      Set the RETURN_CODE of the GET_TYPE verb to PROGRAM_PARAMETER_CHECK.

When WAIT
   Set TCB.CONTROLLING_COMPONENT to SERVICE_COMPONENT.
   Call WAIT_PROC(verb parameters)  (page 5.0-19).
   Set TCB.CONTROLLING_COMPONENT back to TP.
```

DEALLOCATION_CLEANUP_PROC

FUNCTION:	This procedure, which manages the destruction of this process, is invoked after the TP has ended. It calls UPM_RETURN_PROCESSING on page 5.0-23 to deallocate the process's remaining conversations, and sends DEALLOCATE_RCB to RM to get rid of RCBs and any other resources allocated to the process. Finally, it sends a TERMINATE_PS record to RM.
INPUT:	TCB.RESOURCES_LIST (from PS_PROCESS_DATA)
OUTPUT:	DEALLOCATE_RCB and TERMINATE_PS to RM

Referenced procedures, FSMs, and data structures:

RM	page 3-19
UPM_RETURN_PROCESSING	page 5.0-23
FSM_CONVERSATION	page 5.1-65
TCB	page A-9
RCB	page A-6
TERMINATE_PS	page A-17

```
Do for each RCB ID on the TCB.RESOURCES_LIST.
   Find the RCB for the conversation identified in the RCB ID parameter.
   If the state of FSM_CONVERSATION is not RESET or is not END_CONV then
      Call UPM_RETURN_PROCESSING(RCB ID) (page 5.0-23).

Send a TERMINATE_PS record to RM.
```

GET_TP_PROPERTIES_PROC

FUNCTION:	This procedure handles requests for information about a transaction program. Information about the transaction program is retrieved from the appropriate control blocks and placed in the returned parameters of the GET_TP_PROPERTIES verb.
INPUT:	The TCB, LUCB (from PS_PROCESS_DATA), and the GET_TP_PROPERTIES verb
OUTPUT:	GET_TP_PROPERTIES verb's returned parameters containing information about the transaction program

Referenced procedures, FSMs, and data structures:

LUCB	page A-1
TCB	page A-9

```
Set the GET_TP_PROPERTIES returned parameters as follows:
   OWN_TP_NAME to TCB.TRANSACTION_PROGRAM_NAME,
   OWN_TP_INSTANCE to TCB.TCB_ID,
   OWN_FULLY_QUALIFIED_LU_NAME to LUCB.FULLY_QUALIFIED_LU_NAME,
   SECURITY_PROFILE to TCB.INITIATING_SECURITY.PROFILE,
   SECURITY_USER_ID to TCB.INITIATING_SECURITY.USERID,
   the RETURN_CODE of the GET_TP_PROPERTIES verb to OK.
```

WAIT_PROC

FUNCTION:	This procedure processes WAIT verbs. First, it validates the resources specified in the verb's RESOURCE_LIST parameter. While checking this list, this procedure creates a sublist of it called TEMP_RESOURCE_LIST. This sublist contains only those resources from RESOURCE_LIST that are currently activated for posting. Activated for posting means the TP has issued a POST_ON_RECEIPT on the conversation (or resource) and the posting has not been satisfied. (If none of the resources specified in the supplied RESOURCE_LIST parameter is activated for posting, this procedure sets the RETURN_CODE field of the WAIT to POSTING_NOT_ACTIVE.) After creating the TEMP_RESOURCE_LIST, this procedure next checks to see if any of the resources in the list have already been posted. If none of the resources has been posted, this procedure waits for one of the resources to become posted.
INPUT:	WAIT verb parameters, incoming conversation data
OUTPUT:	The verb's returned parameters are set as follows. RETURN_CODE indicates whether the WAIT completed successfully. If the verb completed successfully, RESOURCE_POSTED indicates which resource has been posted.

Referenced procedures, FSMs, and data structures:

TEST_FOR_RESOURCE_POSTED	page 5.0-21
RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
TCB	page A-9
RCB	page A-6

Check that all resources in the supplied RESOURCE_LIST parameter are validly allocated to this transaction program (i.e., occur in TCB.RESOURCES_LIST), and that at least one of them has posting active.
If any resource is invalid then
   Set the RETURN_CODE of the WAIT verb to PROGRAM_PARAMETER_CHECK, and return.
If no resource has been activated for posting then
   Set the RETURN_CODE of the WAIT verb to POSTING_NOT_ACTIVE, and return.

> At this point (since all resources are valid and one or more have "posting active"), it is safe to wait for a resource to become posted. If some resource is already posted, there is no need to wait.

For each resource that has posting active,
   Call TEST_FOR_RESOURCE_POSTED(RCB, RC) (page 5.0-21)
   If the returned RC is not UNSUCCESSFUL then
      Set the RETURN_CODE of the WAIT verb to RC, and return.

> Since no active resource is posted yet, wait until one is.

Initialize RC to UNSUCCESSFUL.
Do While RC remains UNSUCCESSFUL.
   Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(TEMP_RESOURCE_LIST containing list of RCB_IDs) (see FUNCTION above for details; page 5.1-51).
   Set RCB to the RCB for the conversation on which the data has arrived.
   Call TEST_FOR_RESOURCE_POSTED(RCB, RC) (page 5.0-21).
Set the returned RESOURCE_POSTED parameter to RCB.RCB_ID.
Set the RETURN_CODE of the WAIT verb to the returned value RC.

PS_PROTOCOL_ERROR

FUNCTION:	This procedure processes receive error conditions that require the session to be deactivated. An UNBIND_PROTOCOL_ERROR record is sent to the resources manager to request deactivation of the session that committed the protocol violation.
INPUT:	The HS ID for the half-session that committed the protocol violation, the TCB_ID (from PS_PROCESS_DATA), and the sense data to be sent on the UNBIND
OUTPUT:	UNBIND_PROTOCOL_ERROR (to RM) with the TCB ID for this PS process and the input HS ID and sense data

Referenced procedures, FSMs, and data structures:

RM	page 3-19
PS_PROCESS_DATA	page 5.0-24
HS_ID	page 3-91
SENSE_CODE, see SENSE_DATA	page 5.0-25
UNBIND_PROTOCOL_ERROR	page A-17

```
Initialize an UNBIND_PROTOCOL_ERROR record (page A-17)
 with this TCB_ID, HS_ID, and SENSE_CODE.
Send the UNBIND_PROTOCOL_ERROR to RM.
```

INITIALIZE_ATTACHED_RCB

FUNCTION:	This procedure initializes the PS.CONV specific fields in the RCB for the resource specified in the received Attach. The shared fields in the RCB are initialized by RM. This procedure is invoked when RM forwards Attach information to PS.
INPUT:	RCB of the conversation and Attach information (received from RM)
OUTPUT:	The PS.CONV specific fields in the specified RCB are initialized. RM initializes all the shared data in the RCB. The states of FSM_CONVERSATION, FSM_POST, and FSM_ERROR_OR_FAILURE are initialized to the proper states. The receive buffer, HS_TO_PS_BUFFER_LIST of the RCB, is purged. If the conversation is MAPPED, the MAPPED fields of the RCB are initialized.

Referenced procedures, FSMs, and data structures:

RCB	page A-6

Initialize the RCB fields as follows:
CONVERSATION_TYPE to the TYPE value in the Attach,
LIMITED_BUFFER_POOL_ID to the ID value in the Attach,
PERMANENT_BUFFER_POOL_ID to the ID value in the Attach,
SEND_RU_SIZE to the SIZE value in the Attach,
POST_CONDITIONS.FILL to LL,
POST_CONDITIONS.MAX_LENGTH to 0,
LOCKS to SHORT,
RQ_TO_SEND_RCVD to NO.
Purge the receive buffer (RCB.HS_TO_PS_BUFFER_LIST).

Initialize the states of the FSMs as follows:
FSM_CONVERSATION to RCV,
FSM_ERROR_OR_FAILURE to NO_REQUESTS,
FSM_POST to RESET.

If RCB.CONVERSATION_TYPE is MAPPED_CONVERSATION then
Purge the MAPPED receive buffer (RCB.MC_RECEIVE_BUFFER).
Initialize the RCB fields as follows:
MC_RQ_TO_SEND_RCVD to NO,
MAPPER_SAVE_AREA to an implementation-defined value,
MC_MAX_SEND_SIZE according to a implementation-defined algorithm.

TEST_FOR_RESOURCE_POSTED

FUNCTION:	This procedure determines if the resource corresponding to the passed RCB has been posted. Depending on the type of conversation indicated by the RCB, this procedure issues either a TEST (TEST = POSTED) or an MC_TEST (TEST = POSTED) verb, which is processed by PS.CONV ("Chapter 5.1. Presentation Services--Conversation Verbs") or PS.MC ("Chapter 5.2. Presentation Services--Mapped Conversation Verbs"), respectively. The RETURN_CODE field in the returned verb indicates whether posting has occurred for the specified conversation.
INPUT:	The entry in the RCB_LIST corresponding to the resource for which this procedure is to determine if posting has occurred
OUTPUT:	The return code returned for the issued TEST or MC_TEST verb

Referenced procedures, FSMs, and data structures:

TEST_PROC	page 5.1-26
MC_TEST_PROC	page 5.2-28
RCB	page A-6

Select based on RCB.CONVERSATION_TYPE:
When basic
Create TEST record and initialize as follows:
RESOURCE is RCB.RCB_ID, TEST is POSTED.
Call TEST_PROC(TEST verb parameters) (page 5.1-26) to test posting.
When mapped
Create MC_TEST record and initialize as follows:
RESOURCE is RCB.RCB_ID, TEST is POSTED.
Call MC_TEST_PROC(MC_TEST verb parameters) (page 5.2-28) to test posting.
Return to the caller the RETURN_CODE from the TEST or MC_TEST verb.

UPM_EXECUTE

FUNCTION:	This UPM loads and executes a transaction program.
INPUT:	The name of the transaction program, the resource ID (to be passed to the transaction program), and a list of PIP data (to be passed to the transaction program)
OUTPUT:	None

Not defined by SNA

UPM_ATTACH_LOG

FUNCTION:	This UPM is invoked upon discovery of an error in an FMH-5 (Attach). It returns log data describing the error. This data is logged in the local system error log and is sent back to the conversation partner in an Error-Log GDS variable accompanying an FMH-7.
INPUT:	Attach error sense data
OUTPUT:	Log data (may be null)

Not defined by SNA

UPM_RETURN_PROCESSING

FUNCTION:	This UPM is invoked when a TP ends and returns to PS without having deallocated all its resources. It terminates and deallocates a remaining active resource in an implementation-specific way. Two of the many ways in which an implementation could do this are to: • Issue DEALLOCATE TYPE(ABEND_SVC) for the still-allocated resource. • Issue DEALLOCATE TYPE(SYNC_LEVEL) if the resource is in SEND state and data in PS's send buffer is on a logical record boundary. If the attempt to synchronize fails, or the data was not on a logical record boundary, then issue DEALLOCATE TYPE(ABEND_SVC). Regardless of what other actions are taken, this UPM causes FSM_CONVERSATION (page 5.1-65) to enter the reset state.
INPUT:	The RCB_ID of the still-allocated resource
OUTPUT:	See above.

Not defined by SNA

LOCAL DATA STRUCTURES

PS_PROCESS_DATA
PS_PROCESS_DATA is available to all procedures in the presentation services process. The structure is initialized by the PS process (page 5.0-8) and remains unchanged for the lifetime of the PS process.

```
PS_PROCESS_DATA
   LUCB_LIST_PTR:  Pointer to the LUCB_LIST
   LU_ID:          ID of this PS's LU
   LUCB_PTR:       Pointer to the LUCB for this PS's LU
   TCB_LIST_PTR:   Pointer to the TCB_LIST
   TCB_ID:         ID of this PS
   TCB_PTR:        Pointer to the TCB for this PS
   RCB_LIST_PTR:   Pointer to the RCB_LIST of this PS
```

TCB_LIST_PTR

TCB_LIST_PTR: Pointer to the list of TCBs for TP-PS processes at this LU.

RCB_LIST_PTR

RCB_LIST_PTR: Pointer to the RCB_LIST for this TP-PS process.

LUCB_LIST_PTR

LUCB_LIST_PTR: Pointer to the list of LUCBs for LUs known to this LU.

SENSE_DATA

SENSE_DATA: 4-byte sense data

This page intentionally left blank

CHAPTER 5.1. PRESENTATION SERVICES--CONVERSATION VERBS

GENERAL DESCRIPTION

A PS process handles requests for LU services. A transaction program (TP) execution instance makes these requests by issuing verbs. The verbs are divided into categories, and PS is divided into components. Each verb-processing component of PS processes the verbs of one specific category. Presentation services for basic conversations (PS.CONV) is the component of PS that processes verbs of the basic conversation category. Figure 5.1-1 on page 5.1-2 provides an overview of PS, showing the relationship of PS.CONV to the other PS components.

The basic conversation verbs correspond to the most basic services provided by the LU. Other PS components, such as PS.MC ("Chapter 5.2. Presentation Services--Mapped Conversation Verbs") and PS.COPR ("Chapter 5.4. Presentation Services--Control-Operator Verbs") use basic conversation verbs in providing their higher-level functions. "Open-API" implementations may choose to expose only the mapped conversation verbs to user-application transaction programming, while leaving the lower-level basic conversation verbs "closed."

See Chapter 5.0 for an overview of PS and its components, and of the relationship of PS to the other components of the LU. Refer to SNA Transaction Programmer's Reference Manual for LU Type 6.2 for a complete description of the basic conversation verbs.

PS.CONV FUNCTIONS

The functions of PS.CONV include:

- Requesting the allocation and deallocation of conversation resources.
- Maintaining and checking the basic conversation state.
- Transferring conversation data between the half-session and transaction program variables.
- Tracking logical record lengths.

COMPONENT INTERACTIONS

PS.CONV interacts with PS.VERB_ROUTER ("Chapter 5.0. Overview of Presentation Services"), the resources manager ("Chapter 3. LU Resources Manager"), and one or more half-session components ("Chapter 6.0. Half-Session").

All verb service requests are routed through PS.VERB_ROUTER, which forwards basic conversation verbs to PS.CONV. After PS.CONV has performed the requested service, control is returned to the caller, with updated values in those variables that are the verb's returned parameters, or in which it requested a result to be returned.

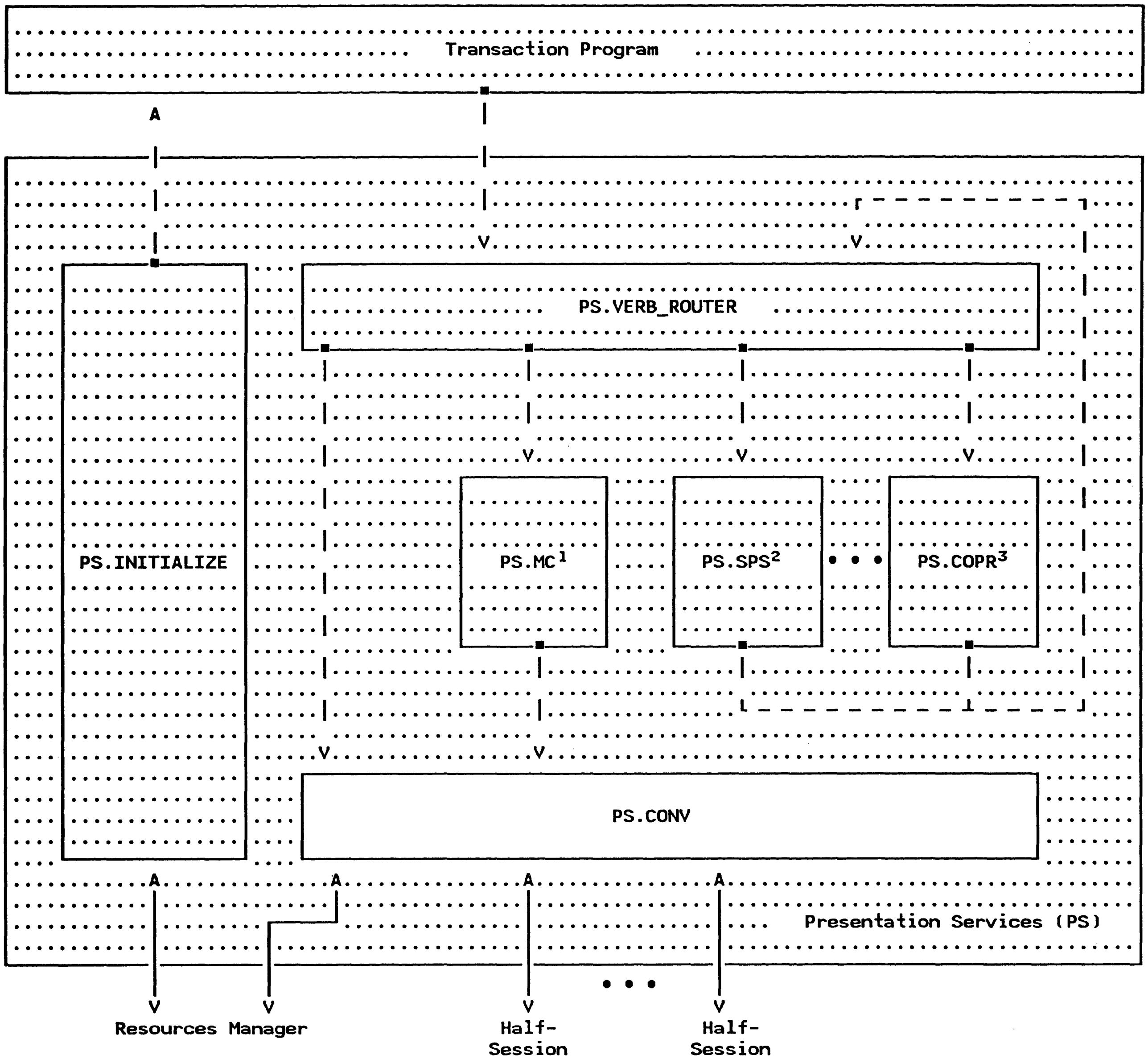
PS.CONV interacts with the resources manager (RM) to request allocation and deallocation of LU resources, such as conversations and associated control blocks, and to report protocol errors. Since PS.CONV and RM are modeled as different processes, this interaction occurs by means of asynchronous interprocess communication (send/receive logic). RM also informs PS.CONV if a conversation being used by PS.CONV fails for some reason.

PS.CONV interacts with one half-session process for each active conversation used by PS.CONV. Each half-session serves a single conversation at a time. Since the TP may have active conversations with several partners simultaneously, PS.CONV may be interacting with a number of different half-session processes.

PS.CONV interacts with the buffer manager to get and free storage used to send and receive data. PS.CONV requests a buffer (for an MU) from the buffer manager when data must be sent. PS.CONV copies the data from the TP verb into the buffer before passing the buffer to HS for sending out of the LU. When PS.CONV is passed a buffer (holding an inbound MU) from HS, it passes the data as requested to the TP. When all the data has been passed to the TP, PS.CONV requests the buffer manager to free that buffer storage.

PS.CONV DATA BASE STRUCTURE

PS.CONV uses a number of control blocks and data structures. The most important ones are described here. See "Appendix A. Node Data Structures" for full details.



[1] See "Chapter 5.2. Presentation Services--Mapped Conversation Verbs"
[2] See "Chapter 5.3. Presentation Services--Sync Point Services Verbs"
[3] See "Chapter 5.4. Presentation Services--Control-Operator Verbs"

Note: A dashed line denotes a synchronous (call/return) protocol boundary between components, while a solid line denotes an asynchronous (send/receive) protocol boundary.

Figure 5.1-1. Overview of Presentation Services, Emphasizing Presentation Services for Basic Conversations

<u>LU Control Block (LUCB) and Associated Lists</u>

The LU control block (LUCB--see Figure 5.1-2 on page 5.1-3) is used by PS.CONV. One LUCB exists for each LU in the node. The LUCB is identified by the LU ID, which is a unique identifier for each LU in the node. Each LUCB contains information such as the network-qualified LU name.

Associated with each LUCB is a TRANSACTION_PROGRAM_LIST. The TRANSACTION_PROGRAM_LIST for an LU contains an entry for each transaction program known locally by the LU. The information in a TRANSACTION_PROGRAM_LIST entry includes the transaction program name and whether it supports various optional features (e.g., sync point, mapped conversations).

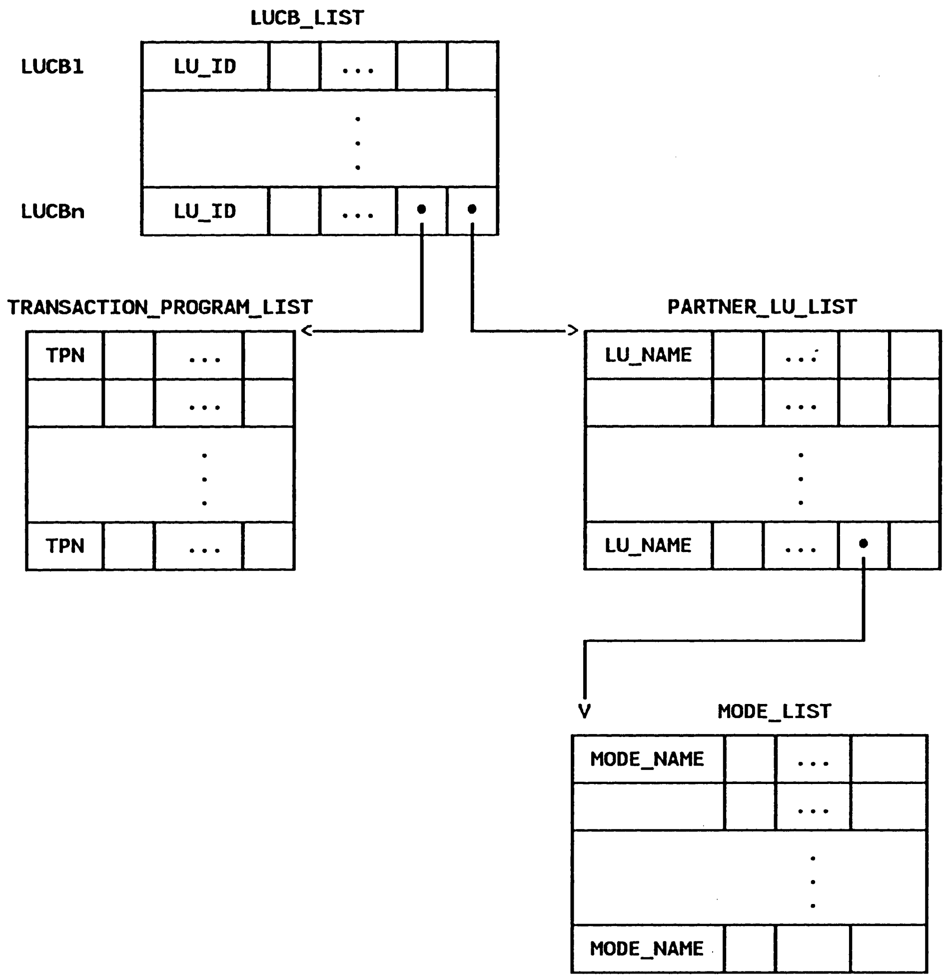


Figure 5.1-2. LU Control Block List and Associated Lists

Another list associated with each LUCB is the PARTNER_LU_LIST (see Figure 5.1-2 on page 5.1-3). The PARTNER_LU_LIST contains one entry for each potential partner LU of the LU represented by the LUCB. The PARTNER_LU entry contains information that is fixed for the specific partner LU, such as the local and network-qualified names of the partner LU.

Associated with each PARTNER_LU entry is a MODE_LIST (see Figure 5.1-2), which has one entry for each mode name that is defined for possible use with the particular partner LU name. The MODE entry contains information that is fixed on a mode basis, such as the mode name and maximum mode session limit.

Transaction Control Block (TCB)

The transaction control block (TCB--see Figure 5.1-3 on page 5.1-4) contains information associated with the combined TP-PS process. One TCB exists for each TP-PS process. Each TCB contains a TCB ID, which is a unique identifier of the TP-PS process being represented by the TCB. The TCB ID is used in all communication between the resources manager and the PS serving the transaction program. For example, when PS sends a record to the resources manager, it provides its TCB ID so that the resources manager will know, of all the TP-PS processes it manages, which one to send a reply to.

Associated with each TCB is the RESOURCES_LIST, a list of the resources used by the TP-PS process. The RESOURCES_LIST has one entry for each resource (e.g., for each conversation) associated with the transaction program.

PS PROCESS DATA

PS_PROCESS_DATA (page 5.0-24) contains data that is available to all procedures in the PS process. It contains information about this particular TP-PS process, such as the LU ID

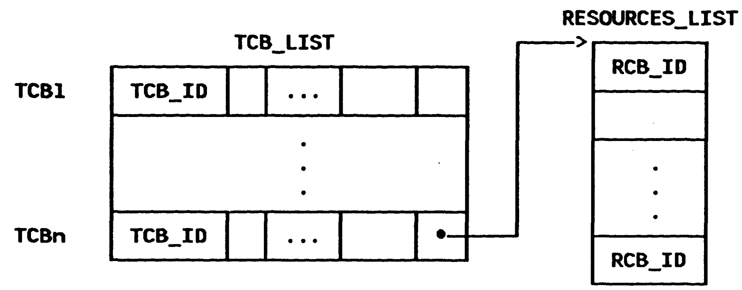


Figure 5.1-3. Transaction Control Block (TCB)

and the pointer to the RCB_LIST. It is initialized by the root procedure of the PS process (page 5.0-8) from parameters received in PS_CREATE_PARMS. PS_CREATE_PARMS is passed to PS from RM when the PS process is created.

Resource Control Block (RCB)

One resource control block (RCB--see Figure 5.1-4 on page 5.1-5) represents each active conversation allocated to a transaction program. The RCBs for all active conversations in an LU are kept in the RCB_LIST. RCBs are added to or removed from the RCB_LIST by the LU resources manager, at the request of PS.CONV. RCBs are also linked to the RESOURCES_LIST for the particular TP-PS process to which they are allocated. The TCB for the process has an associated RESOURCES_LIST which contains a list of RCBs (specified by the RCB_ID). The list of RCBs represents the resources, such as conversations, allocated to the process.

An RCB for a conversation contains information pertaining to a particular conversation, such as its resource ID, state, and characteristics (established when the conversation is allocated). Components of PS will update certain fields of the RCB as the conversation is used.

The RCB is identified by a unique RCB ID. This ID accompanies most transaction program verb issuances (as the RESOURCE parameter) to identify the conversation to which the verb is to be applied. The RCB also contains the TCB ID of its owning TP-PS process, and the HS ID of the local half-session that carries the conversation's data. Other fields associated with the RCB are discussed in more detail below.

FSM_CONVERSATION (page 5.1-65) is a finite-state machine that tracks the state of the conversation associated with the RCB. The state of FSM_CONVERSATION is the state of the conversation from the viewpoint of the local TP. For example, the conversation changes from receive to send state when the transaction program is notified by a WHAT_RECEIVED = SEND from a receive verb. The state of the conversation does not change until PS.CONV has actually notified the transaction program, even though the send indication may have arrived from the half-session sometime earlier.

THE SEND MU (page A-29) is associated with the RCB and is used to store data that has been generated by verb processing (i.e., for the SEND_DATA verb) but that has not yet been sent to the half-session.

FSM_ERROR_OR_FAILURE (page 5.1-67) is a finite-state machine that tracks errors or failures on the conversation associated with the RCB. The state of FSM_ERROR_OR_FAILURE records the receipt of the error or failure (forwarded from RM or HS) until the appropriate notification can be returned to the TP in verb parameters.

FSM_POST (page 5.1-68) is a finite-state machine that tracks the posting condition of a conversation associated with the RCB. The state of FSM_POST records whether the conditions specified to satisfy have been (state POSTED) or have not been (state PEND_POSTED) satisfied.

HS_TO_PS_BUFFER_LIST contains a list of MUs that have been received from the half-session but not yet passed to the transaction program.

SECURITY_SELECT initially contains the type of end-user verification: NONE, SAME, or PGM. This value might be downgraded from PGM to NONE or SAME to NONE (see "Chapter 3. LU Resources Manager" for details of when RM downgrades end-user verification). The Attach is built using the SECURITY_SELECT value.

VERB PARAMETERS

The TP requests LU services by issuing verbs. A verb and its parameters are passed as parameters to PS_CONV (page 5.1-10). The service requested is identified by the verb name and the supplied parameter fields, and some results of the service (along with any other pertinent incoming data) are returned

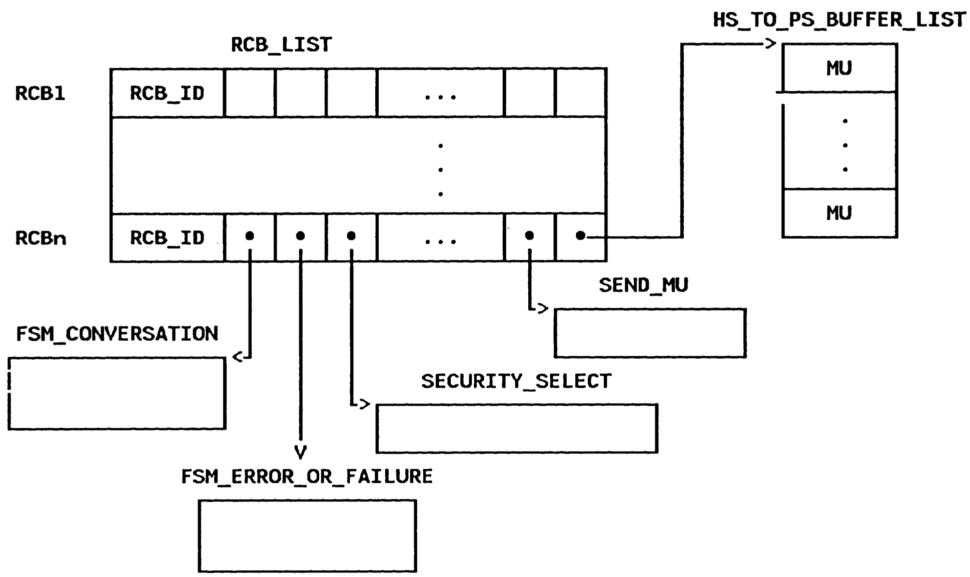


Figure 5.1-4. Resource Control Block (RCB)

to the TP in the returned parameter fields. Each verb issuance has:

- An indicator of which verb is being issued (e.g., ALLOCATE, CONFIRM)
- Some supplied parameters, including (typically) an identifier of the conversation on which the verb is being issued
- Some returned parameters, including (typically) a return code telling whether the requested service was performed successfully

Some examples of exceptions to these parameter rules are the following. ALLOCATE does not supply a conversation ID (although it does return one), while WAIT supplies a list of conversation IDs. CONFIRMED and FLUSH do not need any returned parameters. The basic conversation verbs and their parameters are fully described in SNA Transaction Programmer's Reference Manual for LU Type 6.2.

PS-RM RECORDS

PS.CONV sends records to RM and receives records from RM. PS sends several types of records to RM. Each contains a TCB ID identifying the PS process that sent the record, and possibly additional fields. Records sent from RM to PS are usually sent in reply to a request sent from PS to RM, as shown in Figure 5.1-5.

PS.CONV Request	RM Reply
ALLOCATE_RCB	RCB_ALLOCATED
GET_SESSION	SESSION_ALLOCATED
DEALLOCATE_RCB	RCB_DEALLOCATED

Figure 5.1-5. PS.CONV Requests and Associated RM Replies

The only exception is CONVERSATION_FAILURE, which RM sends unsolicited to PS.CONV when a conversation being used by PS.CONV fails.

PS-HS RECORDS

PS.CONV sends MUs to a half-session and receives MUs and other records from a half-session.

An MU contains a field, identifying the particular type of MU, and additional fields in the case of SEND_DATA_RECORD. SEND_DATA_RECORD is used to send data and RH information to the half-session when the local transaction program is in send state for the conversation. Included in the SEND_DATA_RECORD is the transaction program data to be sent and an encoding of the RH bits (see SNA Formats) that are to be set by the half-session when the data is sent to the remote LU. Data to be sent to a half-session is added to the existing send MU buffer by PS.CONV until the maximum send RU size is exceeded, thus forcing the buffer (SEND_MU) to be passed to the half-session for transmission. The transaction program can also issue a verb that forces the data to be passed to the half-session for transmission (e.g., CONFIRM, RECEIVE_AND_WAIT, or DEALLOCATE).

Other MU types are sent from PS to the half-session only when the local transaction program is in receive state. These include CONFIRMED, used to reply positively to a previous CONFIRM record; REQUEST_TO_SEND, used to request the send indicator from the partner transaction program; and SEND_ERROR, used to send -RSP(0846) to the partner LU.

MU and other records can also be sent from HS to PS, in which case, the BRACKET_ID field in the passed record is used to identify which half-session sent the record to PS.CONV.

TRACKING LOGICAL RECORD LENGTH

Transaction programs using a basic conversation must ensure that the data they exchange is formatted into logical records. The length of a logical record is given by the low-order 15 bits of the first two bytes of the record. The high-order bit is the "continuation bit", which is used for GDS variables by PS.MC (see "Chapter 5.2. Presentation Services--Mapped Conversation Verbs"). The value in the Length field includes the length of the field itself; thus the length value is normally in the range 2-32767. A length value of 0 is invalid while a length value of 1 is used to indicate a PS header (see "Chapter 5.3. Presentation Services--Sync Point Services Verbs" for more details).

When sending data, the transaction program is responsible for correctly setting the Length bytes of each logical record. The amount of data sent by a SEND_DATA verb need have no relation to a logical record; i.e., the transaction program may send partial logical records in SEND_DATA verbs, and may include multiple logical records in one verb.

PS.CONV performs some checking of the logical record Length field supplied by the transaction program. The value of the Length field must be greater than 1, unless TCB.CONTROLLING_COMPONENT = SERVICE_COMPONENT, that is, unless some PS service component (e.g., PS.MC or PS.SPS) is sending a PS header in the record on behalf of a transaction program.

Certain verbs (e.g., CONFIRM) may be validly issued only at logical record boundaries (i.e., between logical records). PS.CONV enforces this rule by remembering how many bytes remain to be sent in the current logical record. SEND_ERROR and DEALLOCATE TYPE(ABEND) are the only verbs that can prematurely truncate a logical record.

PS.CONV also tracks the value of the Length field for logical records received from the partner transaction program. Logical records with a Length of 1 are passed to PS_SPS. PS.CONV maintains a count of the number of bytes remaining in the current logical record. PS.CONV performs an optional receive check to determine if the partner LU has violated PS protocols by allowing the partner transaction program to invalidly truncate the logical record. Only an FMH-7 can validly truncate a logical record.

Finally, when a receive verb is issued specifying FILL(LL), PS uses the receive count remainder (i.e., the number of bytes in the logical record not received) to determine how many bytes of received data to pass to the transaction program.

MAINTAINING AND CHECKING THE BASIC CONVERSATION STATE

PS.CONV maintains the current state of each conversation in FSM_CONVERSATION (page 5.1-65). As noted earlier, the state of FSM_CONVERSATION is the state of the conversation as viewed by the local transaction program.

The state of the conversation may change as a result of verbs issued by the transaction program; e.g., PREPARE_TO_RECEIVE changes the state from send to receive. These inputs have DIRECTION=S in FSM_CONVERSATION. The state may also change as a result of data or indicators received from the half-session; e.g., receiving the send indicator changes the state of the conversation from receive to send. These inputs have DIRECTION=R in FSM_CONVERSATION.

The current state of the conversation determines the verbs that can be validly issued; e.g., a SEND_DATA verb cannot be issued in receive state.

VERB PROCESSING

Details of PS.CONV's processing of some verbs are described here. See also "Chapter 2. Overview of the LU" for more flow diagrams corresponding to the processing of these and other verbs.

Verb Checking

PS.CONV performs a number of send checks on verbs issued by the transaction program. These include:

- Parameter checks, such as checking that:
  - The parameters specified on the ALLOCATE are supported by the LU.
  - The verb conforms to the SYNC_LEVEL of the conversation (as specified on ALLOCATE).
  - The DATA parameter on SEND_DATA contains a valid Length field (see "Tracking Logical Record Length").
- State checks, such as checking that:

- The verb can be issued in the current conversation state (see "Maintaining and Checking the Basic Conversation State" on page 5.1-6).
- The transaction program has completed the current logical record, if necessary (see "Tracking Logical Record Length" on page 5.1-6).

ALLOCATE

Processing of the ALLOCATE verb by PS.CONV includes:

- Requesting that RM allocate a resource control block (RCB).
- Requesting that RM allocate a session for the conversation.
- Creating an Attach FMH-5.

The order of performing the last two items depends on the supplied RETURN_CONTROL parameter of the ALLOCATE verb, as described below.

A conversation resource is represented by a resource control block (RCB--see "PS.CONV Data Base Structure" on page 5.1-1). PS.CONV requests the creation of an RCB by sending an ALLOCATE_RCB record to the resources manager (RM) and waiting for an RCB_ALLOCATED record in reply. If RETURN_CONTROL(IMMEDIATE) is specified, the ALLOCATE_RCB (page A-15) record is a composite request for the creation of an RCB and the allocation of a first-speaker session. This situation is indicated to RM by setting ALLOCATE_RCB.IMMEDIATE_SESSION = YES.

After the RCB has been created, PS.CONV requests the resources manager to allocate a session for use by the conversation (if a session has not already been allocated as a result of IMMEDIATE_SESSION = YES). PS.CONV does this by sending a GET_SESSION record to RM and waiting for a SESSION_ALLOCATED record in reply.

The type of end-user verification is requested in the ALLOCATE as NONE, SAME, or PGM. See <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u> for a more complete description of the security parameter relating to end-user verification.

PS.CONV creates an Attach FMH-5 based on the parameter settings in the ALLOCATE verb and in the RCB. The Attach is stored in the MU buffer, to be sent later. When processing an ALLOCATE verb, the Attach is created after assignment of the session; thus, any security downgrades that are required will have been completed prior to building the Attach in PS.

POST ON RECEIPT

The POST_ON_RECEIPT verb establishes the posting conditions for the conversation. The post conditions (FILL = BUFFER or LL, and LENGTH) are retained in the RCB associated with the conversation. The posting status (reset, pending post, or posted) of a conversation is maintained by FSM_POST. Whenever PS.CONV receives data, end-of-chain type (e.g., SEND, CONFIRM), or both from the half-session, the posting conditions are checked, and the state of FSM_POST is updated if necessary. If POST_ON_RECEIPT has been issued, the state of FSM_POST may be checked by calling TEST_PROC on page 5.1-26. This procedure is used by the WAIT verb to determine whether the post conditions have been satisfied for any of several conversations.

REQUEST TO SEND

When the transaction program issues a REQUEST_TO_SEND verb, PS.CONV checks the conversation state to see if the verb can be validly issued, and checks that the conversation is still active. If so, PS.CONV sends a REQUEST_TO_SEND record to the appropriate half-session process and then waits for a RSP_TO_REQUEST_TO_SEND record from the half-session. By waiting for a response from the half-session before returning to the transaction program, PS.CONV prevents the transaction program from flooding the network with expedited-flow FMD RUs.

On receipt of a REQUEST_TO_SEND record from a half-session (request for send control initialted by partner transaction program), PS.CONV sets RCB.RQ_TO_SEND_RCVD to YES, and notifies the transaction program at the earliest opportunity.

SEND ERROR

Processing of the SEND_ERROR verb by PS.CONV includes:

- If the TP issuing the verb is in receive state:
  - Sending a SEND_ERROR record to the half-session. This causes a -RSP(0846) to be sent to the partner LU.
  - Waiting until an end-of-chain type is received from the partner LU. The half-session purges all data until the end-of-chain type is received.
- If the TP issuing the verb is in send state:
  - Creating an FMH-7 with the sense data based on the SEND_ERROR type and the current state of the conversation. A -RSP(0846) is not sent when the conversation is in send state as done with the conversation when it is in receive state.
  - Creating an Error Log GDS variable (see <u>SNA Formats</u> for format), if log data is present.

If both sides of a conversation issue SEND_ERROR, the side that was in receive state always wins the SEND_ERROR race and obtains send control of the conversation. Figure 5.1-6 on page 5.1-8 shows a flow diagram for a simple SEND_ERROR race.

Figure 5.1-7 on page 5.1-9 shows a SEND_ERROR race with deallocation. In this case, neither error gets reported to the other side. This problem could be avoided by following the SEND_ERROR with a PREPARE_TO_RECEIVE, as shown in the previous figure.

On receipt of a RECEIVE_ERROR record from the half-session (as a result of the partner LU sending a -RSP[0846]), PS.CONV sends an end-of-chain type to the half-session, if it has not already done so. It then receives the expected FMH-7 and notifies the transaction program, at the earliest opportunity, with a return code based on the FMH-7 sense data. If it receives an Error Log GDS variable appended to the FMH-7, the LU logs the data without passing it on to the transaction program.

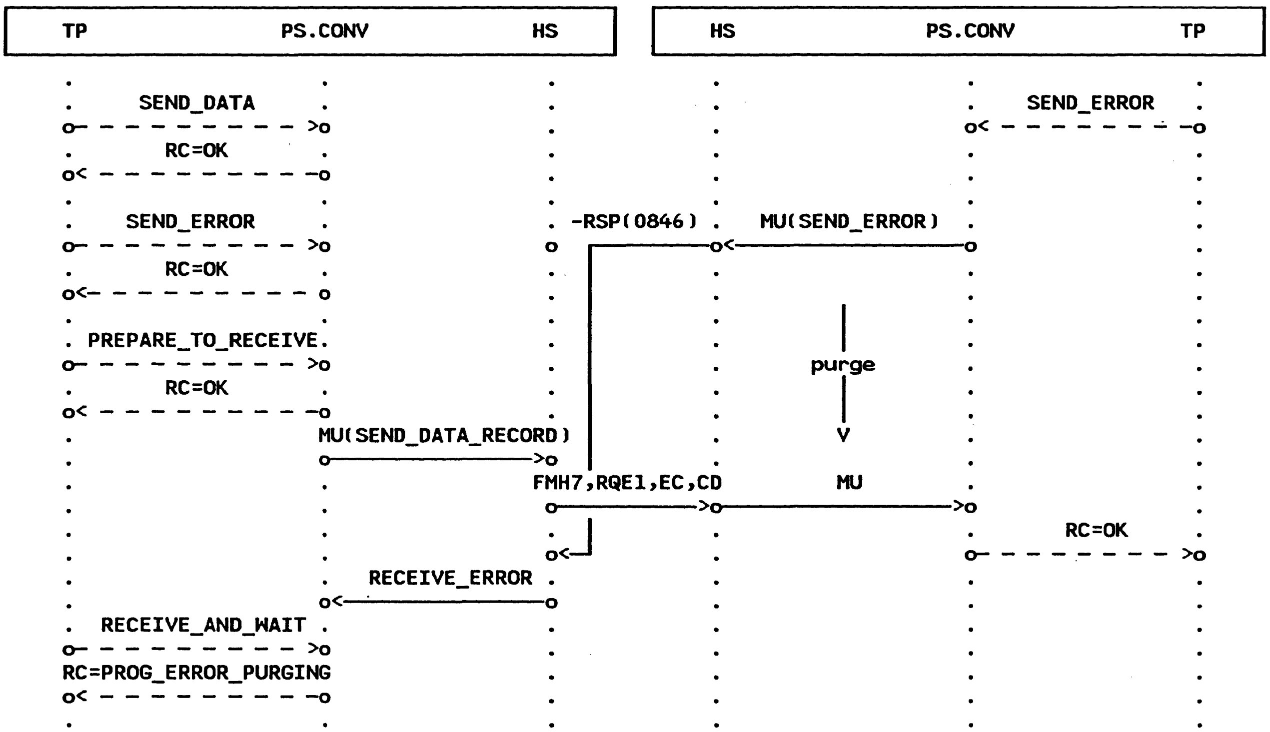


Figure 5.1-6. SEND_ERROR Race

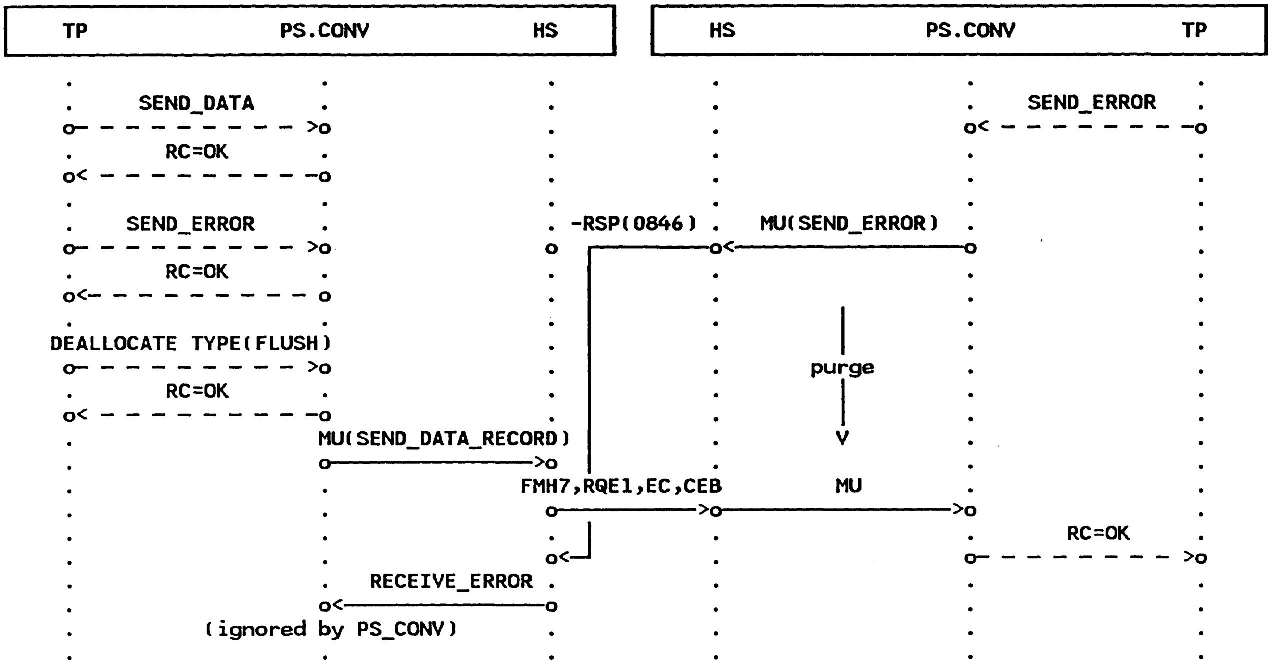


Figure 5.1-7. SEND_ERROR Race with Deallocation

PROTOCOL ERRORS

PS.CONV contains a number of optional receive checks to determine if the partner LU has violated SNA-defined protocols. Examples of protocol violations checked by PS.CONV include:

- Sending data when in receive state
- Invalidly truncating a logical record (see "Tracking Logical Record Length" on page 5.1-6)
- Sending an incorrectly formatted FMH-7

When PS.CONV detects a protocol error, it requests that RM deactivate the session and sets FSM_ERROR_OR_FAILURE (see page 5.1-67) to indicate that a conversation failure (protocol error) occurred. PS.CONV notifies the transaction program of the conversation failure by returning a RESOURCE_FAILURE return code on the next verb that allows a RESOURCE_FAILURE return code.

CONVERSATION FAILURES

PS.CONV is notified of a conversation failure by the CONVERSATION_FAILURE record, sent by RM. The conversation failure may result from either session outage or a protocol violation.

On receipt of a CONVERSATION_FAILURE (see page A-21) record, PS.CONV sets FSM_ERROR_OR_FAILURE to indicate either CONV_FAILURE_SON or CONV_FAILURE_PROTOCOL_ERROR. PS.CONV notifies the transaction program of the conversation failure by returning a RESOURCE_FAILURE return code on the next verb that allows a RESOURCE_FAILURE return code.

PS_CONV

FUNCTION:	This procedure receives conversation verbs issued by the TP or by other PS components, and calls the appropriate procedures to process them.
INPUT:	Transaction program verb and parameters
OUTPUT:	Refer to the procedures that are called from this procedure for the specific outputs.

Referenced procedures, FSMs, and data structures:


```
Select based on the transaction program verb:
   When ALLOCATE
      Call ALLOCATE_PROC(ALLOCATE verb parameters) (page 5.1-11).
   When CONFIRM
      Call CONFIRM_PROC(CONFIRM verb parameters) (page 5.1-12).
   When CONFIRMED
      Call CONFIRMED_PROC(CONFIRMED verb parameters) (page 5.1-14).
   When DEALLOCATE
      Call DEALLOCATE_PROC(DEALLOCATE verb parameters) (page 5.1-15).
   When FLUSH
      Call FLUSH_PROC(FLUSH verb parameters) (page 5.1-16).
   When GET_ATTRIBUTES
      Call GET_ATTRIBUTES_PROC(GET_ATTRIBUTES verb parameters) (page 5.1-17).
   When POST_ON_RECEIPT
      Call POST_ON_RECEIPT_PROC(POST_ON_RECEIPT verb parameters) (page 5.1-17).
   When PREPARE_TO_RECEIVE
      Call PREPARE_TO_RECEIVE_PROC(PREPARE_TO_RECEIVE verb parameters) (page 5.1-18).
   When RECEIVE_AND_WAIT
      Call RECEIVE_AND_WAIT_PROC(RECEIVE_AND_WAIT verb parameters) (page 5.1-19).
   When RECEIVE_IMMEDIATE
      Call RECEIVE_IMMEDIATE_PROC(RECEIVE_IMMEDIATE verb parameters) (page 5.1-21).
   When REQUEST_TO_SEND
      Call REQUEST_TO_SEND_PROC(REQUEST_TO_SEND verb parameters) (page 5.1-23).
   When SEND_DATA
      Call SEND_DATA_PROC(SEND_DATA verb parameters) (page 5.1-24).
   When SEND_ERROR
      Call SEND_ERROR_PROC(SEND_ERROR verb parameters) (page 5.1-25).
   When TEST
      Call TEST_PROC(TEST verb parameters) (page 5.1-26).
```

ALLOCATE_PROC

FUNCTION:	This procedure handles allocation of new resources to the transaction program. If the ALLOCATE parameters are valid, this procedure requests that RM create a new resource control block (RCB). If the supplied RETURN_CONTROL parameter specifies IMMEDIATE, PS at this time also requests RM to acquire a session for use by the conversation resource. If the RETURN_CONTROL is set to WHEN_SESSION_ALLOCATED, PS sends a separate GET_SESSION request to RM at a later time.
INPUT:	ALLOCATE verb with parameters; RCB_ALLOCATED record received from RM
OUTPUT:	The ALLOCATE_RCB record is initialized and sent to RM and the RCB_ALLOCATED record (from RM) is destroyed. If an error is found in the ALLOCATE, the return code is updated.
NOTE:	If the ALLOCATE specifies SECURITY(SAME), but the original conversation was not initialized with security, then the security is downgraded to NONE on the ALLOCATE_RCB before sending the request to the resources manager.

Referenced procedures, FSMs, and data structures:

PS	page 5.0-8
RM	page 3-19
RCB_ALLOCATED_PROC	page 5.1-48
WAIT_FOR_RM_REPLY	page 5.1-62
ALLOCATE_RCB	page A-15
MODE	page A-3
RCB_ALLOCATED	page A-21


```
Check ALLOCATE parameters for validity (see ALLOCATE verb in
 SNA Transaction Programmer's Reference Manual for LU Type 6.2).

If ALLOCATE parameters are valid then
   If MODE control block exists then
      Create and initialize ALLOCATE_RCB request record with the
       parameters of the ALLOCATE.
      SEND ALLOCATE_RCB request to RM.
      Call WAIT_FOR_RM_REPLY to receive RCB_ALLOCATED from RM (page 5.1-62).
      Call RCB_ALLOCATED_PROC(RCB_ALLOCATED, ALLOCATE)(page 5.1-48).
   Else
      Set the RETURN_CODE of the ALLOCATE verb to PARAMETER_ERROR.
Else
   Set the RETURN_CODE of the ALLOCATE verb to PROGRAM_PARAMETER_CHECK.
```

CONFIRM_PROC

FUNCTION:	This procedure handles the CONFIRM verb processing. If it is appropriate for the transaction program to issue a CONFIRM for the specified conversation (i.e., the SYNC_LEVEL of the conversation for which the CONFIRM was issued is CONFIRM or SYNCPT and any data issued by the transaction program is on a logical record boundary), this procedure retrieves any records from HS and RM. Appropriate action is taken depending upon which, if any, record was received (as reflected by the state of FSM_ERROR_OR_FAILURE).
INPUT:	CONFIRM verb parameters
OUTPUT:	An RQ_TO_SEND_RCVD indication could be passed up to the TP at this time if the RCB.RQ_TO_SEND_RCVD field has been set to show receipt of a RQ_TO_SEND. The RCB.RQ_TO_SEND_RCVD field is reset to NO. The return code of the CONFIRM verb is updated. See Notes for additional outputs.
NOTES:	1. If a CONVERSATION_FAILURE has been received from the resources manager, PS returns to the transaction program after setting the RETURN_CODE parameter of the CONFIRM to RESOURCE_FAILURE. 2. If a RECEIVE_ERROR has been received from HS, PS sends a SEND_DATA record with the MU.PS_TO_HS.TYPE field set to PREPARE_TO_RCV_FLUSH to HS. (Any data in the RCB send buffer was purged when the RECEIVE_ERROR record was received.) If an MU is not present, one is created and initialized prior to sending to HS. PS then waits for the expected FMH-7 error message to arrive. The RETURN_CODE parameter of the CONFIRM is set based on the sense data carried in the FMH-7. 3. If there are no error or failure conditions, COMPLETE_CONFIRM_PROC is called to complete the processing of the CONFIRM.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
COMPLETE_CONFIRM_PROC	page 5.1-28
CREATE_AND_INIT_LIMITED_MU	page 5.1-30
DEQUEUE_FMH7_PROC	page 5.1-34
RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
MU	page A-29
RCB	page A-6

```
Find the RCB for the conversation identified in the RESOURCE parameter.
If RCB.SYNC_LEVEL is NONE or the send data is not at a logical record boundary then
   If RCB.SYNC_LEVEL is NONE then
      Set the RETURN_CODE of the CONFIRM verb to PROGRAM_PARAMETER_CHECK.
   Else (not on logical record boundary)
      Set the RETURN_CODE of the CONFIRM verb to PROGRAM_STATE_CHECK.
Else
   If executing FSM_CONVERSATION(S, CONFIRM, RCB) (page 5.1-65) would cause
    a state-check (>) condition then
      Set the RETURN_CODE of the CONFIRM verb to PROGRAM_STATE_CHECK.
   Else
      Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).

      Select based on the state of FSM_ERROR_OR_FAILURE (page 5.1-67):
         When CONV_FAILURE_PROTOCOL_ERROR (see Note 1)
            Set the RETURN_CODE of the CONFIRM verb to RESOURCE_FAILURE_NO_RETRY.
            Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
         When CONV_FAILURE_SON (see Note 1)
            Set the RETURN_CODE of the CONFIRM verb to RESOURCE_FAILURE_RETRY.
            Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
```

```
        When RCVD_ERROR (see Note 2)
           If a send MU buffer does not exist then
              Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
           Set MU.PS_TO_HS.TYPE to PREPARE_TO_RCV_FLUSH and send the MU record to HS.
           Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(SUSPEND_LIST containing RCB_ID) (page 5.1-51).
           If state of FSM_ERROR_OR_FAILURE (page 5.1-67) is CONV_FAILURE_SON or
            CONV_FAILURE_PROTOCOL_ERROR then
              If state of FSM_ERROR_OR_FAILURE (page 5.1-67) is CONV_FAILURE_SON then
                 Set the RETURN_CODE of the CONFIRM verb to RESOURCE_FAILURE_RETRY.
              Else
                 Set the RETURN_CODE of the CONFIRM verb to RESOURCE_FAILURE_NO_RETRY.
              Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
           Else
              Call DEQUEUE_FMH7_PROC(CONFIRM verb parameters, RCB) (page 5.1-34).
        When NO_REQUESTS (see Note 3)
           Call COMPLETE_CONFIRM_PROC(CONFIRM verb parameters, RCB) (page 5.1-28).
     Set the REQUEST_TO_SEND_RECEIVED of the CONFIRM verb to RCB.RQ_TO_SEND_RCVD.
     Set RCB.REQUEST_TO_SEND_RECEIVED to NO.
```

CONFIRMED_PROC

FUNCTION:		This procedure handles CONFIRMED verb processing. PS first retrieves any records from HS and RM. Appropriate action is taken depending upon which, if any, record was received.
INPUT:		CONFIRMED verb parameters
OUTPUT:		The return code of the CONFIRMED verb is set. The states of FSM_CONVERSATION and FSM_ERROR_OR_FAILURE may change. See Notes for additional outputs.
NOTES:	1.	If a CONVERSATION_FAILURE record has been received from the resources manager, PS returns to the transaction program without sending any data to HS. Since CONFIRMED verb does not support a RESOURCE_FAILURE return code, the conversation failure cannot be reported to the transaction program at this time. PS remembers the failure (via FSM_ERROR_OR_FAILURE) and reports it to the transaction program at a later time (i.e., when the transaction program issues a verb that supports a RESOURCE_FAILURE return code).
	2.	If a CONVERSATION_FAILURE record has not been received, PS sends a CONFIRMED record to HS.

Referenced procedures, FSMs, and data structures:

PS_PROTOCOL_ERROR	page 5.0-20
SEND_CONFIRMED_PROC	page 5.1-53
RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
RCB	page A-6


```
Find the RCB for the conversation identified in the RESOURCE parameter.
If executing FSM_CONVERSATION(S, CONFIRMED, RCB) (page 5.1-65) would
 cause a state-check (>) condition then
   Set the RETURN_CODE of the CONFIRMED verb to PROGRAM_STATE_CHECK.
Else
   Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).
   Select based on the state of FSM_ERROR_OR_FAILURE (page 5.1-67):
      When NO_REQUESTS (see Note 2)
         Call SEND_CONFIRMED_PROC(RCB) (page 5.1-53).
      When RCVD_ERROR
         Call PS_PROTOCOL_ERROR (RCB.HS_ID, X'10010000') (page 5.0-20).
      When CONV_FAILURE_PROTOCOL_ERROR or CONV_FAILURE_SON  (see Note 1)
         Do nothing.

   Call FSM_CONVERSATION(S, CONFIRMED, RCB) (page 5.1-65).
   Set the RETURN_CODE of the CONFIRMED verb to OK.
```

DEALLOCATE_PROC

FUNCTION:	This procedure handles the deallocation of resources. If the resource specified in the DEALLOCATE is a valid resource and the conversation is in a pertinent state, PS calls the appropriate deallocation procedure to continue processing the DEALLOCATE.
INPUT:	DEALLOCATE verb parameters
OUTPUT:	The return code of the DEALLOCATE is set here or in one of the called procedures, and FSM_CONVERSATION may change states. Also, the pertinent deallocation procedure is called. When appropriate, PS sends DEALLOCATE_RCB to RM.

Referenced procedures, FSMs, and data structures:

DEALLOCATE_ABEND_PROC	page 5.1-31
DEALLOCATE_CONFIRM_PROC	page 5.1-32
DEALLOCATE_FLUSH_PROC	page 5.1-33
END_CONVERSATION_PROC	page 5.1-34
FSM_CONVERSATION	page 5.1-65
RCB	page A-6

```
Find the RCB for the conversation identified in the RESOURCE parameter.
Select based on the following conditions:
   When the TYPE parameter of DEALLOCATE is FLUSH, or the TYPE parameter is SYNC_LEVEL
    and RCB.SYNC_LEVEL is NONE
      If executing FSM_CONVERSATION(S, DEALLOCATE_FLUSH, RCB) (page 5.1-65) would
       cause a state-check (>) condition then
         Set the RETURN_CODE of the DEALLOCATE verb to PROGRAM_STATE_CHECK.
      Else
         Call DEALLOCATE_FLUSH_PROC(DEALLOCATE verb parameters, RCB) (page 5.1-33).
   When the TYPE parameter is  CONFIRM
      If executing FSM_CONVERSATION(S, DEALLOCATE_CONFIRM, RCB) (page 5.1-65) would
       cause a state-check (>) condition then
         Set the RETURN_CODE of the DEALLOCATE verb to PROGRAM_STATE_CHECK.
      Else
         If RCB.SYNC_LEVEL is CONFIRM or SYNCPT then
            Call DEALLOCATE_CONFIRM_PROC(DEALLOCATE verb parameters, RCB) (page 5.1-32).
         Else
            Set the RETURN_CODE of the DEALLOCATE verb to PROGRAM_PARAMETER_CHECK.
   When the TYPE parameter is SYNC_LEVEL and RCB.SYNC_LEVEL is CONFIRM
      If executing FSM_CONVERSATION(S, DEALLOCATE_CONFIRM, RCB) (page 5.1-65) would
       cause a state-check (>) condition then
         Set the RETURN_CODE of the DEALLOCATE verb to PROGRAM_STATE_CHECK.
      Else
         Call DEALLOCATE_CONFIRM_PROC(DEALLOCATE verb parameters, RCB) (page 5.1-32).
   When the TYPE parameter is SYNC_LEVEL and RCB.SYNC_LEVEL is SYNCPT
      If executing FSM_CONVERSATION(S, DEALLOCATE_DEFER, RCB) (page 5.1-65) would
       cause a state-check (>) condition then
         Set the RETURN_CODE of the DEALLOCATE verb to PROGRAM_STATE_CHECK.
      Else
         If the data sent by TP is on a logical record boundary then
            Call FSM_CONVERSATION(S, DEALLOCATE_DEFER, RCB) (page 5.1-65).
            Set the RETURN_CODE of the DEALLOCATE verb to OK.
         Else
            Set the RETURN_CODE of the DEALLOCATE verb to PROGRAM_STATE_CHECK.
   When the TYPE parameter is ABEND_PROG, ABEND_SVC, or ABEND_TIMER
      If executing FSM_CONVERSATION(S, DEALLOCATE_ABEND, RCB) (page 5.1-65) would
       cause a state-check (>) condition then
         Set the RETURN_CODE of the DEALLOCATE verb to PROGRAM_STATE_CHECK.
      Else
         Call DEALLOCATE_ABEND_PROC(DEALLOCATE verb parameters, RCB) (page 5.1-31).
   When the TYPE parameter is LOCAL
      If executing FSM_CONVERSATION(S, DEALLOCATE_LOCAL, RCB) (page 5.1-65) would
       cause a state-check (>) condition then
         Set the RETURN_CODE of the DEALLOCATE verb to PROGRAM_STATE_CHECK.
      Else
         Set the RETURN_CODE of the DEALLOCATE verb to OK.
         Call FSM_CONVERSATION(S, DEALLOCATE_LOCAL, RCB) (page 5.1-65).
         Call END_CONVERSATION_PROC(RCB) (page 5.1-34).
```

FLUSH_PROC

FUNCTION:	This procedure handles the FLUSH verb processing. The procedure first receives records from RM and HS. Appropriate action is taken depending upon the type of the received record as indicated by the FSM_CONVERSATION and FSM_ERROR_OR_FAILURE states.
INPUT:	FLUSH verb parameters, records from RM and HS
OUTPUT:	The send MU.PS_TO_HS.TYPE field is set according to the state of FSM.CONVERSATION, and the return code on the FLUSH verb is set. If a sending MU is present and contains data, the MU will be sent to HS. See Notes for additional outputs.
NOTES:	1. If PS has received a RECEIVE_ERROR from HS, or no error records have been received, PS sends any data remaining in the RCB send buffer to HS with the MU.PS_TO_HS.TYPE field of the SEND_DATA set to FLUSH, PREPARE_TO_RCV_FLUSH, or DEALLOCATE_FLUSH, depending on the state of the conversation. (If a RECEIVE_ERROR was received, any data in PS's send buffer has already been purged.) 2. If FSM_ERROR_OR_FAILURE indicates that a conversation failure has occurred, PS returns to the transaction program without sending any data to HS. Since FLUSH does not support RESOURCE_FAILURE return code, the error cannot be reported to the transaction program at this time. PS remembers the error (via FSM_ERROR_OR_FAILURE) and reports it to the transaction program at a later time (i.e., when PS receives a record from the transaction program that supports a RESOURCE_FAILURE return code).

Referenced procedures, FSMs, and data structures:


```
Find RCB for the conversation identified in the RESOURCE parameter.
If executing FSM_CONVERSATION(S, FLUSH, RCB) (page 5.1-65) would
 cause a state-check (>) condition then
   Set the RETURN_CODE of the FLUSH verb to PROGRAM_STATE_CHECK.
Else
   Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).
   If the state of FSM_ERROR_OR_FAILURE (page 5.1-67) is RCVD_ERROR
    or NO_REQUESTS then  (see Note 1)
      Select based on the state of FSM_CONVERSATION (page 5.1-65):
         When SEND_STATE
            If a send MU buffer exists and the MU contains data then
               Send the MU record to HS.
         When PREP_TO_RCV_DEFER
            If a send MU buffer does not exist then
               Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
            Set MU.PS_TO_HS.TYPE to PREPARE_TO_RCV_FLUSH and send the MU record to HS.
         When DEALL_DEFER
            If a send MU buffer does not exist then
               Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
            Set MU.PS_TO_HS.TYPE to DEALLOCATE_FLUSH and send the MU record to HS.
      If the state of FSM_CONVERSATION is DEALL_DEFER then
         Call END_CONVERSATION_PROC(RCB) (page 5.1-34).
      Call FSM_CONVERSATION(S, FLUSH, RCB) (page 5.1-65).
   Set the RETURN_CODE of the FLUSH verb to OK.
```

GET_ATTRIBUTES_PROC

FUNCTION:	This procedure handles requests for information about a conversation. Information about the conversation resource is retrieved from the pertinent control blocks, and placed in the returned parameters of the GET_ATTRIBUTES verb.
INPUT:	GET_ATTRIBUTES verb parameters
OUTPUT:	GET_ATTRIBUTES verb returned parameters containing information about the conversation

Referenced procedures, FSMs, and data structures:

FSM_CONVERSATION	page 5.1-65
PARTNER_LU	page A-2
RCB	page A-6

```
Find the RCB for the conversation identified in the RESOURCE parameter.
Set the returned parameters of the GET_ATTIBUTES verb as follows:
   PARTNER_FULLY_QUALIFIED_LU_NAME to PARTNER_LU.FULLY_QUALIFIED_LU_NAME,
   PARTNER_LU_NAME to RCB.LU_NAME,
   MODE_NAME to RCB.MODE_NAME,
   SYNC_LEVEL to RCB.SYNC_LEVEL,
   Set the RETURN_CODE of the GET_ATTRIBUTES verb to OK.
Call FSM_CONVERSATION(S, GET_ATTRIBUTES, RCB) (page 5.1-65).
```

POST_ON_RECEIPT_PROC

FUNCTION:	This procedure performs the processing of the POST_ON_RECEIPT verb. This procedure updates FSM_CONVERSATION and FSM_POST, saves the post conditions in the RCB, and retrieves any records originated in RM and HS. The data just received from RM or HS may cause the resource to be posted.
INPUT:	POST_ON_RECEIPT verb parameters
OUTPUT:	The return code of the verb is updated. If the verb is issued in a valid state, then the state of FSM_POST is changed. FSM_CONVERSATION is called but does not change states. Also, POST_CONDITIONS.FILL and POST_CONDITIONS.MAX_LENGTH in the RCB are updated to the posting conditions.

Referenced procedures, FSMs, and data structures:

RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
FSM_CONVERSATION	page 5.1-65
FSM_POST	page 5.1-68
RCB	page A-6

POST_ON_RECEIPT_PROC

```
Find the RCB for the conversation identified in the RESOURCE parameter.
If executing FSM_CONVERSATION(S, POST_ON_RECEIPT, RCB) (page 5.1-65).
 would cause a state-check (>) condition then
   Set the RETURN_CODE of the POST_ON_RECEIPT verb to PROGRAM_STATE_CHECK.
Else
   Call FSM_CONVERSATION(S, POST_ON_RECEIPT, RCB) (page 5.1-65).
   Call FSM_POST (page 5.1-68) and pass it a POST_ON_RECEIPT signal.
   Set RCB.POST_CONDITIONS.FILL to the FILL parameters of the POST_ON_RECEIPT verb.
   Set RCB.POST_CONDITIONS.MAX LENGTH to the LENGTH parameters of the POST_ON_RECEIPT verb.
   Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).
   Set the RETURN_CODE of the POST_ON_RECEIPT verb to OK.
```

PREPARE_TO_RECEIVE_PROC

FUNCTION:	This procedure handles the PREPARE_TO_RECEIVE verb. Depending on the TYPE of the PREPARE_TO_RECEIVE (FLUSH, CONFIRM or SYNC_LEVEL) and the SYNC_LEVEL of the conversation (NONE, CONFIRM, or SYNCPT), the processing of the PREPARE_TO_RECEIVE is continued by other procedures.
INPUT:	PREPARE_TO_RECEIVE verb parameters
OUTPUT:	If the PREPARE_TO_RECEIVE specifies TYPE = SYNC_LEVEL and the SYNC_LEVEL of the conversation is SYNCPT, the RETURN_CODE is set to OK and FSM_CONVERSATION is updated to indicate that completion of the PREPARE_TO_RECEIVE processing is deferred until a FLUSH, CONFIRM, or SYNCPT verb is issued. Otherwise, processing is continued by other procedures.

Referenced procedures, FSMs, and data structures:

PREPARE_TO_RECEIVE_CONFIRM_PROC	page 5.1-41
PREPARE_TO_RECEIVE_FLUSH_PROC	page 5.1-42
FSM_CONVERSATION	page 5.1-65
RCB	page A-6

```
Find the RCB for the conversation identified in the RESOURCE parameter.
If the data sent by TP is not on a logical record boundary then
   Set the RETURN_CODE of the PREPARE_TO_RECEIVE verb to PROGRAM_STATE_CHECK.
Else
   Select based on the following conditions:
      When the TYPE parameter of the PREPARE_TO_RECEIVE verb is FLUSH, or the TYPE is
       SYNC_LEVEL and the conversation sync level is NONE
         If executing FSM_CONVERSATION(S, PREPARE_TO_RECEIVE_FLUSH, RCB) (page 5.1-65).
          would cause a state-check (>) condition then
            Set the RETURN_CODE of the PREPARE_TO_RECEIVE verb to PROGRAM_STATE_CHECK.
         Else
            Call PREPARE_TO_RECEIVE_FLUSH_PROC(PREPARE_TO_RECEIVE verb parameters, RCB)
             (page 5.1-42).
      When the TYPE parameter of the PREPARE_TO_RECEIVE verb is CONFIRM
         If executing FSM_CONVERSATION(S, PREPARE_TO_RECEIVE_CONFIRM, RCB) (page 5.1-65).
          would cause a state-check (>) condition then
            Set the RETURN_CODE of the PREPARE_TO_RECEIVE verb to PROGRAM_STATE_CHECK.
         Else
            If sync level of the conversation is CONFIRM or SYNCPT then
               Call PREPARE_TO_RECEIVE_CONFIRM_PROC(PREPARE_TO_RECEIVE verb parameters, RCB)
                (page 5.1-41).
            Else
               Set the RETURN_CODE of the PREPARE_TO_RECEIVE verb to PROGRAM_PARAMETER_CHECK.
      When the TYPE parameter of the PREPARE_TO_RECEIVE verb is SYNC_LEVEL and the conversation
       sync level is CONFIRM
         If executing FSM_CONVERSATION(S, PREPARE_TO_RECEIVE_CONFIRM, RCB) (page 5.1-65).
          would cause a state-check (>) condition then
            Set the RETURN_CODE of the PREPARE_TO_RECEIVE verb to PROGRAM_STATE_CHECK.
         Else
            Call PREPARE_TO_RECEIVE_CONFIRM_PROC(PREPARE_TO_RECEIVE verb parameters, RCB)
             (page 5.1-41).
```

When the TYPE parameter of the PREPARE_TO_RECEIVE verb is SYNC_LEVEL and the conversation sync level is SYNCPT
  If executing FSM_CONVERSATION(S, PREPARE_TO_RECEIVE_DEFER, RCB) (page 5.1-65). would cause a state-check (>) condition then
    Set the RETURN_CODE of the PREPARE_TO_RECEIVE verb to PROGRAM_STATE_CHECK.
  Else
    Call FSM_CONVERSATION(S, PREPARE_TO_RECEIVE_DEFER, RCB) (page 5.1-65).
    Set RCB.LOCKS to the LOCKS parameter in the PREPARE_TO_RECEIVE verb.
    Set the RETURN_CODE of the PREPARE_TO_RECEIVE verb parameter to OK.

RECEIVE_AND_WAIT_PROC

FUNCTION: This procedure handles the RECEIVE_AND_WAIT verb.

If the conversation is in an appropriate state (i.e., RECEIVE_AND_WAIT can be issued when the conversation is in the send or receive state), processing of the record continues. PS first receives any records from RM and HS. Appropriate action is taken depending upon which, if any, record was received (as reflected by the state of FSM_ERROR_OR_FAILURE).

INPUT: RECEIVE_AND_WAIT verb parameters

OUTPUT: The DATA field is cleared before receiving data from HS. RCB.RQ_TO_SEND_RCVD is updated. If a RQ_TO_SEND has been received, an indication will be passed up to the TP at this time, and the field in the RCB is updated. The state of FSM_CONVERSATION may change. See below for additional outputs.

NOTES:
1. If a CONVERSATION_FAILURE has been received from the resources manager, PS returns to the transaction program after setting the RETURN_CODE parameter to RESOURCE_FAILURE. The setting of RESOURCE_FAILURE is done after all data currently buffered is passed up to the transaction program.
2. If a RECEIVE_ERROR has been received from HS, PS sends a SEND_DATA record with the MU.PS_TO_HS.TYPE field set to PREPARE_TO_RCV_FLUSH to HS. (Any data in the RCB send buffer was purged when the RECEIVE_ERROR record was received.) PS then waits for the expected FMH-7 error message to arrive. The RETURN_CODE parameter of the RECEIVE_AND_WAIT is set based on the sense data carried in the FMH-7.
3. If the conversation is in the SEND state, PS sends the MU record, containing all saved data from the transaction program, with the MU.PS_TO_HS.TYPE field set to PREPARE_TO_RCV_FLUSH to HS. Regardless of the state of the conversation, PS initializes the post conditions, waits for information to arrive to cause the conversation to become posted, and returns to the transaction program with the received information.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
CREATE_AND_INIT_LIMITED_MU	page 5.1-30
DEQUEUE_FMH7_PROC	page 5.1-34
RECEIVE_AND_TEST_POSTING	page 5.1-50
RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
RCB	page A-6

```
Find the RCB for the conversation identified in the RESOURCE parameter.
If executing FSM_CONVERSATION(S,RECEIVE_AND_WAIT, RCB) would
 cause a state-check (>) condition then
   Set the RETURN_CODE of the RECEIVE_AND_WAIT verb to PROGRAM_STATE_CHECK.
Else
   Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).
   If the state of FSM_ERROR_OR_FAILURE is RCVD_ERROR then (see Note 2)
      If the state of FSM_CONVERSATION is SEND_STATE then (see Note 3)
         If a send MU buffer does not exist then
            Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
         Set MU.PS_TO_HS.TYPE to PREPARE_TO_RCV_FLUSH and send the MU record to HS.
      If the FMH7 is not in the RCB.HS_TO_PS_BUFFER_LIST then
         Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(SUSPEND_LIST containing RCB_ID) (page 5.1-51).
      If the state of FSM_ERROR_OR_FAILURE is CONV_FAILURE_SON or
       CONV_FAILURE_PROTOCOL_ERROR then (see Note 1)
         If the state of FSM_ERROR_OR_FAILURE is CONV_FAILURE_SON then
            Set the RETURN_CODE of the RECEIVE_AND_WAIT verb to RESOURCE_FAILURE_RETRY.
         Else
            Set the RETURN_CODE of the RECEIVE_AND_WAIT verb to RESOURCE_FAILURE_NO_RETRY.
         Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
      Else
         Call DEQUEUE_FMH7_PROC(RECEIVE_AND_WAIT verb parameters, RCB) (page 5.1-34).
   Else
      Call FSM_CONVERSATION(S, RECEIVE_AND_WAIT, RCB) (page 5.1-65).
      Initialize the DATA parameter of the RECEIVE_AND_WAIT verb to null.
      Call RECEIVE_AND_TEST_POSTING(RCB, RECEIVE_AND_WAIT verb parameters) (page 5.1-50).
   Set REQUEST_TO_SEND_RECEIVED of the RECEIVE_AND_WAIT verb to RCB.RQ_TO_SEND_RCVD.
   Set RCB.RQ_TO_SEND_RCVD to NO.
```

RECEIVE_IMMEDIATE_PROC

FUNCTION: This procedure performs the processing of the RECEIVE_IMMEDIATE verb. It receives any information available from the specified conversation, but does not wait for information to arrive.

The procedure first receives any records from the RM_TO_PS and HS_TO_PS queues. Appropriate action is taken depending upon which, if any, record was received (as reflected by the state of FSM_ERROR_OR_FAILURE).

INPUT: RECEIVE_IMMEDIATE verb parameters

OUTPUT: The RCB.POST_CONDITIONS.MAX_LENGTH and FILL are updated to reflect the values on the verb. The DATA field of the RECEIVE_IMMEDIATE verb is cleared before the data is received from HS. After receive processing is performed, the state of FSM_POST is reset and the data is returned to the TP. The RETURN_CODE and REQUEST_TO_SEND_RECEIVED fields of the RECEIVE_IMMEDIATE record are also set to indicate the result of the verb. See below for additional output.

NOTES:

1. If a CONVERSATION_FAILURE has been received from the resources manager, PS returns to the transaction program after setting the RETURN_CODE field in the RECEIVE_IMMEDIATE to RESOURCE_FAILURE.

2. If a RECEIVE_ERROR has been received from HS, PS waits for the expected FMH-7 error message to arrive. The RETURN_CODE field in the RECEIVE_IMMEDIATE is set based on the sense data carried in the FMH-7.

3. If no error or failure condition has occurred, PS calls PERFORM_RECEIVE_PROCESSING (page 5.1-40), which checks to see if any information has arrived on the specified conversation and passes the received information (if any) to the transaction program.

Referenced procedures, FSMs, and data structures:

DEQUEUE_FMH7_PROC	page 5.1-34
PERFORM_RECEIVE_PROCESSING	page 5.1-40
RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
FSM_POST	page 5.1-68
RCB	page A-6

```
Find the RCB for the conversation identified in the RESOURCE parameter.
If executing FSM_CONVERSATION(S, RECEIVE_IMMEDIATE, RCB) would
 cause a state-check (>) condition then
   SET the RETURN_CODE of the RECEIVE_IMMEDIATE verb to PROGRAM_STATE_CHECK.
Else
   Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).
   If the state of FSM_ERROR_OR_FAILURE is RCVD_ERROR then
      If the FMH7 is not in the RCB.HS_TO_PS_BUFFER_LIST then
         Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(SUSPEND_LIST containing RCB_ID) (page 5.1-51).
      If the state of FSM_ERROR_OR_FAILURE is CONV_FAILURE_SON or
       CONV_FAILURE_PROTOCOL_ERROR then
         If the state of FSM_ERROR_OR_FAILURE is CONV_FAILURE_SON then
            Set the RETURN_CODE of the RECEIVE_IMMEDIATE verb to RESOURCE_FAILURE_RETRY.
         Else
            Set the RETURN_CODE of the RECEIVE_IMMEDIATE verb to RESOURCE_FAILURE_NO_RETRY.
         Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
      Else
         Call DEQUEUE_FMH7_PROC(RECEIVE_IMMEDIATE verb parameters, RCB) (page 5.1-34).
   Else (see Notes 1 and 3)
      Call FSM_CONVERSATION(S, RECEIVE_IMMEDIATE, RCB) (page 5.1-65).
      Set RCB.POST_CONDITIONS.MAX_LENGTH to the RECEIVE_IMMEDIATE verb MAX_LENGTH value.
      Set RCB.POST_CONDITIONS.FILL to the RECEIVE_IMMEDIATE verb FILL value.

      Initialize the DATA parameter of the RECEIVE_IMMEDIATE verb to null.
      Call PERFORM_RECEIVE_PROCESSING(RCB, RECEIVE_IMMEDIATE verb parameters) (page 5.1-40).
      Call FSM_POST (page 5.1-68) and pass it a RECEIVE_IMMEDIATE signal.
   Set MAX_LENGTH of the RECEIVE_IMMEDIATE verb to the length of data returned.
   Set REQUEST_TO_SEND_RECEIVED of the RECEIVE_IMMEDIATE verb to RCB.RQ_TO_SEND_RCVD.
   Set RCB.RQ_TO_SEND_RCVD to NO.
```

REQUEST_TO_SEND_PROC

FUNCTION:	This procedure handles REQUEST_TO_SEND verb processing. If the conversation is in the RECEIVE state, PS completes the processing of the REQUEST_TO_SEND record, as described below.
INPUT:	REQUEST_TO_SEND verb parameters
OUTPUT:	The REQUEST_TO_SEND return code is updated, and a REQUEST_TO_SEND will be sent. See below for additional outputs.
NOTES:	1. Since REQUEST_TO_SEND does not support a RESOURCE_FAILURE return code, error conditions cannot be relayed to the transaction program at this time. PS remembers the error (via FSM_ERROR_OR_FAILURE) and reports it to the transaction program at a later time (i.e., when a verb is issued by the transaction program that supports a RESOURCE_FAILURE return code). 2. A REQUEST_TO_SEND record is not sent to HS if the partner transaction program has already issued a DEALLOCATE for the specified conversation. 3. A REQUEST_TO_SEND record is not sent to HS if the partner transaction program has already issued a PREPARE_TO_RECEIVE for the specified conversation. 4. If no records have been received from HS, or records have been received but do not indicate DEALLOCATE or PREPARE_TO_RCV, this procedure sends REQUEST_TO_SEND to HS and waits for the expected RSP_TO_REQUEST_TO_SEND before returning to the transaction program.

Referenced procedures, FSMs, and data structures:

RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
SEND_REQUEST_TO_SEND_PROC	page 5.1-58
WAIT_FOR_RSP_TO_RQ_TO_SEND_PROC	page 5.1-63
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
RCB	page A-6

```
Find the RCB for the conversation identified in the RESOURCE parameter.
If executing FSM_CONVERSATION(S, RECEIVE_IMMEDIATE, RCB) would cause
 a state-check (>) condition then
   Set the RETURN_CODE of the REQUEST_TO_SEND verb to PROGRAM_STATE_CHECK.
Else
   Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).

   If the state of FSM_ERROR_OR_FAILURE is NO_REQUESTS or RCVD_ERROR (see Note 1) then
      Select based on the received end-of-chain type for the conversation:
         When DEALLOCATE_FLUSH or DEALLOCATE_CONFIRM (see Note 2)
            Do nothing.
         When PREPARE_TO_RCV_FLUSH or PREPARE_TO_RCV_CONFIRM (see Note 3)
            Do nothing.
         Otherwise (see Note 4)
            Call SEND_REQUEST_TO_SEND_PROC(RCB) (page 5.1-58).
            Call WAIT_FOR_RSP_TO_RQ_TO_SEND_PROC(RCB) (page 5.1-63).
   Set the RETURN_CODE of the REQUEST_TO_SEND verb to OK.
```

SEND_DATA_PROC

FUNCTION:	This procedure handles the receipt of data from the transaction program. If the resource specified in the SEND_DATA is valid and the conversation is in the SEND state, processing of the record continues. PS first retrieves any records from RM and HS. Appropriate action is taken depending upon which, if any, record was received.
INPUT:	SEND_DATA verb parameters and a possible RQ_TO_SEND may have been received on this conversation.
OUTPUT:	The RETURN_CODE of the SEND_DATA verb is set. The states of FSM_CONVERSATION and FSM_ERROR_OR_FAILURE may changed. If RQ_TO_SEND_RCVD has been received, an indication is stored in the RCB.RQ_TO_SEND_RCVD field. This YES/NO indication will be passed up to the TP and then the RCB.RQ_TO_SEND_RCVD field is reset to indicate that no RQ_TO_SENDs are outstanding. See Notes for additional outputs.
NOTES:	1. If a CONVERSATION_FAILURE record has been received from the resources manager, PS returns to the transaction program after setting the RETURN_CODE parameter of the SEND_DATA to RESOURCE_FAILURE. 2. If a RECEIVE_ERROR has been received from HS, PS sends a SEND_DATA record with the MU.PS_TO_HS.TYPE field set to PREPARE_TO_RCV_FLUSH to HS. (Any data in the RCB send buffer was purged when the RECEIVE_ERROR record was received.) PS then waits for the expected FMH-7 error message to arrive. The RETURN_CODE parameter of the SEND_DATA is set based on the sense data carried in the FMH-7. 3. If no error or failure condition has occurred, PS scans the data in the passed SEND_DATA for logical record boundaries. (PS maintains in the RCB a count of the number of bytes of data remaining to be sent from the transaction program to finish the current logical record.) If there is enough data to send to HS, PS sends it.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
CREATE_AND_INIT_LIMITED_MU	page 5.1-30
DEQUEUE_FMH7_PROC	page 5.1-34
RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
SEND_DATA_BUFFER_MANAGEMENT	page 5.1-54
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
MU	page A-29
RCB	page A-6

```
Find the RCB for the conversation identified in the RESOURCE parameter.
If executing FSM_CONVERSATION(S, RECEIVE_IMMEDIATE, RCB) would cause a
 state-check (>) condition then
   Set the RETURN_CODE of the SEND_DATA verb to PROGRAM_STATE_CHECK.
Else
   Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).
   Select based on the state of FSM_ERROR_OR_FAILURE
      When CONV_FAILURE_PROTOCOL_ERROR or CONV_FAILURE_SON (see Note 1)
         If the state of FSM_ERROR_OR_FAILURE is CONV_FAILURE_SON then
            Set the RETURN_CODE of the SEND_DATA verb to RESOURCE_FAILURE_RETRY.
         Else
            Set the RETURN_CODE of the SEND_DATA verb to RESOURCE_FAILURE_NO_RETRY.
         Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
```

```
      When RCVD_ERROR (see Note 2)
         If a send MU buffer does not exist then
            Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
         Set MU.PS_TO_HS.TYPE to PREPARE_TO_RCV_FLUSH and send the MU record to HS.
         Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(SUSPEND_LIST containing RCB_ID) (page 5.1-51).

         If the state of FSM_ERROR_OR_FAILURE is CONV_FAILURE_SON or
          CONV_FAILURE_PROTOCOL_ERROR (see Note 1) then
            If the state of FSM_ERROR_OR_FAILURE is CONV_FAILURE_SON then
               Set the RETURN_CODE of the SEND_DATA verb to RESOURCE_FAILURE_RETRY.
            Else
               Set the RETURN_CODE of the SEND_DATA verb to RESOURCE_FAILURE_NO_RETRY.
            Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
         Else
            Call DEQUEUE_FMH7_PROC(SEND_DATA verb parameters, RCB) (page 5.1-34).
      When NO_REQUESTS (see Note 3)
         Set the RETURN_CODE of the SEND_DATA verb to OK.
         If MAX_LENGTH of the SEND_DATA verb is greater than 0 then
            Perform the LL processing (see Note 3).
            If LL is not valid (i.e., values X'0000', X'8000', and X'8001' are not valid;
             X'0001' is valid only for PS headers--see SNA Formats) then
               Set the RETURN_CODE of the SEND_DATA verb to PROGRAM_PARAMETER_CHECK.
            Else
               Call SEND_DATA_BUFFER_MANAGEMENT(DATA from SEND_DATA verb, RCB) (page 5.1-54).
   Set REQUEST_TO_SEND_RECEIVED of the SEND_DATA verb to RCB.RQ_TO_SEND_RCVD.
   Set RCB.RQ_TO_SEND_RCVD to NO.
```

SEND_ERROR_PROC

FUNCTION:	This procedure handles the SEND_ERROR verb processing. If the resource specified in the SEND_ERROR is valid and the conversation is in an appropriate state, processing of the SEND_ERROR continues. PS first retrieves any records from RM and HS. Appropriate action is taken depending upon which, if any, record was received (as reflected by the state of FSM_ERROR_OR_FAILURE).
INPUT:	SEND_ERROR verb parameters
OUTPUT:	The return code of the SEND_ERROR verb is updated. If the RCB indicates that a RQ_TO_SEND_RCVD has been received, it will be passed up to the TP at this time and the RCB.RQ_TO_SEND_RCVD field will be reset to NO. The state of FSM_CONVERSATION may be changed. See below for additional outputs.
NOTES:	1. If a CONVERSATION_FAILURE has been received from the resources manager, PS returns to the transaction program after setting the RETURN_CODE parameter of the SEND_ERROR to RESOURCE_FAILURE. 2. If RECEIVE_ERROR has been received from HS or no error records have been received, further processing of the SEND_ERROR is performed, depending upon the state of the conversation.

Referenced procedures, FSMs, and data structures:

RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
SEND_ERROR_DONE_PROC	page 5.1-55
SEND_ERROR_IN_RECEIVE_STATE	page 5.1-56
SEND_ERROR_IN_SEND_STATE	page 5.1-57
SEND_ERROR_TO_HS_PROC	page 5.1-58
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
RCB	page A-6

SEND_ERROR_PROC

```
Find the RCB for the conversation identified in the RESOURCE parameter.
If executing FSM_CONVERSATION(S, SEND_ERROR, RCB) would  cause a
 state-check (>) condition then
   SET the RETURN_CODE of the SEND_ERROR verb to PROGRAM_STATE_CHECK.
Else
   Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).
   Select based on the state of FSM_ERROR_OR_FAILURE (page 5.1-67):
      When CONV_FAILURE_PROTOCOL_ERROR or CONV_FAILURE_SON (see Note 1)
         Call FSM_CONVERSATION(S, SEND_ERROR, RCB) (page 5.1-65).
         If the state of FSM_ERROR_OR_FAILURE is CONV_FAILURE_SON then
            Set the RETURN_CODE of the SEND_ERROR verb to RESOURCE_FAILURE_RETRY.
         Else
            Set the RETURN_CODE of the SEND_ERROR verb to RESOURCE_FAILURE_NO_RETRY.
         Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
      When NO_REQUESTS or RCVD_ERROR (see Note 2)
         Select based on the state of FSM_CONVERSATION (page 5.1-65):
            When SEND_STATE
               Call SEND_ERROR_IN_SEND_STATE(SEND_ERROR verb parameters, RCB) (page 5.1-57).
            When RCVD_CONFIRM, RCVD_CONFIRM_SEND, or RCVD_CONFIRM_DEALL
               Call SEND_ERROR_TO_HS_PROC(RCB) (page 5.1-58).
               Call FSM_CONVERSATION(S, SEND_ERROR, RCB) (page 5.1-65).
               Call SEND_ERROR_DONE_PROC(SEND_ERROR verb parameters, RCB) (page 5.1-55).
            When RCV_STATE
               Call SEND_ERROR_IN_RECEIVE_STATE(SEND_ERROR verb parameters, RCB) (page 5.1-56).
   Set REQUEST_TO_SEND_RECEIVED of the SEND_ERROR verb to RCB.RQ_TO_SEND_RCVD.
   Set RCB.RQ_TO_SEND_RCVD to NO.
```

TEST_PROC

FUNCTION:	This procedure performs the processing of a TEST record. The procedure first receives any records from RM and HS. It then tests whether the conversation has been posted or whether REQUEST_TO_SEND notification has been received from the remote transaction. The RETURN_CODE field of TEST records the result of the test.
INPUT:	TEST record
OUTPUT:	The RETURN_CODE field of TEST records the result of the test. If the TP is informed that a RQ_TO_SEND has been received, then the RCB.RQ_TO_SEND_RCVD field is reset to NO.

Referenced procedures, FSMs, and data structures:

DEQUEUE_FMH7_PROC	page 5.1-34
RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
TEST_FOR_POST_SATISFIED	page 5.1-60
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
FSM_POST	page 5.1-68
RCB	page A-6

```
Find the RCB for the resource identified in the RESOURCE field of the TEST record.
Set the RETURN_CODE of the TEST verb of TEST to OK.
Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).
Select based on the TEST parameter of the TEST verb:
   When POSTED
      If executing FSM_CONVERSATION(S, TEST_POSTED, RCB) (page 5.1-65)
       would cause a state-check (>) condition then
         Set the RETURN_CODE of the TEST verb to PROGRAM_STATE_CHECK.
      Else
         If state of FSM_POST is RESET then
            Set the RETURN_CODE of the TEST verb to POSTING_NOT_ACTIVE.
         Else
            Select based on the state of FSM_ERROR_OR_FAILURE (page 5.1-67):
               When CONV_FAILURE_SON
                  Set the RETURN_CODE of the TEST verb to RESOURCE_FAILURE_RETRY.
                  Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
               When CONV_FAILURE_PROTOCOL_ERROR
                  Set the RETURN_CODE of the TEST verb to RESOURCE_FAILURE_NO_RETRY.
                  Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
               When RCVD_ERROR
                  If the FMH7 is not in the RCB.HS_TO_PS_BUFFER_LIST then
                     Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(SUSPEND_LIST containing RCB_ID)
                      (page 5.1-51).
                  If state of FSM_ERROR_OR_FAILURE (page 5.1-67) is CONV_FAILURE_SON or
                   CONV_FAILURE_PROTOCOL_ERROR then
                     If state of FSM_ERROR_OR_FAILURE (page 5.1-67) is CONV_FAILURE_SON then
                        Set the RETURN_CODE of the TEST verb to RESOURCE_FAILURE_RETRY.
                     Else
                        Set the RETURN_CODE of the TEST verb to RESOURCE_FAILURE_NO_RETRY.
                     Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
                  Else
                     Call DEQUEUE_FMH7_PROC(TEST verb parameters, RCB) (page 5.1-34).
               When NO_REQUESTS
                  Call TEST_FOR_POST_SATISFIED(RCB) (page 5.1-60).
                  Select on state of FSM_POST:
                     When PEND_POSTED
                        Set the RETURN_CODE of the TEST verb to UNSUCCESSFUL.
                     When POSTED
                        If an FMH-7 is the next thing to process then
                           Call DEQUEUE_FMH7_PROC(TEST verb parameters, RCB)
                            (page 5.1-34).
                        Else
                           Set the RETURN_CODE subcode to NOT_DATA or DATA as
                            appropriate.
         If the state of FSM_CONVERSATION is not END_CONV then (page 5.1-65).
            Call FSM_CONVERSATION(S, TEST, RCB) (page 5.1-65).
         Call FSM_POST (page 5.1-68) and pass it a TEST signal.
   When REQUEST_TO_SEND_RECEIVED
      If executing FSM_CONVERSATION(S, TEST_RQ_TO_SEND_RCVD, RCB) (page 5.1-65)
       would cause a state-check (>) condition then
         Set the RETURN_CODE of the TEST verb to PROGRAM_STATE_CHECK.
      Else
         If RCB.RQ_TO_SEND_RCVD is YES then
            Set RCB.RQ_TO_SEND_RCVD to NO.
         Else
            Set the RETURN_CODE of the TEST verb to UNSUCCESSFUL.
         Call FSM_CONVERSATION(S, TEST_RQ_TO_SEND_RCVD, RCB) (page 5.1-65).
```

COMPLETE_CONFIRM_PROC

FUNCTION:		This procedure completes the processing of a CONFIRM verb. It is called by CONFIRM_PROC (page 5.1-12) when no error or failure conditions are indicated by FSM_ERROR_OR_FAILURE (page 5.1-67). The action of this procedure is dependent on the state of the conversation, as described below.
INPUT:		CONFIRM parameters and the RCB corresponding to the resource specified in the CONFIRM verb
OUTPUT:		The MU.PS_TO_HS.TYPE field is set before sending the MU to HS. See Notes for additional outputs.
NOTES:	1.	If FSM_CONVERSATION is in the SEND_STATE, an MU with MU.PS_TO_HS.TYPE field set to CONFIRM is sent to HS.
	2.	If FSM_CONVERSATION is in the PREPARE_TO_RECEIVE_DEFER state, an MU with MU.PS_TO_HS.TYPE field set to PREPARE_TO_RCV_CONFIRM is sent to HS.
	3.	If FSM_CONVERSATION is in the DEALLOCATE_DEFER state, an MU with MU.PS_TO_HS.TYPE field set to DEALLOCATE_CONFIRM is sent to HS.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
CREATE_AND_INIT_LIMITED_MU	page 5.1-30
WAIT_FOR_CONFIRMED_PROC	page 5.1-61
FSM_CONVERSATION	page 5.1-65
MU	page A-29
RCB	page A-6

```
If a send MU buffer does not exist then
   Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
Select based on the state of FSM_CONVERSATION (page 5.1-65):
   When SEND_STATE (see Note 1)
      Set MU.PS_TO_HS.TYPE to CONFIRM and send the MU record to HS.
   When PREP_TO_RCV_DEFER (see Note 2)
      Set MU.PS_TO_HS.TYPE to PREPARE_TO_RCV_CONFIRM_SHORT or  PREPARE_TO_RCV_CONFIRM_LONG as
       indicated by RCB.LOCKS and send the MU record to HS.
   When DEALL_DEFER (see Note 3)
      Set MU.PS_TO_HS.TYPE to DEALLOCATE_CONFIRM and send the MU record to HS.
Call FSM_CONVERSATION(S, CONFIRM, RCB) (page 5.1-65).
Call WAIT_FOR_CONFIRMED_PROC(CONFIRM verb parameters, RCB) (page 5.1-61).
```

COMPLETE_DEALLOCATE_ABEND_PROC

FUNCTION:	This procedure completes the processing of a DEALLOCATE verb that specifies TYPE = ABEND. If an MU buffer for storage of data sent by the transaction program currently exists, PS sends it to the HS and another MU buffer is obtained for storing the FMH-7; otherwise, a new MU buffer is obtained for storing the FMH-7. PS creates an FMH-7 and places it in the newly-created MU. The FMH-7 carries sense data indicating DEALLOCATE_ABEND. If any log data is associated with the DEALLOCATE, PS creates an Error Log GDS variable (see SNA Formats) and places it in the MU to be sent to the partner LU. PS also places the GDS variable (minus the LL and GDS ID fields) in the local LU's system error log. PS then sends the MU, containing the FMH-7 and optional Error Log GDS variable, to HS.
INPUT:	DEALLOCATE verb parameters and the RCB corresponding to the resource specified in the DEALLOCATE
OUTPUT:	One or more MUs are sent to HS. Any log data supplied with the DEALLOCATE is logged.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
CREATE_AND_INIT_LIMITED_MU	page 5.1-30
SEND_DATA_BUFFER_MANAGEMENT	page 5.1-54
MU	page A-29

```
Set sense data based on the TYPE parameter of the DEALLOCATE verb as follows:
 to X'08640000' if ABEND_PROG, or
 to X'08640001' if ABEND_SVC, or
 to X'8640002'if ABEND_TIMER.
If a send MU buffer exists then
   Send the MU record to HS.
Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
If LOG_DATA parameter has been supplied then
   Store the FMH-7 indicating log data present and sense data in the MU.
   Create an Error Log GDS variable (see SNA Formats for format).
   Call SEND_DATA_BUFFER_MANAGEMENT(Error log GDS variable, RCB) (page 5.1-54)
    to concatenate the error log GDS variable to the FMH-7 in the MU.
   Log the Error Log GDS variable in the system error log.
Else
   Store the FMH-7 with sense data but no log data present in the MU.
Set MU.PS_TO_HS.TYPE to DEALLOCATE_FLUSH and send the MU record to HS.
```

CONVERSATION_FAILURE_PROC

FUNCTION:	This procedure processes CONVERSATION_FAILURE records.
INPUT:	A CONVERSATION_FAILURE record
OUTPUT:	FSM_ERROR_OR_FAILURE is set to the appropriate state. PS remembers the conversation failure until that information can be relayed to the transaction program. If posting is active, FSM_POST is called to change the state to POSTED.

Referenced procedures, FSMs, and data structures:

FSM_ERROR_OR_FAILURE	page 5.1-67
FSM_POST	page 5.1-68
CONVERSATION_FAILURE	page A-21
RCB	page A-6

CONVERSATION_FAILURE_PROC

Find the RCB for the conversation identified in the RESOURCE parameter.
If the RCB for the CONVERSATION_FAILURE record is found, then
   If CONVERSATION_FAILURE.REASON is PROTOCOL_VIOLATION then
      Call FSM_ERROR_OR_FAILURE (page 5.1-67) and pass it a CONV_FAIL_PROTOCOL signal.
   Else
      Call FSM_ERROR_OR_FAILURE (page 5.1-67) and pass it a CONV_FAIL_SON signal.
   If the state of FSM_POST is PEND_POSTED then
      Call FSM_POST (page 5.1-68) and pass it a POST signal.

CREATE_AND_INIT_LIMITED_MU

FUNCTION:	This procedure creates and initializes an MU in a buffer from the limited buffer pool associated with the LIMITED_BUFFER_POOL_ID value stored in the RCB.
INPUT:	The RCB corresponding to the conversation for which the MU is being requested.
OUTPUT:	Appropriate fields in the MU are initialized, the MU is returned to the calling procedure.
NOTE:	As a result of a race condition, the half-session for this PS may have been destroyed and PS has not received the CONVERSATION_FAILURE record from RM. When the half-session is destroyed, the POOL for the buffers is also destroyed; thus, the attempt to retrieve a POOL buffer will fail with a BAD_POINTER return code. The calling tree structure of the PS process requires that a buffer be present, so a demand buffer is obtained before returning from this procedure. Later, this demand buffer will be freed when PS attempts to send the buffer to HS and discovers that HS has been destroyed.

Referenced procedures, FSMs, and data structures:

MU	page A-29
RCB	page A-6

Get buffer for MU-buffer size is RCB.SEND_RU_SIZE.

Call buffer manager (GET_BUFFER, limited buffer pool ID, wait) to create a
 MU buffer; the buffer from this pool will be equal to the SEND_RU_SIZE
 value stored in the RCB (Appendix B).
If the buffer manager return code is BAD_POINTER (see Note) then
   Call buffer manager (GET_BUFFER, demand, buffer size, wait) to create a buffer
    for the MU record; specify the buffer size to be RCB.SEND_RU_SIZE plus the
    length of the MU overhead (Appendix B).

Initialize fields in the MU

Set MU.HEADER_TYPE to PS_TO_HS.
Set MU.PS_TO_HS.BRACKET_ID to RCB.BRACKET_ID.
Set MU.PS_TO_HS.PS_TO_HS_VARIANT to SEND_DATA_RECORD.
Set MU.PS_TO_HS.ALLOCATE to NO.
Set MU.PS_TO_HS.FMH to NO.
Set MU.PS_TO_HS.TYPE to FLUSH.
Set MU.DCF to indicate the length of data and the RH field.

The MU is available for storing data from the TP.

DEALLOCATE_ABEND_PROC

FUNCTION:	This procedure is invoked when the TYPE parameter of DEALLOCATE verb is ABEND_PROG, ABEND_SVC, or ABEND_TIMER. PS first receives any records from RM and HS. Appropriate action is taken depending upon which, if any, record was received and upon the state of the conversation. The state of the conversation and the information in the HS_TO_PS_BUFFER_LIST determine whether or not a SEND_ERROR MU is sent to HS prior to sending the FMH-7 that is created as a result of the DEALLOCATE (TYPE = ABEND_*). Receipt of certain types of information (e.g., notification that the conversation has been deallocated by the partner transaction program) causes PS to return to the transaction program without taking any action.
INPUT:	DEALLOCATE verb parameters and the RCB corresponding to the resource specified in the DEALLOCATE
OUTPUT:	Depending upon the state of the conversation and the information contained in the HS_TO_PS_BUFFER_LIST, an FMH-7 (possibly preceded by a SEND_ERROR MU) is created and sent to HS, or no output is created. All received MUs are purged from the HS_TO_PS_BUFFER_LIST before returning to the transaction program.

Referenced procedures, FSMs, and data structures:

COMPLETE_DEALLOCATE_ABEND_PROC	page 5.1-29
END_CONVERSATION_PROC	page 5.1-34
RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
SEND_ERROR_TO_HS_PROC	page 5.1-58
WAIT_FOR_SEND_ERROR_DONE_PROC	page 5.1-64
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
RCB	page A-6

```
Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).
If the state of FSM_ERROR_OR_FAILURE (page 5.1-67) is NO_REQUEST or RCVD_ERROR then
   Select based on the state of FSM_CONVERSATION (page 5.1-65):
      When RCV_STATE
         If DEALLOCATE_FLUSH has not been received on the conversation then
            Call SEND_ERROR_TO_HS_PROC(RCB) (page 5.1-58).
            Call WAIT_FOR_SEND_ERROR_DONE_PROC(DEALLOCATE parameters, RCB) (page 5.1-64).
      When RCVD_CONFIRM, RCVD_CONFIRM_SEND, or RCVD_CONFIRM_DEALL
         Call SEND_ERROR_TO_HS_PROC(RCB) (page 5.1-58).
         Call COMPLETE_DEALLOCATE_ABEND_PROC(DEALLOCATE verb parameters, RCB) (page 5.1-29).
      When SEND_STATE, PREP_TO_RCV_DEFER, or DEALL_DEFER
         Call COMPLETE_DEALLOCATE_ABEND_PROC(DEALLOCATE verb parameters, RCB) (page 5.1-29).
Set the RETURN_CODE of the DEALLOCATE verb to OK.
Call FSM_CONVERSATION(S, DEALLOCATE_ABEND, RCB) (page 5.1-65).
Call END_CONVERSATION_PROC(RCB) (page 5.1-34).
```

DEALLOCATE_CONFIRM_PROC

FUNCTION:		This procedure is invoked when DEALLOCATE TYPE(CONFIRM) or DEALLOCATE TYPE(SYNC_LEVEL) is issued for a conversation whose SYNC_LEVEL is CONFIRM.
		PS first retrieves any records from HS. Appropriate action is taken depending upon which, if any, record was received.
INPUT:		DEALLOCATE verb parameters and the RCB corresponding to the resource specified in the DEALLOCATE
OUTPUT:		See below.
NOTES:	1.	If a CONVERSATION_FAILURE has been received from the resources manager, PS returns to the transaction program after setting the RETURN_CODE parameter of the DEALLOCATE to RESOURCE_FAILURE.
	2.	If a RECEIVE_ERROR has been received from HS, PS sends a SEND_DATA record with the MU.PS_TO_HS.TYPE field set to PREPARE_TO_RCV_FLUSH to HS. (Any data in the RCB send buffer was purged when the RECEIVE_ERROR record was received.) PS then waits for the expected FMH-7 error message to arrive. The RETURN_CODE parameter of the CONFIRM is set based on the sense data carried in the FMH-7.
	3.	If no error or failure condition has occurred, PS sends an MU with the MU.PS_TO_HS.TYPE field set to DEALLOCATE_CONFIRM to HS.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
CREATE_AND_INIT_LIMITED_MU	page 5.1-30
DEQUEUE_FMH7_PROC	page 5.1-34
RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
WAIT_FOR_CONFIRMED_PROC	page 5.1-61
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
MU	page A-29
RCB	page A-6

```
If the data sent by TP is not at a logical record boundary then
   Set the RETURN_CODE of the DEALLOCATE verb to PROGRAM_STATE_CHECK.
Else
   Call FSM_CONVERSATION(S, DEALLOCATE_CONFIRM, RCB) (page 5.1-65).
   Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).
   Select based on the state of FSM_ERROR_OR_FAILURE (page 5.1-67):
      When CONV_FAILURE_PROTOCOL_ERROR (see Note 1)
         Set the RETURN_CODE of the DEALLOCATE verb to RESOURCE_FAILURE_NO_RETRY.
         Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
      When CONV_FAILURE_SON (see Note 1)
         Set the RETURN_CODE of the DEALLOCATE verb to RESOURCE_FAILURE_RETRY.
         Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
      When RCVD_ERROR (see Note 2)
         If a send MU buffer does not exist then
            Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
         Set MU.PS_TO_HS.TYPE to PREPARE_TO_RCV_FLUSH and send the MU to record HS.
         If the FMH7 is not in the RCB.HS_TO_PS_BUFFER_LIST then
            Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(SUSPEND_LIST containing RCB_ID) (page 5.1-51).
         Else
            Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).
         If the state of FSM_ERROR_OR_FAILURE (page 5.1-67) is CONV_FAILURE_SON or
          CONV_FAILURE_PROTOCOL_ERROR then
            If the state of FSM_ERROR_OR_FAILURE (page 5.1-67) is CONV_FAILURE_SON then
               Set the RETURN_CODE of the DEALLOCATE verb to RESOURCE_FAILURE_RETRY.
            Else
               Set the RETURN_CODE of the DEALLOCATE verb to RESOURCE_FAILURE_NO_RETRY.
            Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
         Else
            Call DEQUEUE_FMH7_PROC(DEALLOCATE verb parameters, RCB) (page 5.1-34).
```

```
When NO_REQUESTS (see Note 3)
   If a send MU buffer does not exist then
      Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
   Set MU.PS_TO_HS.TYPE to DEALLOCATE_CONFIRM and send the MU record to HS.
   Call WAIT_FOR_CONFIRMED_PROC(DEALLOCATE verb parameters, RCB) (page 5.1-61).
```

DEALLOCATE_FLUSH_PROC

FUNCTION:		This procedure is invoked when a DEALLOCATE is received that specifies TYPE = FLUSH, or TYPE = SYNC_LEVEL and the SYNC_LEVEL of the conversation is NONE. After checking that the data for the conversation is on a logical record boundary, the procedure accepts any records from RM and HS. Appropriate action is taken, depending upon which, if any, record was received (as reflected by the state of FSM_ERROR_OR_FAILURE).
INPUT:		DEALLOCATE verb parameters and the RCB corresponding to the resource specified in the DEALLOCATE
OUTPUT:		DEALLOCATE return code is set. See Notes for additional outputs.
NOTES:	1.	Since the conversation is currently being ended, as a result of processing the DEALLOCATE, if a RECEIVE_ERROR has been received from HS, it will be ignored by PS.
	2.	If CONVERSATION_FAILURE record has been received from RM, no further records are sent to HS.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
CREATE_AND_INIT_LIMITED_MU	page 5.1-30
END_CONVERSATION_PROC	page 5.1-34
RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
MU	page A-29
RCB	page A-6

```
If the data sent by TP is not at a logical record boundary then
   Set the RETURN_CODE of the DEALLOCATE verb to PROGRAM_STATE_CHECK.
Else
   Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).
   If state of FSM_ERROR_OR_FAILURE is RCVD_ERROR or NO_REQUESTS (see Note 1) then
      If a send MU buffer does not exist then
         Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
      Set MU.PS_TO_HS.TYPE to DEALLOCATE_FLUSH and send the MU record to HS.
   Else (see Note 2)
      Do nothing.
   Set the RETURN_CODE of the DEALLOCATE verb to OK.
   Call FSM_CONVERSATION(S, DEALLOCATE_FLUSH, RCB) (page 5.1-65).
   Call END_CONVERSATION_PROC(RCB) (page 5.1-34).
```

DEQUEUE_FMH7_PROC

FUNCTION:	This procedure is invoked upon receipt of a RECEIVE_ERROR from HS. The next element expected in the HS_TO_PS_BUFFER_LIST is an FMH-7. If the next element in the buffer is an FMH-7, it is removed from the buffer and processed (the RETURN_CODE parameter of the passed verb parameters is set based upon the sense data carried in the FMH-7). If the next element is not an FMH-7, the partner LU has violated the protocol and the session over which the protocol violation occurred is deactivated in an implementation-dependent fashion.
INPUT:	The transaction program verb parameters currently being processed and the RCB corresponding to the resource specified in parameters of the verb
OUTPUT:	The state of FSM_POST is changed to RESET. If the record in the buffer is an FMH-7, then is processed; otherwise, the return code and FSM_CONVERSATION are set to indicate the protocol violation, and the session is deactivated.

Referenced procedures, FSMs, and data structures:

PROCESS_FMH7_PROC	page 5.1-46
PS_PROTOCOL_ERROR	page 5.0-20
FSM_CONVERSATION	page 5.1-65
FSM_POST	page 5.1-68
RCB	page A-6

```
Call FSM_POST (page 5.1-68) and pass it a RECEIVE_IMMEDIATE signal.
If the first entry in RCB.HS_TO_PS_BUFFER_LIST is an FMH-7 then
   Remove the first entry of RCB.HS_TO_PS_BUFFER_LIST.
   Call PROCESS_FMH7_PROC(RCB, TP verb parameters) (page 5.1-46).
Else (as an implementation-dependent option)
   Call PS_PROTOCOL_ERROR (RCB.HS_ID, X'1008201D') (page 5.0-20).
   Set the RETURN_CODE parameter of the verb to RESOURCE_FAILURE_NO_RETRY.
   Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
```

END_CONVERSATION_PROC

FUNCTION:	This procedure creates a DEALLOCATE_RCB and sends it to RM. RM's processing of this record includes removing the RESOURCE from the RESOURCE_LIST, destroying the RCB, and returning a RCB_DEALLOCATED to inform PS that the processing is complete. Before the DEALLOCATE_RCB record is sent to RM, all MUs for the conversation are freed. This includes any that might be present in the HS_TO_PS_BUFFER_LIST (received MUs), or stored in the RCB (current send MU).
INPUT:	The RCB corresponding to the resource being deallocated, and RCB_DEALLOCATED records from RM.
OUTPUT:	DEALLOCATE_RCB record is sent to RM after all received MUs (if present) and sending MU (if present) have been freed.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
RM	page 3-19
WAIT_FOR_RM_REPLY	page 5.1-62
MU	page A-29
RCB	page A-6
DEALLOCATE_RCB	page A-16
RCB_DEALLOCATED	page A-21

Do for each MU in the RCB.HS_TO_PS_BUFFER_LIST (Purge receive buffers)
   Call buffer manager (FREE_BUFFER, buffer address) (Appendix B).
If a send MU buffer exists then (Purge the send buffer)
   Call buffer manager (FREE_BUFFER, buffer address) (Appendix B).
Create and initialize the DEALLOCATE_RCB record and send it to HS.
Call WAIT_FOR_RM_REPLY(RCB) to receive RCB_DEALLOCATED from RM (page 5.1-62).
Destroy the RCB_DEALLOCATED record received from RM.

GET_DEALLOCATE_FROM_HS

FUNCTION:	This procedure removes from the RCB receive buffer a DEALLOCATE MU. If the receive buffer is empty, PS waits until an MU whose MU.HS_TO_PS.TYPE field is set to DEALLOCATE is received from HS. This procedure is invoked only when the next element expected in the RCB receive buffer is a DEALLOCATE MU. This situation occurs, for example, when PS has received an FMH-7 whose sense code indicates an allocation error. The FMH-7 is followed by a notification that the conversation is being deallocated. It is PS's responsibility, rather than the transaction program's, to receive and process the deallocation notification.
INPUT:	The transaction program verb (TRANSACTION_PGM_VERB) currently being processed and the entry in the RCB_LIST corresponding to the resource specified in the current TRANSACTION_PGM_VERB
OUTPUT:	The DEALLOCATE MU is removed from the HS_TO_PS_BUFFER_LIST receive buffer.

Referenced procedures, FSMs, and data structures:

GET_END_CHAIN_FROM_HS	page 5.1-36
PS_PROTOCOL_ERROR	page 5.0-20
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
RCB	page A-6

Call GET_END_CHAIN_FROM_HS(RCB) (page 5.1-36).

Remove the DEALLOCATE from the buffer.

Select based on the following conditions:
   When the end-of-chain type is DEALLOCATE_FLUSH or DEALLOCATE_CONFIRM
      Do nothing.
   When the state of FSM_ERROR_OR_FAILURE (page 5.1-67) is
    CONV_FAILURE_PROTOCOL_ERROR or CONV_FAILURE_SON
      Do nothing.
   Otherwise (as an implementation-dependent option)
      Call PS_PROTOCOL_ERROR (RCB.HS_ID, X'1008201D') (page 5.0-20).
      Set the RETURN_CODE parameter of the verb to RESOURCE_FAILURE_NO_RETRY.
      Call RCB.FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).

GET_END_CHAIN_FROM_HS

FUNCTION: This procedure is invoked after PS sends a SEND_ERROR record to HS (as a result of either 1) a SEND_ERROR, DEALLOCATE (TYPE = ABEND_PROG, ABEND_SVC, ABEND_TIMER) issued for the conversation while it is in the receive state or 2) an invalid Attach resulting in an FMH-7 being sent). This procedure waits for a MU whose MU.HS_TO_PS.TYPE field indicates an end-of-chain type is received from HS. End-of-chain types include CONFIRM, PREPARE_TO_RCV_CONFIRM, PREPARE_TO_RCV_FLUSH, DEALLOCATE_CONFIRM, and DEALLOCATE_FLUSH.

INPUT: The RCB corresponding to the conversation for which the end-of-chain type is desired

OUTPUT: RCB.RQ_TO_SEND_RCVD may be updated. All records received from HS are destroyed or FREEd, as appropriate for the record. The fields in the RCB are reset after the end-of-chain type is received. See Notes for additional outputs.

NOTES:

1. Receipt of a CONVERSATION_FAILURE record will be regarded as an implied end-of-chain type.
2. If a REQUEST_TO_SEND record is received, PS stores that information in the RCB to be relayed to the transaction program at a later time, and continues to wait for the end-of-chain type.
3. If a RECEIVE_ERROR record is received, no action is taken. PS continues to wait for the end-of-chain type to arrive. This situation occurs if, immediately prior to issuing the SEND_ERROR or DEALLOCATE (TYPE = ABEND_*), the transaction program issued a PREPARE_TO_RECEIVE (TYPE = FLUSH) or PREPARE_TO_RECEIVE (TYPE = SYNC_LEVEL) and the SYNC_LEVEL of the conversation is NONE, and the partner transaction program (while still in RECEIVE state) issues a SEND_ERROR or DEALLOCATE (TYPE = ABEND_*).
4. When PS sends SEND_ERROR to HS, it begins to purge any data it receives from HS until a record indicating end-of-chain type is received.

Referenced procedures, FSMs, and data structures:

CONVERSATION_FAILURE_PROC	page 5.1-29
PS_PROTOCOL_ERROR	page 5.0-20
FSM_CONVERSATION	page 5.1-65
MU	page A-29
RCB	page A-6

Determine if end-of-chain type is already in buffer.

```
If end-of-chain type has not been received for this conversation then
   Do for each MU in the RCB.HS_TO_PS_BUFFER_LIST while an end-of-chain
    type has not arrived:
      If MU.HS_TO_PS.TYPE is an end-of-chain type then
         Save the end-of-chain type for later processing.
      Call buffer manager (FREE_BUFFER, buffer address).
```

Wait for the end-of-chain type to arrive.

```
Do while the end-of-chain type has not been received:
   Find a CONVERSATION_FAILURE, REQUEST_TO_SEND, RECEIVE_ERROR, or MU
    record for this conversation.
   Select based on the record found:
      When CONVERSATION_FAILURE (see Note 1)
         Call CONVERSATION_FAILURE_PROC with RECORD (page 5.1-29).
          The CONVERSATION_FAILURE is treated as an end-of-chain indication.
      When REQUEST_TO_SEND (see Note 2)
         Set RCB.RQ_TO_SEND_RCVD to YES.
         Destroy the REQUEST_TO_SEND record.
      When RECEIVE_ERROR (see Note 3)
         Destroy the RECEIVE_ERROR record.
```

```
      When MU (see Note 4)
         If MU.HS_TO_PS.TYPE is an end-of-chain type then
            Save the end-of-chain type for later processing.
         Call buffer manager (FREE_BUFFER, buffer address).
      Otherwise
         Call PS_PROTOCOL_ERROR (RCB.HS_ID, X'10010000') (page 5.0-20).
         Call FSM_CONVERSATION(S, CONFIRMED, RCB) (page 5.1-65).
Update the fields in the RCB to reflect the receipt of the end-of-chain type.
```

OBTAIN_SESSION_PROC

FUNCTION: This procedure handles the acquisition of a session for use by a conversation resource.

This procedure sends a GET_SESSION record to the resources manager and waits for a SESSION_ALLOCATED reply.

INPUT: The RCB corresponding to the conversation that is to use the obtained session and the ALLOCATE verb are passed as a parameters to this procedure. SESSION_ALLOCATED is received from RM.

OUTPUT: A GET_SESSION record is sent to RM, and the SESSION_ALLOCATED record is destroyed. If a session is obtained, an MU is created, and the PERMANENT_BUFFER_POOL_ID, and LIMITED_BUFFER_POOL_ID fields are set along with the send MU.PS_TO_HS.ALLOCATE field; otherwise, these fields remain as initialized. Also, the RETURN_CODE on the ALLOCATE verb can be updated to reflect detected errors.

NOTES:
1. The resources manager returns to PS an ALLOCATE_FAILURE return code to a session allocation request when no sessions having the specified (LU name, mode name) pair are active and a condition (either temporary or permanent, as reflected in the return code) exists such that no sessions can currently be activated.

2. The resources manager returns to PS a SYNC_LEVEL_NOT_SUPPORTED return code to a session allocation request when a session having the specified (LU name, mode name) pair is active, but the synchronization level specified by the transaction program on ALLOCATE is not supported by the partner LU.

Referenced procedures, FSMs, and data structures:

RM	page 3-19
CREATE_AND_INIT_LIMITED_MU	page 5.1-30
WAIT_FOR_RM_REPLY	page 5.1-62
MU	page A-29
RCB	page A-6
GET_SESSION	page A-16
SESSION_ALLOCATED	page A-22

OBTAIN_SESSION_PROC

```
Create and initialize the GET_SESSION record and send it to RM.
Call WAIT_FOR_RM_REPLY (page 5.1-62) to receive SESSION_ALLOCATED.
Select based on the RETURN_CODE of the SESSION_ALLOCATED record:
   When OK
      Set RCB.SEND_RU_SIZE to SESSION_ALLOCATED.SEND_RU_SIZE.
      Set RCB.LIMITED_BUFFER_POOL_ID to SESSION_ALLOCATED.LIMITED_BUFFER_POOL_ID.
      Set RCB.PERMANENT_BUFFER_POOL_ID to SESSION_ALLOCATED.PERMANENT_BUFFER_POOL_ID.
      Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
      If SESSION_ALLOCATED.IN_CONVERSATION is YES then
         Set the send MU.PS_TO_HS.ALLOCATE to NO.
      Else
         Set the send MU.PS_TO_HS.ALLOCATE to YES.
   Otherwise
      Set the RETURN_CODE of the ALLOCATE verb to ALLOCATION_ERROR.
      Select based on the RETURN_CODE of the SESSION_ALLOCATED record:
         When UNSUCCESSFUL_RETRY
            Set the subcode of the ALLOCATE verb to ALLOCATION_FAILURE_RETRY.
         When UNSUCCESSFUL_NO_RETRY
            Set the subcode of the ALLOCATE verb to ALLOCATION_FAILURE_NO_RETRY.
         When SYNC_LEVEL_NOT_SUPPORTED
            Set the subcode of the ALLOCATE verb to SYNC_LEVEL_NOT_SUPPORTED_BY_LU.

Destroy SESSION_ALLOCATED record.
```

PERFORM_RECEIVE_EC_PROCESSING

FUNCTION:	This procedure processes the end-of-chain type that has been received and saved for this conversation. This procedure is called only if the end-of-chain type received is a value other than NOT_END_OF_DATA.
INPUT:	The RCB corresponding to the resource specified in the verb parameters, and RECEIVE verb parameters
OUTPUT:	The return code field of the RECEIVE verb is updated to the appropriate value. The state of FSM_CONVERSATION may be changed.

Referenced procedures, FSMs, and data structures:

PS_PROTOCOL_ERROR	page 5.0-20
FSM_CONVERSATION	page 5.1-65
RCB	page A-6

```
Select based on the following conditions:
   When the data sent by the TP is not on a logical record boundary
      Call PS_PROTOCOL_ERROR (RCB.HS_ID, X'10010000') (page 5.0-20).
      Set the RETURN_CODE of the RECEIVE verb to RESOURCE_FAILURE_NO_RETRY.
      Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
   When RCB.SYNC_LEVEL is NONE and the end-of-chain type is CONFIRM,
    PREPARE_TO_RCV_CONFIRM, or DEALLOCATE_CONFIRM
      Call PS_PROTOCOL_ERROR (RCB.HS_ID, X'10010000') (page 5.0-20).
      Set the RETURN_CODE of the RECEIVE verb to RESOURCE_FAILURE_NO_RETRY.
      Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
   Otherwise
      Select based on the end-of-chain type received:
         When CONFIRM
            Set the RETURN_CODE of the RECEIVE verb to OK.
            Set the WHAT_RECEIVED parameter of the RECEIVE verb to CONFIRM.
            Call FSM_CONVERSATION(R, CONFIRM_INDICATOR, RCB) (page 5.1-65).
         WHEN PREPARE_TO_RCV_CONFIRM
            Set the RETURN_CODE of the RECEIVE verb to OK.
            Set the WHAT_RECEIVED parameter of the RECEIVE verb to CONFIRM_SEND.
            Call FSM_CONVERSATION(R, CONFIRM_SEND_INDICATOR, RCB) (page 5.1-65).
         WHEN PREPARE_TO_RCV_FLUSH
            Set the RETURN_CODE of the RECEIVE verb to OK.
            Set the WHAT_RECEIVED parameter of the RECEIVE verb to SEND.
            Call FSM_CONVERSATION(R, SEND_INDICATOR, RCB) (page 5.1-65).
         WHEN DEALLOCATE_CONFIRM
            Set the RETURN_CODE of the RECEIVE verb to OK.
            Set the WHAT_RECEIVED parameter of the RECEIVE verb to CONFIRM_DEALLOCATE.
            Call FSM_CONVERSATION(R, CONFIRM_DEALLOCATE_INDICATOR, RCB) (page 5.1-65).
         WHEN DEALLOCATE_FLUSH
            Set the RETURN_CODE of the RECEIVE verb to DEALLOCATE_NORMAL.
            Call FSM_CONVERSATION(R, DEALLOCATE_NORMAL_RC, RCB) (page 5.1-65).
```

PERFORM_RECEIVE_PROCESSING

FUNCTION:	This procedure checks the RCB.HS_TO_PS_BUFFER_LIST receive buffer to see if any information has arrived for the conversation specified in the passed RECEIVE verb parameters and, if so, updates the verb parameters to reflect that information. Examples of the type of information that can be received include a request for confirmation, notification that the partner transaction program has deallocated the conversation, and conversation data. If no information has been received for the specified conversation, the RETURN_CODE parameter is set to UNSUCCESSFUL and control is returned to the calling procedure.
INPUT:	The RCB corresponding to the resource specified in the verb parameters, and RECEIVE verb parameters
OUTPUT:	The information is copied from the MU into the RECEIVE verb data buffer to be passed up to the TP. If the data in the MU is exhausted, the MU is freed and the next MU in the receive buffer, if present, will begin to be processed.
NOTES:	1. PS performs an optional receive check to determine if the partner LU has violated protocols by allowing the partner transaction program to invalidly truncate the logical record the program was in the process of sending (i.e., the partner transaction program issued a verb, such as CONFIRM, before completing the current logical record). Only an FMH-7 can validly be received before the current logical record is completed, in which case the FMH-7 contains sense data indicating data truncation. 2. PS performs an optional receive check to determine if the partner LU has violated the protocols by allowing the partner transaction program to issue a request for confirmation on a conversation whose SYNC_LEVEL is NONE.

Referenced procedures, FSMs, and data structures:

PERFORM_RECEIVE_EC_PROCESSING	page 5.1-38
PROCESS_FMH7_PROC	page 5.1-46
PROCESS_DATA_PROC	page 5.1-43
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
MU	page A-29
RCB	page A-6

```
Set MU_PTR to the first MU in the RCB.HS_TO_PS_BUFFER_LIST.

If the MU_PTR is not NULL or end chain indicator is not NOT_END_OF_DATA then
   Do while the MU_PTR is not NULL and posting condition is not satisfied:
      If an FMH_7 is contained in the MU then
         If no data has been copied to pass to the TP then
            Call PROCESS_FMH7_PROC(RCB, RECEIVE verb parameters) (page 5.1-46).
            Set the MU_PTR to the next MU in the RCB.HS_TO_PS_BUFFER_LIST.
      Else
         If the MU has more data to be received then
            Call PROCESS_DATA_PROC(RCB, RECEIVE verb parameters, DATA_NEEDED) (page 5.1-43).

         If the data in the MU has all been received then
            If MU.HS_TO_PS.TYPE is NOT_END_OF_DATA then
               Call buffer manager (FREE_BUFFER, buffer address) (Appendix B).
               Set MU_PTR to the next MU in the RCB.HS_TO_PS_BUFFER_LIST.
            Else
               End-of-chain type has been received; posting is satisfied.

   If no data is being returned to the TP and an FMH_7 was not processed and the end-of-chain
    type was not NOT_END_OF_DATA then
      Call PERFORM_RECEIVE_EC_PROCESSING(RCB, RECEIVE verb parameters) (page 5.1-38).
      End-of-chain type is returned to the TP.

Else (MU_PTR is NULL or end-of-chain type is not NOT_END_OF_DATA)
   Select based on the state of FSM_ERROR_OR_FAILURE (page 5.1-67):
      When CONV_FAILURE_PROTOCOL_ERROR
         Set the RETURN_CODE of the RECEIVE verb to RESOURCE_FAILURE_NO_RETRY.
         Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
```

```
      When CONV_FAILURE_SON
         Set the RETURN_CODE of the RECEIVE verb to RESOURCE_FAILURE_RETRY.
         Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
      Otherwise
         If the RECEIVE verb is a RECEIVE_IMMEDIATE verb and no data is being
          returned to the TP then
            Set the RETURN_CODE of the RECEIVE_IMMEDIATE verb to UNSUCCESSFUL.
         Else
            Set the RETURN_CODE parameter of the RECEIVE verb to OK.
```

PREPARE_TO_RECEIVE_CONFIRM_PROC

FUNCTION:	This procedure continues the processing of a PREPARE_TO_RECEIVE when TYPE = SYNC_LEVEL and the SYNC_LEVEL of the conversation is CONFIRM.
INPUT:	PREPARE_TO_RECEIVE verb parameters and the RCB corresponding to the resource specified in the PREPARE_TO_RECEIVE
OUTPUT:	Depending on the state of FSM_ERROR_OR_FAILURE, the MU.PS_TO_HS.TYPE field may be set prior to sending the MU to HS. See Notes for additional outputs.
NOTES:	1. If a CONVERSATION_FAILURE has been received from the resources manager, PS returns to the transaction program after setting the RETURN_CODE parameter of the PREPARE_TO_RECEIVE verb to RESOURCE_FAILURE.
	2. If a RECEIVE_ERROR has been received from HS, PS sends the current send MU record with the MU.PS_TO_HS.TYPE field set to PREPARE_TO_RCV_FLUSH to HS. (Any data in the RCB send buffer was purged when the RECEIVE_ERROR record was received.) PS then waits for the expected FMH-7 error message to arrive. The RETURN_CODE parameter of the CONFIRM is set based on the sense data carried in the FMH-7.
	3. If no error or failure condition has occurred, PS sends the current send MU with the MU.PS_TO_HS.TYPE field set to PREPARE_TO_RCV_CONFIRM to HS and waits for a CONFIRMED reply.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
CREATE_AND_INIT_LIMITED_MU	page 5.1-30
DEQUEUE_FMH7_PROC	page 5.1-34
RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
WAIT_FOR_CONFIRMED_PROC	page 5.1-61
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
MU	page A-29
RCB	page A-6

```
Call FSM_CONVERSATION(S, PREPARE_TO_RECEIVE_CONFIRM, RCB) (page 5.1-65).
Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).

Select based on state of FSM_ERROR_OR_FAILURE (page 5.1-67):
   When CONV_FAILURE_PROTOCOL_ERROR (see Note 1)
      Set the RETURN_CODE of PREPARE_TO_RECEIVE verb to RESOURCE_FAILURE_NO_RETRY.
      Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
   When CONV_FAILURE_SON (see Note 1)
      Set the RETURN_CODE of PREPARE_TO_RECEIVE verb to RESOURCE_FAILURE_RETRY.
      Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
```

```
When RCVD_ERROR (see Note 2)
   If a send MU buffer does not exist then
      Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
   Set MU.PS_TO_HS.TYPE to PREPARE_TO_RCV_FLUSH and send the MU record to HS.
   Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(SUSPEND_LIST containing RCB_ID)(page 5.1-51).
   If state of FSM_ERROR_OR_FAILURE (page 5.1-67) is CONV_FAILURE_SON or
    CONV_FAILURE_PROTOCOL_ERROR then
      If state of FSM_ERROR_OR_FAILURE (page 5.1-67) is CONV_FAILURE_SON then
         Set the RETURN_CODE of PREPARE_TO_RECEIVE verb to RESOURCE_FAILURE_RETRY.
      Else
         Set the RETURN_CODE of PREPARE_TO_RECEIVE to verb RESOURCE_FAILURE_NO_RETRY.
      Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
   Else
      Call DEQUEUE_FMH7_PROC(PREPARE_TO_RECEIVE verb parameters, RCB) (page 5.1-34).
When NO_REQUESTS (see Note 3)
   Set MU.PS_TO_HS.TYPE to PREPARE_TO_RCV_CONFIRM_SHORT or PREPARE_TO_RCV_CONFIRM_LONG as
    indicated by RCB.LOCKS.
   Call WAIT_FOR_CONFIRMED_PROC(PREPARE_TO_RECEIVE verb parameters, RCB) (page 5.1-61).
```

PREPARE_TO_RECEIVE_FLUSH_PROC

FUNCTION:	This procedure continues the processing of a PREPARE_TO_RECEIVE when TYPE = FLUSH, or TYPE = SYNC_LEVEL and the SYNC_LEVEL of the conversation is NONE.
INPUT:	PREPARE_TO_RECEIVE verb parameters and the RCB corresponding to the resource specified in the PREPARE_TO_RECEIVE
OUTPUT:	The RETURN_CODE is set to OK, the state of FSM_CONVERSATION is changed and an MU may be sent to HS. See below for additional output.
NOTES:	1. If a RECEIVE_ERROR has been received from HS, PS sends a SEND_DATA record with the MU.PS_TO_HS.TYPE field set to PREPARE_TO_RCV_FLUSH to HS. (Any data in the RCB send buffer was purged when the RECEIVE_ERROR record was received.) PS then waits for the expected FMH-7 error message to arrive. The RETURN_CODE parameter of the CONFIRM is set based on the sense data carried in the FMH-7.
	2. If a conversation failure has occurred, no action is taken. PS reports the error to the transaction program at a later time.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
CREATE_AND_INIT_LIMITED_MU	page 5.1-30
RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
MU	page A-29
RCB	page A-6

```
Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(empty SUSPEND_LIST) (page 5.1-51).
If the state of FSM_ERROR_OR_FAILURE (page 5.1-67) is RCVD_ERROR or NO_REQUESTS then
   If a send MU buffer does not exist then
      Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
   Set MU.PS_TO_HS.TYPE to PREPARE_TO_RECEIVE_FLUSH and send the MU record to HS.
Set the RETURN_CODE of the PREPARE_TO_RECEIVE verb to OK.
Call FSM_CONVERSATION(S, PREPARE_TO_RECEIVE_FLUSH, RCB) (page 5.1-65).
```

PROCESS_DATA_PROC

FUNCTION: This procedure handles the processing of MUs from the HS_TO_PS_BUFFER_LIST.

The procedure first checks to see if the data in the MU being processed is a PS header or a logical record having an invalid LL value, in order to take appropriate action.

If this data is not a PS header or an invalid LL, then further processing of the data in the MU occurs as described below.

INPUT: The RCB corresponding to the resource specified in the passed RECEIVE verb parameters, the MU (contained in the RCB.HS_TO_PS_BUFFER_LIST), and the RECEIVE verb parameters.

OUTPUT: The parameters of the RECEIVE_VERB are updated.

NOTES:

1. If the data in the MU being processed begins on a logical record boundary (i.e., the last data passed to the transaction program was a complete conversation record or the last remaining portion of a logical record, or no data has been passed to the transaction program since it last entered the receive state) and both bytes of the next logical record's LL field are present in the MU, data is moved from the MU parameter to the DATA parameter of the passed RECEIVE verb.

2. If the data in the MU being processed begins on a logical record boundary, but only the first byte of the next 2-byte LL field is present in the MU, this procedure checks to see if any other information has been received following the first byte of the LL. If the LL has been truncated by receipt of an FMH-7, the LL byte is placed in the DATA parameter of the passed RECEIVE verb and control is returned to the transaction program. (The FMH-7 is processed when the transaction program issues another verb.) If the LL has been truncated invalidly by receipt of information other than an FMH-7, the partner LU has committed a protocol violation and the session over which the conversation is occurring is deactivated. If no information follows the first byte of the LL, it is saved in the buffer and control is returned to the transaction program. (The first byte of the LL is not passed to the transaction program. Until the second byte of the 2-byte LL field arrives, PS does not know if the LL is associated with a logical record or with a PS header.)

3. If the data in the passed MU does not begin on a logical record boundary (i.e., part, but not all, of a logical record has already been passed to the transaction program), data is moved from the MU to the DATA parameter of the passed RECEIVE verb.

MUs from the RCB.HS_TO_PS_BUFFER_LIST are passed in, one at a time, for this procedure to copy the data from the MU into the RECEIVE_* verb DATA parameter to be returned to the transaction program.

While processing the MU, information on the location in the MU, location in the logical record, and number of bytes remaining in the current logical record is continually updated as the data is copied from the MU to the RECEIVE_* verb DATA parameter.

As the data is copied into the RECEIVE_* verb, one or more of the following RECEIVE_* verb parameters must also be set accordingly:

- WHAT_RECEIVED
- RETURN_CODE
- LENGTH of data returned

See <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u> for a complete list and definition of the possible values. Also, see Notes 1, 2, and 3 for additional information.

PROCESS_FMH7_LOG_DATA_PROC

FUNCTION:		This procedure is invoked upon encountering an FMH-7 with LOG_DATA following it in the HS_TO_PS_BUFFER_LIST. The RETURN_CODE parameter of the passed transaction program verb is set based upon the sense data carried in the FMH-7. This procedure simulates a RECEIVE_AND_WAIT verb and causes receive processing to take place. The RECEIVE_AND_WAIT processing waits for one or more logical records, which consists of the log data, to arrive from HS. If the sense data carried in the FMH-7 indicates a type of DEALLOCATE_ABEND_*, this procedure retrieves the deallocation notification from the receive buffer before returning to the transaction program.
INPUT:		The RCB corresponding to the resource to which the FMH-7 applies, the FMH_SENSE_DATA associated with the error, and the transaction program verb currently being processed
OUTPUT:		The RETURN_CODE parameter of the verb is set based upon the sense data carried in the passed FMH-7. The one or more logical records containing the Error Log GDS variable are placed (minus the LL and GDS ID fields) in the system error log of the local LU.
NOTES:	1.	This error occurs when the FMH-7 specifies that log data follows, but either no log data is present, or the logical record containing the log data is invalidly truncated by receipt of a CONFIRM (for example). If the error that occurred is that the log data was invalidly truncated, the error has already been detected by the PERFORM_RECEIVE_PROCESSING procedure, which has already appropriately set the return code of the current verb to reflect this error.
	2.	When the sense data in the FMH-7 indicates a type of DEALLOCATE_ABEND_*, a deallocation notification is expected. If this expected notification is not received, a protocol violation has occurred. The procedure GET_DEALLOCATE_FROM_HS performs the appropriate processing, which includes placing the conversation in END_CONV state.

Referenced procedures, FSMs, and data structures:

GET_DEALLOCATE_FROM_HS	page 5.1-35
PS_PROTOCOL_ERROR	page 5.0-20
SET_FMH7_RC	page 5.1-59
RECEIVE_AND_TEST_POSTING	page 5.1-50
FSM_ERROR_OR_FAILURE	page 5.1-67
RCB	page A-6

GET THE ERROR LOG DATA

```
Create a RECEIVE_AND_WAIT and initialize as follows:
   Set RECEIVE_AND_WAIT.RESOURCE to RCB.RCB_ID.
   Set RECEIVE_AND_WAIT.FILL to LL.
   Set RECEIVE_AND_WAIT.MAX_LENGTH to X'7FFF'.
   Set RECEIVE_AND_WAIT.DATA to NULL.
Call RECEIVE_AND_TEST_POSTING(RCB, RECEIVE_AND_WAIT verb parameters) (page 5.1-50) to
 receive the log data following the FMH_7.
If the RETURN_CODE of the RECEIVE_AND_WAIT verb is OK and the WHAT_RECEIVED indicator
 is DATA_COMPLETE then
   If the GDS ID is X'12E1' then
      Record the log data to the system log.
   Else (see Note 1.)
      Record the error receiving the LOG_DATA to the system log.
      Call PS_PROTOCOL_ERROR(RCB.HS_ID, X'1008201D') (page 5.0-20).
      Call FSM_ERROR_OR_FAILURE SIGNAL(CONV_FAIL_PROTOCOL) (page 5.1-67).
```

```
Else (see Note 1.)
   If the RETURN_CODE of the RECEIVE_AND_WAIT verb is RESOURCE_FAILURE_RETRY then
      Record a SON error occurred receiving log data to the system log.
   Else
     Record a PROTOCOL_ERROR occurred to the system log.
Destroy the created RECEIVE_AND_WAIT verb.
```

Set the states of the FSMs

```
If the passed sense data is X'08640000', X'08640001', or X'8640002' then (see Note 2.)
   If the state of FSM_ERROR_OR_FAILURE is NO_REQUESTS then
      Call GET_DEALLOCATE_FROM_HS(verb parameters, RCB) (page 5.1-35).
   Call SET_FMH7_RC(RCB, FMH_7 sense data, verb parameters) (page 5.1-59).
Else
   Select based on the state of FSM_ERROR_OR_FAILURE (page 5.1-67):
      When CONV_FAILURE_PROTOCOL_ERROR
         Call SET_FMH7_RC(RCB, FMH_7 sense data, verb parameters) (page 5.1-59).
         Call FSM_ERROR_OR_FAILURE (page 5.1-67) and pass it
          a CONV_FAIL_PROTOCOL signal.

      When CONV_FAILURE_SON
         Call SET_FMH7_RC(RCB, FMH_7 sense data, verb parameters) (page 5.1-59).
         Call FSM_ERROR_OR_FAILURE (page 5.1-67) and pass it
          a CONV_FAIL_SON signal.

      Otherwise
         Call SET_FMH7_RC(RCB, FMH_7 sense data, verb parameters) (page 5.1-59).

Set all the RCB receive processing fields to their initial values (processing begins
 anew following receipt of an FMH-7).
```

PROCESS_FMH7_PROC

FUNCTION:	This procedure is invoked upon encountering an FMH-7 in the HS_TO_PS_BUFFER_LIST. The RETURN_CODE parameter of the passed transaction program verb is set based upon the sense data carried in the FMH-7. If the FMH-7 indicates that log data follows, this procedure simulates a RECEIVE_AND_WAIT verb and causes receive processing to take place. The RECEIVE_AND_WAIT processing waits for a logical record, which consists of the log data, to arrive from HS. If the sense data carried in the FMH-7 indicates a type of DEALLOCATE_ABEND_* this procedure retrieves the deallocation notification from the receive buffer before returning to the transaction program.
INPUT:	The RCB corresponding to the resource to which the FMH-7 applies, the MU containing the received FMH-7 (contained in the RCB.HS_TO_PS_BUFFER_LIST), and the transaction program verb currently being processed
OUTPUT:	The RETURN_CODE parameter of the verb is set, based upon the sense data carried in the passed FMH-7; if log data follows the FMH-7, PS retrieves the logical record containing the Error Log GDS variable and places it (minus the LL and GDS ID fields) in the system error log of the local LU.
NOTES:	1. Logical record processing begins anew following receipt of an FMH-7. 2. When the sense data in the FMH-7 indicates a type of DEALLOCATE_ABEND_*, a deallocation notification is expected. If this expected notification is not received, a protocol violation has occurred. The procedure GET_DEALLOCATE_FROM_HS performs the appropriate processing, which includes placing the conversation in END_CONV state.

Referenced procedures, FSMs, and data structures:

GET_DEALLOCATE_FROM_HS	page 5.1-35
PROCESS_FMH7_LOG_DATA_PROC	page 5.1-44
PS_PROTOCOL_ERROR	page 5.0-20
SET_FMH7_RC	page 5.1-59
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
MU	page A-29
RCB	page A-6

Set the MU_PTR to the first entry in the HS_TO_PS_BUFFER_LIST.

Validate the FMH-7.

```
As an implementation-dependent option perform receive checks of the FMH-7.
If an error is found then
   Call PS_PROTOCOL_ERROR(RCB.HS_IS, X'nnnnnnnn') (page 5.0-20) with X'nnnnnnnn' set to:
    X'10086000' (Request Error--FMH Length Incorrect) or
    X'1008200E' (Request Error--Invalid Concatenation Bit).
   Set the RETURN_CODE parameter of the verb to RESOURCE_FAILURE_NO_RETRY.
   Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
   Set RCB.POST_CONDITIONS.MAX_LENGTH to 0.
```

Processing necessary as a result of the FMH-7

```
Else
   Set all the RCB receive processing fields to their initial values (see Note 1).
   If the data in the MU has all been received then
      Remember if the FMH-7 has log data present and the sense data value.
      Save the end-of-chain type for later processing.
      Call buffer manager (FREE_BUFFER, buffer address) (Appendix B).
   Set the MU_PTR to the first entry in the HS_TO_PS_BUFFER_LIST.
```

Get the log data if present.

```
If the FMH_7 indicates log data is present then
   Call PROCESS_FMH7_LOG_DATA_PROC(RCB, FMH_7 sense data, verb parameters) (page 5.1-44).
Else
   If FMH_SENSE_DATA is X'08640000', X'08640001', or X'08640002' then (see Note 2.)
      If the state of FSM_ERROR_OR_FAILURE is NO_REQUESTS then
         Call GET_DEALLOCATE_FROM_HS(verb parameters, RCB) (page 5.1-35).
         Call SET_FMH7_RC(RCB, FMH_7 sense data, verb parameters) (page 5.1-59).
      Else
         Select based on the state of FSM_ERROR_OR_FAILURE (page 5.1-67):
            When CONV_FAILURE_PROTOCOL_ERROR
               Call SET_FMH7_RC(RCB, FMH_7 sense data, verb parameters) (page 5.1-59).
               Call FSM_ERROR_OR_FAILURE (page 5.1-67) and pass
                it a CONV_FAIL_PROTOCOL signal.
            When CONV_FAILURE_SON
               Call SET_FMH7_RC(RCB, FMH_7 sense data, verb parameters) (page 5.1-59).
               Call FSM_ERROR_OR_FAILURE (page 5.1-67) and pass
                it a CONV_FAIL_SON signal.
            Otherwise
               Call SET_FMH7_RC(RCB, FMH_7 sense data, verb parameters) (page 5.1-59).
```

RCB_ALLOCATED_PROC

FUNCTION:	This procedure performs further processing of an ALLOCATE request. It is invoked when PS receives an RCB_ALLOCATED record from the resources manager. This procedure sets the RETURN_CODE parameter of the ALLOCATE verb based upon the return code field of the RCB_ALLOCATED record. If the return code is OK, it finishes initializing the new RCB (i.e., those fields not already initialized by RM). In addition, if the RETURN_CONTROL parameter of ALLOCATE is WHEN_SESSION_ALLOCATED, PS requests that a session be obtained for this conversation. If the return code in RCB_ALLOCATED is OK, then RM has created an RCB and entered a RESOURCE entry in the RESOURCE_LIST for the appropriate TCB.
INPUT:	RCB_ALLOCATED record and ALLOCATE verb parameters
OUTPUT:	The ALLOCATE return code and RESOURCE are set. If no errors occur obtaining the session, then PS creates an FMH-5 Attach header and either stores it in the send buffer in the RCB or optionally flushes it, which requires setting the MU.PS_TO_HS.TYPE field to FLUSH. An MU is created and MU.PS_TO_HS.ALLOCATE set to YES if ALLOCATE(IMMEDIATE) is specified.
NOTES:	1. If RETURN_CONTROL = IMMEDIATE, RM has allocated both an RCB and a session as a result of receiving ALLOCATE_RCB from PS. 2. A return code of UNSUCCESSFUL in reply to an ALLOCATE (RETURN_CONTROL = IMMEDIATE) indicates that no first-speaker half-sessions are currently available. 3. The resources manager returns to PS a SYNC_LEVEL_NOT_SUPPORTED return code to a session allocation request when a session having the specified (LU name, mode name) pair is active, but the synchronization level specified by the transaction program on ALLOCATE is not supported by the partner LU.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
CREATE_AND_INIT_LIMITED_MU	page 5.1-30
OBTAIN_SESSION_PROC	page 5.1-37
FSM_CONVERSATION	page 5.1-65
RCB_ALLOCATED	page A-21
RCB	page A-6
MU	page A-29
TCB	page A-9

```
Select based on the RETURN_CODE of the RCB_ALLOCATED record:
   When OK
      Set the RETURN_CODE of the ALLOCATE verb to OK.
      Find the RCB for the conversation identified by the RCB_ID in the RCB_ALLOCATED record.
      Set the RESOURCE parameter of the ALLOCATE verb to RCB identifier.
      Set the fields in the RCB to their initial values.

      If the RETURN_CONTROL parameter of the ALLOCATE verb is IMMEDIATE then (see Note 1)
         Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).

         Set MU.PS_TO_HS.ALLOCATE to YES.
      Else
         Call OBTAIN_SESSION_PROC(RCB, ALLOCATE verb parameters) (page 5.1-37).
      If the RETURN_CODE of the ALLOCATE verb is OK then
         Build an FMH-5 Attach (see SNA Formats) with the data in ALLOCATE.
         If the FMH-5 is to be flushed (as an implementation-dependent option) then
            Send the MU record to HS.
      Else (error found during ALLOCATE processing)
         Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-65).
   When UNSUCCESSFUL (see Note 2)
      Set the RETURN_CODE of the ALLOCATE verb to UNSUCCESSFUL.
   When SYNC_LEVEL_NOT_SUPPORTED
      Find the RCB for the conversation identified by the RCB_ID in the RCB_ALLOCATED record.
      Initialize the allocated RCB.
      Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-65).
      Set the RETURN_CODE of the ALLOCATE verb to ALLOCATION_ERROR with a subcode of
       SYNC_LEVEL_NOT_SUPPORTED_BY_LU (see Note 3).
```

RECEIVE_AND_TEST_POSTING

FUNCTION:	This procedure transfers data from the received MUs into the RECEIVE_AND_WAIT data buffer while checking for posting to be satisfied.
INPUT:	RCB of the conversation, and the RECEIVE_AND_WAIT verb
OUTPUT:	Buffer area of the verb is filled with the requested amount of data. If data is to be returned to the TP, RECEIVE_AND_WAIT.MAX_LENGTH is set to the amount of data being returned. RECEIVE_AND_WAIT.RETURN_CODE and RECEIVE_AND_WAIT.WHAT_RECEIVED are initialized. Also, the FSM_POST undergoes state transitions, along with updates to the RCB.POST_CONDITIONS.MAX_LENGTH and RCB.POST_CONDITIONS.FILL fields.

Referenced procedures, FSMs, and data structures:

PERFORM_RECEIVE_PROCESSING	page 5.1-40
RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
TEST_FOR_POST_SATISFIED	page 5.1-60
FSM_POST	page 5.1-68
RCB	page A-6

TEST POSTING

```
Call FSM_POST (page 5.1-68) and pass it a POST_ON_RECEIPT signal.
Set RCB.POST_CONDITIONS.FILL to the FILL parameter on the RECEIVE_AND_WAIT verb.
Set RCB.POST_CONDITIONS.MAX_LENGTH to the MAX_LENGTH parameter on the RECEIVE_AND_WAIT verb.
Call TEST_FOR_POST_SATISFIED(RCB) (page 5.1-60).
Call PERFORM_RECEIVE_PROCESSING(RCB, RECEIVE_AND_WAIT verb parameters) (page 5.1-40).

If the state of FSM_POST is PEND_POSTED then
   Do while the state of FSM_POST is PEND_POSTED.
      Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(SUSPEND_LIST containing RCB_ID) (page 5.1-51).
      Call TEST_FOR_POST_SATISFIED(RCB) (page 5.1-60).
      Call PERFORM_RECEIVE_PROCESSING(RCB, RECEIVE_AND_WAIT verb parameters) (page 5.1-40).

Set RECEIVE_AND_WAIT.MAX_LENGTH to indicate the amount of data returned to the TP.
Call FSM_POST (page 5.1-68) and pass it a RECEIVE_IMMEDIATE signal.
```

RECEIVE_RM_OR_HS_TO_PS_RECORDS

FUNCTION: This procedure receives records from RM and all HS processes and updates the appropriate RCB. If SUSPEND_LIST is not empty, this procedure waits until at least one record associated with an RCB in SUSPEND_LIST is received.

All records passed up from HS to PS will be processed at this time. Records from HS that do not have RCB associated with them are destroyed or freed by the buffer manager.

CONFIRMED and RSP_TO_REQUEST_TO_SEND are not received in this procedure. When a TP is expecting one of these responses, PS does not return control to the TP until the response is received in the LU. This processing is done in seperate procedures.

INPUT: SUSPEND_LIST containing RCB_IDs of conversations awaiting incoming records.

OUTPUT: Records received from HS are stored in the HS_TO_PS_BUFFER_LIST for the appropriate conversation.

NOTES:

1. CONVERSATION_FAILURE is the only possible record that can arrive from RM. Other records from RM are received by other procedures when a reply is expected from RM.
2. This "continuous purging" of records is required by PS for records received from HS that do not have an RCB associated with them.
3. This "continuous purging" of records is required by PS for records received from HS after receipt of a PROTOCOL_VIOLATION or SON record.

Referenced procedures, FSMs, and data structures:

CONVERSATION_FAILURE_PROC	page 5.1-29
PS_PROTOCOL_ERROR	page 5.0-20
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
RCB	page A-6
CONVERSATION_FAILURE	page A-21
CONFIRMED	page A-10
RECEIVE_ERROR	page A-10
REQUEST_TO_SEND	page A-10
RSP_TO_REQUEST_TO_SEND	page A-11
MU	page A-29

```
Set MORE_RECORDS to TRUE.
If the SUSPEND_LIST is empty then
   Set SUSPEND_FLAG to NO_SUSPEND.
Else
   Set SUSPEND_FLAG to SUSPEND.

Do while MORE_RECORDS is TRUE:
   Select based on the value of SUSPEND_FLAG:
      When SUSPEND
         Find record in RM_TO_PS_Q or HS_TO_PS_Q and SUSPEND.
      When NO_SUSPEND
         Find record in RM_TO_PS_Q or HS_TO_PS_Q.

   If a record is found then
      If the record is a CONVERSATION_FAILURE then
         Remove the CONVERSATION_FAILURE record from the RM_TO_PS_Q (see Note 1).
         Find the RCB for the conversation identified by the RCB_ID parameter.

         If the RCB is found then
            Call CONVERSATION_FAILURE_PROC(CONVERSATION_FAILURE) (page 5.1-29).
         Else
            Destroy the CONVERSATION_FAILURE record.
      Else
         Remove the record from the HS_TO_PS_Q.
         Select based on record TYPE:
            When REQUEST_TO_SEND
               Find RCB in the RCB_LIST for the BRACKET_ID specified.
               If the RCB is found then
                  Set RCB.RQ_TO_SEND_RCVD to YES.
               Destroy the REQUEST_TO_SEND record.

            When RECEIVE_ERROR
               Find RCB in the RCB_LIST for the BRACKET_ID specified.
               If the RCB is found then
                  Call FSM_ERROR_OR_FAILURE(RECEIVE_ERROR, RCB);
               Destroy the RECEIVE_ERROR record.

            When RSP_TO_REQUEST_TO_SEND, CONFIRMED
               Destroy the record (see Note 3).

            When MU
                Find the RCB for the conversation with the BRACKET_ID specified in the MU.
                If the RCB for this conversation is found then
                   If the state of FSM_CONVERSATION (page 5.1-65) is RCV_STATE or
                    the state of FSM_ERROR_OR_FAILURE (page 5.1-67) is RCVD_ERROR then
                      Add the MU to the RCB.HS_TO_PS_BUFFER_LIST.
                   Else
                      Call buffer manager (FREE BUFFER, buffer address) (Appendix B).
                      If the state of FSM_CONVERSATION is END_CONV then
                         Call PS_PROTOCOL_ERROR(RCB.HS_ID, X'20040000') (page 5.0-20).
                Else (see Note 2)
                   Call buffer manager (FREE BUFFER, buffer address) (Appendix B).

      If SUSPEND_FLAG is SUSPEND & the RCB ID for the record just
processed was
        found in the SUSPEND_LIST then
            Set SUSPEND_FLAG to NO_SUSPEND.
   Else (no record found)
      Set MORE_RECORDS to FALSE.
```

SEND_CONFIRMED_PROC

FUNCTION:	This procedure creates a CONFIRMED MU and sends it to HS.
INPUT:	The RCB associated with the half-session to which the CONFIRMED is to be sent
OUTPUT:	A CONFIRMED MU sent to HS

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
MU	page A-29
RCB	page A-6

```
Call buffer manager (GET_BUFFER, permanent buffer pool ID, no wait) to create an
 MU for the CONFIRMED; specify the buffer size to be CONFIRMED_MAX_LEN plus the length
 of the MU overhead.
If a permanent buffer is not available then
   Call buffer manager (GET_BUFFER, demand, buffer size, no wait) to create an  MU for
   the CONFIRMED; specify the buffer size to be CONFIRMED_MAX_LEN plus the length of
    the MU overhead.

Set MU.HEADER_TYPE to PS_TO_HS.
Set MU.PS_TO_HS.BRACKET_ID to RCB.BRACKET_ID.
Set MU.PS_TO_HS.PS_TO_HS_VARIANT to CONFIRMED.

Send this CONFIRMED record to HS.
```

SEND_DATA_BUFFER_MANAGEMENT

FUNCTION:	This procedure determines if there is enough data to be sent to HS. PS continues to send data to HS until the amount of data remaining to be sent is less than or equal to the maximum send RU size, in which case PS stores the data in the MU in the RCB until more data is issued by the transaction program or the buffer is forced to be sent (e.g., FLUSH, CD) to the partner. If the data in the buffer is exactly equal to the maximum send RU size, PS stores the data to be sent later.
INPUT:	Data to be sent to HS, and the RCB corresponding to the resource specified in the current TRANSACTION_PGM_VERB
OUTPUT:	If enough data has been sent by the transaction program, one or more MUs are sent to HS. Otherwise, the data is stored in the MU in the RCB to be sent at a later time. Data is copied into the send MU.RU field while the send MU.DCF field is incremented to indicate the amount of data present in the send MU.
NOTES:	1. After the MU has been completely filled with data, the presence of additional data determines if the MU will be sent to HS. If no more data is present, it will be held by PS until more data arrives or the direction of the conversation forces the MU to be sent. In the latter case, all information (e.g., CONFIRM) can be stored in the RH bits. 2. Additional MUs are not requested unless data is present to store in the MU. This will prevent utilizing storage for the MU until it is absolutely necessary.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
CREATE_AND_INIT_LIMITED_MU	page 5.1-30
RCB	page A-6
MU	page A-29

```
If a send MU buffer doesn't exist then
   Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
Else
   If the send MU is full and there is more data to send (see Note 1) then
      Send the MU buffer to HS.
      Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
Do while there is more data to send:
   Copy the data into the MU record.
      If the MU is full and there is more data to send then
         Send the MU buffer to HS.
         If there is more data to send (see Note 2) then
            Call CREATE_AND_INIT_LIMITED_MU(RCB, created MU) (page 5.1-30).
      Else (MU not full or no more data)
         Save the MU to send later (see Note 1).
```

SEND_ERROR_DONE_PROC

FUNCTION:	This procedure performs further processing of the SEND_ERROR verb. It creates an FMH-7 record and selects the sense data to be inserted in the FMH-7 based upon the type of SEND_ERROR, the state of the conversation, and whether the outgoing logical record is complete. If the transaction program is in send state and has completed the current logical record, sense data indicating that no truncation of data has taken place is inserted into the FMH-7. If the transaction program is in send state and has not completed the current logical record, sense data indicating data truncation has occurred is inserted into the FMH-7. Finally, if the transaction program is in receive state, sense data indicating that data sent by the partner transaction program is being purged by the half-session is inserted into the FMH-7. Sense data X'08890000' and X'08890100' have either of two meanings, depending upon whether the transaction program is in send or receive state.
INPUT:	SEND_ERROR verb parameters and the RCB corresponding to the resource specified in the SEND_ERROR
OUTPUT:	An FMH-7 is created and stored in the RCB send buffer. If any log data is associated with the SEND_ERROR, PS creates an Error Log GDS variable (see <u>SNA Formats</u>) and stores the GDS variable in the RCB send buffer following the FMH-7. PS also places the GDS variable (minus the LL and GDS ID fields) in the system error log at the local LU. PS returns to the transaction program with the RETURN_CODE parameter in the SEND_ERROR set to OK.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
SEND_DATA_BUFFER_MANAGEMENT	page 5.1-54
FSM_CONVERSATION	page 5.1-65
RCB	page A-6
MU	page A-29

```
Select based on the following conditions:
   When TYPE parameter of SEND_ERROR verb is PROG and state of FSM_CONVERSATION
    (page 5.1-65) is SEND_STATE
      If data sent by the TP is at a logical record boundary then
         Set SENSE_DATA to X'08890000'.
      Else
         Set SENSE_DATA to X'08890001'.
   When TYPE parameter of SEND_ERROR verb is PROG and state of FSM_CONVERSATION
    (page 5.1-65) is RCV_STATE, RCVD_CONFIRM, RCVD_CONFIRM_SEND, or RCVD_CONFIRM_DEALL
      Set SENSE_DATA to X'08890000'.
   When TYPE of SEND_ERROR is SVC and state of FSM_CONVERSATION (page 5.1-65) is SEND_STATE
      If data sent by the TP is at a logical record boundary then
         Set SENSE_DATA to X'08890100'.
      Else
         Set SENSE_DATA to X'08890101'.
   When TYPE parameter of SEND_ERROR is SVC and state of FSM_CONVERSATION (page 5.1-65) is
    RCV_STATE, RCVD_CONFIRM, RCVD_CONFIRM_SEND, or RCVD_CONFIRM_DEALL
      Set SENSE_DATA to X'08890100'.
If LOG_DATA parameter of SEND_ERROR is not null then
   Create an FMH-7 indicating that LOG_DATA will be present.
   Create Error log GDS variable with the LOG_DATA.
   Call SEND_DATA_BUFFER_MANAGEMENT (Error log GDS variable, RCB) (page 5.1-54).
    (See SNA Formats for Error log GDS format.)
   Log the error in the system error log.
Else
   Create an FMH-7 indicating that LOG_DATA will not be present.

If FLUSH verb is not implemented or the FMH-7 is to be flushed immediately
 (as an implementation-dependent option) then
   Send the MU record to HS.

Set the RETURN_CODE of the SEND_ERROR verb to OK.
```

SEND_ERROR_IN_RECEIVE_STATE

FUNCTION:	This procedure if invoked when the transaction program issues a SEND_ERROR for a conversation that is in the RECEIVE state. Further processing of the SEND_ERROR is dependent upon what information, if any, has been received from HS and stored in the HS_TO_PS_BUFFER_LIST, as described below.
INPUT:	SEND_ERROR verb parameters and the RCB corresponding to the resource specified in the SEND_ERROR record
OUTPUT:	See below.
NOTES:	1. If an MU with the MU.HS_TO_PS.TYPE parameter set to DEALLOCATE has been received from HS, PS returns to the transaction program after setting the RETURN_CODE parameter of the SEND_ERROR to DEALLOCATE_NORMAL.
	2. If the first MU in the RCB.HS_TO_PS_BUFFER_LIST is not a DEALLOCATE, or if the RCB.HS_TO_PS_BUFFER_LIST is empty, PS sends a SEND_ERROR record to HS. PS then creates an FMH-7 and stores it in the RCB send buffer.

Referenced procedures, FSMs, and data structures:

SEND_ERROR_TO_HS_PROC	page 5.1-58
WAIT_FOR_SEND_ERROR_DONE_PROC	page 5.1-64
FSM_CONVERSATION	page 5.1-65
MU	page A-29
RCB	page A-6

```
Set MU_PTR to the first entry in the RCB.HS_TO_PS_BUFFER_LIST.

If the end-of-chain type for this conversation is DEALLOCATE_FLUSH then (See Note 1)
   If MU_PTR is not NULL then
      Call buffer manager (FREE_BUFFER, buffer address) (Appendix B).
   Set the RETURN_CODE of the SEND_ERROR verb to DEALLOCATE_NORMAL.
   Call FSM_CONVERSATION(R, DEALLOCATE_NORMAL_RC, RCB) (page 5.1-65).
Else (See Note 2)
   Call SEND_ERROR_TO_HS_PROC(RCB) (page 5.1-58).
   Call WAIT_FOR_SEND_ERROR_DONE_PROC(SEND_ERROR verb parameters, RCB) (page 5.1-64).
```

SEND_ERROR_IN_SEND_STATE

FUNCTION:	This procedure is invoked when the transaction program issues a SEND_ERROR verb for a conversation that is in the SEND state. If the state of FSM_ERROR_OR_FAILURE indicates that no RECEIVE_ERROR record has been received from HS, any data in PS's send buffer is sent to HS and an FMH-7 is created and stored in a newly created MU. If the state of FSM_ERROR_OR_FAILURE indicates that a RECEIVE_ERROR record has been received from HS, PS sends an MU with the MU.PS_TO_HS.TYPE field set to PREPARE_TO_RCV_FLUSH to HS. (Any data in the RCB send buffer was purged when the RECEIVE_ERROR record was received.) PS then waits for the expected FMH-7 to arrive. The RETURN_CODE parameter of the SEND_ERROR is set based upon the sense data carried in the FMH-7.
INPUT:	SEND_ERROR verb parameters and the RCB corresponding to the resource specified in the SEND_ERROR
OUTPUT:	Any data in PS's buffer is sent to HS and an FMH-7 is created and stored in the RCB. Send processing begins anew after sending an FMH-7; therefore, the send processing fields of the RCB are reset to their initial values.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
RECEIVE_RM_OR_HS_TO_PS_RECORDS	page 5.1-51
SEND_ERROR_DONE_PROC	page 5.1-55
DEQUEUE_FMH7_PROC	page 5.1-34
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
MU	page A-29
RCB	page A-6

```
If the state of FSM_ERROR_OR_FAILURE is NO_REQUESTS then
   If a send MU buffer is present then
      Send the MU record to HS.

   Call FSM_CONVERSATION(S, SEND_ERROR, RCB) (page 5.1-65).
   Call SEND_ERROR_DONE_PROC(SEND_ERROR verb parameters, RCB) (page 5.1-55).
Else (the state is RCVD_ERROR)
   Set MU.PS_TO_HS.TYPE to PREPARE_TO_RCV_FLUSH and send the MU record to HS.
   Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(SUSPEND_LIST containing RCB_ID) (page 5.1-51).
   If the state of FSM_ERROR_OR_FAILURE is CONV_FAILURE_SON or
    CONV_FAILURE_PROTOCOL_ERROR then
      If the state of FSM_ERROR_OR_FAILURE is CONV_FAILURE_SON then
         Set the RETURN_CODE of the SEND_ERROR verb to RESOURCE_FAILURE_RETRY.
      Else
         Set the RETURN_CODE of the SEND_ERROR verb to RESOURCE_FAILURE_NO_RETRY.
      Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
   Else
      Call DEQUEUE_FMH7_PROC(SEND_ERROR verb parameters, RCB) (page 5.1-34).
Set the RCB send processing fields to their initial values.
```

SEND_ERROR_TO_HS_PROC

FUNCTION:	This procedure creates a SEND_ERROR MU and sends it to HS.
INPUT:	The RCB associated with the HS to which the SEND_ERROR MU is to be sent
OUTPUT:	A SEND_ERROR is created to be sent to HS

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
MU	page A-29
RCB	page A-6

Call buffer manager (GET_BUFFER, buffer size) to create an MU for the SEND_ERROR; specify the buffer size to be SEND_ERROR_MAX_LEN plus the length of the MU overhead.

Set MU.HEADER_TYPE to PS_TO_HS.
Set MU.PS_TO_HS.BRACKET_ID to RCB.BRACKET_ID.
Set MU.PS_TO_HS.PS_TO_HS_VARIANT to SEND_ERROR.

Send this SEND_ERROR record to HS.

SEND_REQUEST_TO_SEND_PROC

FUNCTION:	This procedure creates an MU containing a REQUEST_TO_SEND and sends it to HS.
INPUT:	The RCB associated with the half-session to which the REQUEST_TO_SEND is to be sent
OUTPUT:	A REQUEST_TO_SEND is created to be sent to HS.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
MU	page A-29
RCB	page A-6

Call buffer manager (GET_BUFFER, buffer size) to create an MU for the REQUEST_TO_SEND; specify the buffer size to be REQUEST_TO_SEND_MAX_LEN plus the length of the MU overhead.

Set MU.HEADER_TYPE to PS_TO_HS.
Set MU.PS_TO_HS.BRACKET_ID to RCB.BRACKET_ID.
Set MU.PS_TO_HS.PS_TO_HS_VARIANT to REQUEST_TO_SEND.

Send this REQUEST_TO_SEND record to HS.

SET_FMH7_RC

FUNCTION:	This procedure sets the RETURN_CODE parameter of the passed transaction program verb based upon the passed sense data.
INPUT:	The RCB corresponding to the resource to which the FMH-7 applies, the received FMH-7 sense data, and the transaction program verb parameters currently being processed
OUTPUT:	The RETURN_CODE parameter of the verb is set, based upon the sense data passed from the received FMH-7.
NOTE:	When the sense data in the FMH-7 indicates an allocation error, a deallocation notification is expected. If this expected notification is not received, a protocol violation has occurred. The procedure GET_DEALLOCATE_FROM_HS performs the appropriate processing, which includes placing the conversation in END_CONV state.

Referenced procedures, FSMs, and data structures:

PS_PROTOCOL_ERROR	page 5.0-20
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
RCB	page A-6


```
Select based on the sense data in FMH-7:
   When ALLOCATION_ERROR code
      Get the DEALLOCATE from the RCB.HS_TO_PS_BUFFER_LIST.
      If the state of FSM_CONVERSATION is not END_CONV then
        Set RETURN_CODE parameter of the verb to the corresponding value (see SNA Formats to
          find the value corresponding to a given sense data).
        Call FSM_CONVERSATION(R, ALLOCATION_ERROR, RCB) (page 5.1-65).
   When RESOURCE_FAILURE_NO_RETRY
      Set RETURN_CODE parameter of the TP verb to RESOURCE_FAILURE_NO_RETRY.
      Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
   When PROG_ERROR_NO_TRUNC or PROG_ERROR_PURGING
      If state of FSM_ERROR_OR_FAILURE (page 5.1-67) is  RCVD_ERROR then
         Set RETURN_CODE parameter of the verb to PROG_ERROR_PURGING.

      Else
         Set RETURN_CODE parameter of the verb to PROG_ERROR_NO_TRUNC.
      Call FSM_CONVERSATION(R, PROGRAM_ERROR_RC, RCB) (page 5.1-65).
   When PROG_ERROR_TRUNC
      Set RETURN_CODE parameter of the verb to PROG_ERROR_TRUNC.
      Call FSM_CONVERSATION(R, PROGRAM_ERROR_RC, RCB) (page 5.1-65).
   When SVC_ERROR_NO_TRUNC or SVC_ERROR_PURGING
      If state of FSM_ERROR_OR_FAILURE (page 5.1-67) is RCVD_ERROR then
         Set RETURN_CODE parameter of the verb to SVC_ERROR_PURGING.

      Else
         Set RETURN_CODE parameter of the verb to SVC_ERROR_NO_TRUNC.
      Call FSM_CONVERSATION(R, SERVICE_ERROR_RC, RCB) (page 5.1-65).
   When SVC_ERROR_TRUNC
      Set RETURN_CODE parameter of the verb to SVC_ERROR_TRUNC.
      Call FSM_CONVERSATION(R, SERVICE_ERROR_RC, RCB) (page 5.1-65).
   When DEALLOCATE_ABEND
      Set RETURN_CODE parameter of the verb to DEALLOCATE_ABEND_PROG, DEALLOCATE_ABEND_SVC,
       DEALLOCATE_ABEND_TIMER, or to DEALLOCATE_ABEND_RC
       as shown in SNA Formats under X'0864' Sense Code.
      Call FSM_CONVERSATION(R, DEALLOCATE_ABEND_RC, RCB).
   Otherwise (as an implementation-dependent option):
      Call PS_PROTOCOL_ERROR(RCB.HS_ID, FMH-7 sense data) (page 5.0-20).
      Set RETURN_CODE parameter to RESOURCE_FAILURE_NO_RETRY.
      Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
```

TEST_FOR_POST_SATISFIED

FUNCTION:	This procedure tests whether the post conditions specified in the RCB have been satisfied.
INPUT:	The RCB corresponding to the resource to be tested
OUTPUT:	The state of FSM_POST is set to POSTED if the post conditions are satisfied.

Referenced procedures, FSMs, and data structures:

FSM_POST	page 5.1-68
MU	page A-29

Testing for posting involves examining the data (logical records) in the received MUs to see if the data will satisfy the conditions specified in the RECEIVE_* verb. Receipt of any of the following can cause posting to be satisfied:

- End-of-Chain type received
- Conversation Failure record from RM--Session-Outage Notification (SON)
- Conversation Failure record from RM--Protocol Violation
- FMH-7
- Complete logical record with FILL=LL
- Remainder of partial returned logical record with FILL=LL
- Enough data in the buffer to satisfy the FILL=BUFFER condition
- An invalid LL in the received data

During the testing process, the data in the received MUs is traversed one logical record at a time until any one of the conditions listed above is recognized. Information on the location in the MU, location in the logical record, and number of bytes remaining in the current logical record is continually updated as the data is examined.

A logical record is preceded by a 2-byte length field. These length bytes are not passed to the transaction program until both bytes are present in the receive buffer. If only the first byte is in the receive buffer, it is saved until the second byte arrives. With both bytes present, the length field is checked for validity. While waiting for the second length byte to arrive, a number of conditions can validly truncate the logical record, i.e., receipt of:

- Conversation Failure--Session-Outage Notification (SON)
- Conversation Failure--Protocol Violation
- FMH-7

WAIT_FOR_CONFIRMED_PROC

FUNCTION:	This procedure is invoked after an MU indicating CONFIRM has been sent to HS and a CONFIRMED record is expected in reply. HS can send other records to PS while PS is waiting for the expected CONFIRMED record. Appropriate action is taken, depending upon the record received (see below).
INPUT:	The transaction program verb that caused the CONFIRM indicator to be sent to HS, and the RCB corresponding to the resource specified in the transaction program verb
OUTPUT:	See below.
NOTES:	1. If a REQUEST_TO_SEND record is received, PS stores that information in the RCB to be relayed to the transaction program at a later time, and continues to wait for the expected CONFIRMED record. 2. If a RECEIVE_ERROR record is received, PS waits for the FMH-7 corresponding to the RECEIVE_ERROR to arrive from HS. The RETURN_CODE of the passed transaction program verb is set based upon the sense data carried in the FMH-7. Control is then returned to the transaction program. 3. If the expected CONFIRMED is received, PS returns control to the transaction program. 4. If the transaction program has issued a DEALLOCATE (TYPE = SYNC_LEVEL) and the SYNC_LEVEL of the conversation is CONFIRM, FSM_CONVERSATION will be in the PEND_DEALL state when the CONFIRMED record arrives. The CONFIRMED record causes the requested deallocation to be completed.

Referenced procedures, FSMs, and data structures:


```
Do while CONFIRMED response has not been received:
   Get the first record for this conversation.
   Select based on the type of the record:
      When CONVERSATION_FAILURE then
         Call CONVERSATION_FAILURE_PROC with record (see page 5.1-29).
         If state of FSM_ERROR_OR_FAILURE (page 5.1-67) is CONV_FAILURE_SON then
            Set RETURN_CODE parameter of the verb to RESOURCE_FAILURE_RETRY.
         Else
            Set RETURN_CODE parameter of the verb to RESOURCE_FAILURE_NO_RETRY.
         Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
         Response condition is satisfied.
```

WAIT_FOR_CONFIRMED_PROC

```
          When REQUEST_TO_SEND (see Note 1)
             Set RCB.RQ_TO_SEND_RCVD to YES.
             Destroy the REQUEST_TO_SEND record.
          When RECEIVE_ERROR (see Note 2)
             Call FSM_ERROR_OR_FAILURE(RECEIVE_ERROR, RCB) (page 5.1-67).
             Call RECEIVE_RM_OR_HS_TO_PS_RECORDS(SUSPEND_LIST contains RCB ID) (page 5.1-38).
             If state of FSM_ERROR_OR_FAILURE (page 5.1-67) is CONV_FAILURE_SON or
              CONV_FAILURE_PROTOCOL_ERROR then
                If state of FSM_ERROR_OR_FAILURE (page 5.1-67) is CONV_FAILURE_SON then
                   Set RETURN_CODE parameter of the verb to RESOURCE_FAILURE_RETRY.
                Else
                   Set RETURN_CODE parameter of the verb to RESOURCE_FAILURE_NO_RETRY.
                Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
             Else
                Call DEQUEUE_FMH7_PROC(CONFIRM verb parameters, RCB) (page 5.1-34).
             Response condition is satisfied.
             Destroy the RECEIVE_ERROR record.
          When CONFIRMED (see Note 3)
             Set RETURN_CODE parameter of the verb to OK.
             If state of FSM_CONVERSATION is PEND_DEALL then (see Note 4)
                Call FSM_CONVERSATION(R, DEALLOCATION_INDICATOR, RCB) (page 5.1-65).
                Call END_CONVERSATION_PROC(RCB) (page 5.1-34).
             Response condition is satisfied.
             Destroy the CONFIRMED record.
          Otherwise
             Call PS_PROTOCOL_ERROR(RCB.HS_ID, X'10010000') (page 5.0-20).
             Call FSM_CONVERSATION(R, CONFIRMED, RCB) (page 5.1-65).
```

WAIT_FOR_RM_REPLY

FUNCTION:	This procedure waits for an expected reply from the resources manager.
INPUT:	At least one record from the RM_TO_PS_Q
OUTPUT:	A record received from the resources manager
NOTES: 1.	CONVERSATION_FAILURE is the only record that can arrive unexpectedly from the resources manager.
2.	Any record from the resources manager, other than CONVERSATION_FAILURE, must be the expected reply. No more than one reply from the resources manager is outstanding at any time.

Referenced procedures, FSMs, and data structures:

RM	page 3-19
CONVERSATION_FAILURE_PROC	page 5.1-29
CONVERSATION_FAILURE	page A-21

```
Do while record from RM has not arrived:
   Wait for a record to arrived from RM.
   If the record from RM is a CONVERSATION_FAILURE then (see Note 1)
      Call CONVERSATION_FAILURE_PROC(CONVERSATION_FAILURE record) (page 5.1-29).
   Else (see Note 2)
      Record received from RM, return the record.
```

WAIT_FOR_RSP_TO_RQ_TO_SEND_PROC

FUNCTION: This procedure is invoked after PS has issued a REQUEST_TO_SEND to HS. The next record that is expected from HS is RSP_TO_REQUEST_TO_SEND.

HS can send records to PS while PS is waiting for the expected RSP_TO_REQUEST_TO_SEND record. Appropriate action is taken, depending upon the record received (see below).

INPUT: The RCB corresponding to the conversation for which the REQUEST_TO_SEND was issued is passed as a parameter to this procedure; HS_TO_PS_RECORDs are received from HS.

OUTPUT: See below.

NOTES:

1. If a REQUEST_TO_SEND is received, PS stores that information in the RCB and continues to wait for the RSP_TO_REQUEST_TO_SEND.
2. Since REQUEST_TO_SEND does not support the RESOURCE_FAILURE return code, if a RECEIVE_ERROR is received, the information is stored in FSM_ERROR_OR_FAILURE to be presented to the transaction program when it issues a verb that does support the RESOURCE_FAILURE return code.
3. When RSP_TO_REQUEST_TO_SEND is received, control is returned to the transaction program.
4. Any data received from HS before the RSP_TO_REQUEST_TO_SEND arrives is stored in the HS_TO_PS_BUFFER_LIST. PS continues to wait for the RSP_TO_REQUEST_TO_SEND. However, if an MU with TYPE field set to DEALLOCATE_FLUSH is received, the RSP_TO_REQUEST_TO_SEND will not be received by PS, so PS returns control to the transaction program.

Referenced procedures, FSMs, and data structures:

CONVERSATION_FAILURE_PROC	page 5.1-29
PS_PROTOCOL_ERROR	page 5.0-20
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
COVERSATION_FAILURE, see CONVERSATION_FAILURE	page A-21
MU	page A-29
REQUEST_TO_SEND	page A-10
RECEIVE_ERROR, see RSP_TO_REQUEST_TO_SEND	page A-11
RSP_TO_REQUEST_TO_SEND, see RECEIVE_ERROR	page A-10
RCB	page A-6

```
Do while the response has not been received (response condition not satisfied):
  Get the first record for this conversation.
  Select based on the type of the record:
     When CONVERSATION_FAILURE then
        Call CONVERSATION_FAILURE_PROC with record (page 5.1-29).
        Response condition is satisfied.
     When REQUEST_TO_SEND (see Note 1)
        Set RCB.RQ_TO_SEND_RCVD to YES.
        Destroy the REQUEST_TO_SEND record.
     When RECEIVE_ERROR (see Note 2)
        Call FSM_ERROR_OR_FAILURE(RECEIVE_ERROR, RCB) (see page 5.1-67)
        Destroy the RECEIVE_ERROR record.
     When RSP_TO_REQUEST_TO_SEND (see Note 3)
        Response condition is satisfied.
        Destroy the RSP_TO_REQUEST_TO_SEND record.
     When MU (see Note 4)
        Add the MU to the RCB.HS_TO_PS_BUFFER_LIST.
        Set MU to the last entry of HS_TO_PS_BUFFER_LIST.
        If the end-of-chain type has been received and is DEALLOCATE_FLUSH then
           Response condition is satisfied.
     Otherwise
      Call PS_PROTOCOL_ERROR(RCB.HS_ID, FMH-7 Sense Data) (page 5.0-20).
      Call FSM_CONVERSATION(S, CONFIRMED, RCB) (page 5.1-65).
```

WAIT_FOR_SEND_ERROR_DONE_PROC

FUNCTION: This procedure is invoked after a SEND_ERROR MU has been sent to HS. The SEND_ERROR was sent to HS as a result of the transaction program issuing a SEND_ERROR or DEALLOCATE (TYPE = ABEND_PROG, ABEND_SVC, or ABEND_TIMER) for a conversation that is in receive state.

The procedure calls GET_END_CHAIN_FROM_HS (page 5.1-36) to await the arrival from HS of a record indicating the end-of-chain type. Appropriate action is taken depending on the type of record received.

INPUT: Transaction program verb parameters and the RCB corresponding to the resource specified in the verb

OUTPUT: See below.

NOTES:

1. If the record received from HS is an MU with the MU.HS_TO_PS.TYPE field set to DEALLOCATE_FLUSH, the conversation is deallocated and the return code of the verb is set to indicate the deallocation.

2. If the record received from HS is an MU with the MU.HS_TO_PS.TYPE field set to DEALLOCATE_CONFIRM, CONFIRM, PREPARE_TO_RCV_CONFIRM, or PREPARE_TO_RCV_FLUSH, the processing of the verb is continued.

3. FSM_ERROR_OR_FAILURE is reset to NO_REQUESTS because, in certain SEND_ERROR race cases, a RCVD_ERROR condition is not reported to the transaction program. Normally, FSM_ERROR_OR_FAILURE is reset to NO_REQUESTS by SET_FMH7_RC (page 5.1-59) when the error is reported to the TP.

Referenced procedures, FSMs, and data structures:

GET_END_CHAIN_FROM_HS	page 5.1-36
SEND_ERROR_DONE_PROC	page 5.1-55
COMPLETE_DEALLOCATE_ABEND_PROC	page 5.1-29
FSM_CONVERSATION	page 5.1-65
FSM_ERROR_OR_FAILURE	page 5.1-67
RCB	page A-6

```
Call GET_END_CHAIN_FROM_HS(RCB) (page 5.1-36).

Select based on the state of FSM_ERROR_OR_FAILURE (page 5.1-67):
   When CONV_FAILURE_SON
      Set the RETURN_CODE of the verb to RESOURCE_FAILURE_RETRY.
      Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
   When CONV_FAILURE_PROTOCOL_ERROR
      Set the RETURN_CODE of the verb to RESOURCE_FAILURE_NO_RETRY.
      Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-65).
   Otherwise
      Select based on the following conditions:
         When end-of-chain type is DEALLOCATE_FLUSH (see Note 1) and the verb is SEND_ERROR
            Set the RETURN_CODE of the verb to DEALLOCATE_NORMAL.
            Call FSM_CONVERSATION(R, DEALLOCATE_NORMAL_RC, RCB) (page 5.1-65).
         When end-of-chain type is DEALLOCATE_FLUSH and the verb is DEALLOCATE
            Set the RETURN_CODE of the verb to OK.
         When end-of-chain type is DEALLOCATE_CONFIRM, CONFIRM, PREPARE_TO_RCV_CONFIRM,
          or PREPARE_TO_RCV_FLUSH (see Note 2) and the verb is SEND_ERROR
            Purge the end-of-chain type received on this conversation.
            Call SEND_ERROR_DONE_PROC(SEND_ERROR verb parameters, RCB)
             (page 5.1-55).
            Call FSM_CONVERSATION(S, SEND_ERROR, RCB) (page 5.1-65).
         When end-of-chain type is DEALLOCATE_CONFIRM, CONFIRM, PREPARE_TO_RCV_CONFIRM,
          or PREPARE_TO_RCV_FLUSH, and the verb is DEALLOCATE
            Call COMPLETE_DEALLOCATE_ABEND_PROC(DEALLOCATE verb parameters, RCB)
             (page 5.1-29).
Call FSM_ERROR_OR_FAILURE (page 5.1-67) and pass it a RESET signal (see Note 3).
```

FSM_CONVERSATION

FUNCTION:	This finite-state machine maintains the status of a conversation resource. The states have the following meanings: • RESET = conversation initial state, the program can allocate it • SEND_STATE = the program can send data, request confirmation, or request sync point • RCV_STATE = receive, the program can receive information from the remote program • RCVD_CONFIRM = received confirm, PS received the confirm indicator from the HS • RCVD_CONFIRM_SEND = received confirm send, PS received the confirm send indicator from HS • RCVD_CONFIRM_DEALL = received confirm deallocate, PS received the confirm deallocate from HS • PREP_TO_RCV_DEFER = prepare to receive defer, the program issued a PREPARE_TO_RECEIVE verb with SYNCPT • DEALL_DEFER = deallocate defer, the program issued DEALLOCATE verb with SYNCPT • PEND_DEALL = pending deallocate, the program issued DEALLOCATE verb with CONFIRM
INPUT:	The inputs are marked with S if they result from an action of the local transaction program and with R if they result from a record sent to PS by HS. The RCB is passed to provide the information needed to perform the state transition of the FSM_CONVERSATION and its output function.
NOTE:	PEND_DEALL is an intermediate state. PS does not return control to the transaction program when the conversation is in this state.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
FSM_ERROR_OR_FAILURE	page 5.1-67
MU	page A-29
RCB	page A-6

INPUTS / STATE NAMES----> / STATE NUMBERS-->	RESET 01	SEND STATE 02	RCV STATE 03	RCVD CONFIRM 04	RCVD CONFIRM SEND 05	RCVD CONFIRM DEALL 06	PREP TO RCV DEFER 07	DEALL DEFER 08	PEND DEALL 09	END CONV 010
S, ALLOCATE	2	/	/	/	/	/	/	/	/	/
R, ATTACH	3	/	/	/	/	/	/	/	/	/
S, SEND_DATA	/	-	>	>	>	>	>	>	/	>
S, PREP_TO_RCV_FLUSH	/	3	>	>	>	>	>	>	/	>
S, PREP_TO_RCV_CONFIRM	/	3	>	>	>	>	>	>	/	>
S, PREP_TO_RCV_DEFER	/	7	>	>	>	>	>	>	/	>
S, FLUSH	/	-	>	>	>	>	3	1	/	>
S, CONFIRM	/	-	>	>	>	>	3	9	/	>
S, SEND_ERROR	/	-	2	2	2	2	>	>	/	>
S, RECEIVE_AND_WAIT	/	3(A)	-	>	>	>	>	>	/	>
S, POST_ON_RECEIPT	/	>	-	>	>	>	>	>	/	>
S, WAIT	/	>	-	>	>	>	>	>	/	>
S, TEST_POSTED	/	>	-	>	>	>	>	>	/	>
S, TEST_RQ_TO_SEND_RCVD	/	-	-	>	>	>	-	-	/	>
S, RECEIVE_IMMEDIATE	/	>	-	>	>	>	>	>	/	>
S, REQUEST_TO_SEND	/	>	-	-	>	>	>	>	/	>
R, SEND_INDICATOR	/	/	2	/	/	/	/	/	/	/
R, CONFIRM_INDICATOR	/	/	4	/	/	/	/	/	/	/
R, CONFIRM_SEND_IND	/	/	5	/	/	/	/	/	/	/
R, CONFIRM_DEALLOC_IND	/	/	6	/	/	/	/	/	/	/
S, CONFIRMED	/	>	>	3	2	10	>	>	/	>
R, PROGRAM_ERROR_RC	/	3(B)	-(B)	/	/	/	3(B)	3(B)	3(B)	/
R, SERVICE_ERROR_RC	/	3(B)	-(B)	/	/	/	3(B)	3(B)	3(B)	/
R, DEALLOC_NORMAL_RC	/	10	10	/	/	/	/	/	/	/
R, DEALLOC_ABEND_RC	/	10(B)	10(B)	/	/	/	10(B)	10(B)	10(B)	/
R, RESOURCE_FAILURE_RC	/	10(B)	10(B)	/	/	/	10(B)	10(B)	10(B)	/
R, ALLOCATION_ERROR_RC	/	10(B)	10(B)	/	/	/	10(B)	10(B)	10(B)	/
S, DEALLOCATE_FLUSH	/	1	>	>	>	>	>	>	/	>
S, DEALLOCATE_CONFIRM	/	9	>	>	>	>	>	>	/	>
S, DEALLOCATE_DEFER	/	8	>	>	>	>	>	>	/	>
S, DEALLOCATE_ABEND	/	1	1	1	1	1	1	1	/	>
S, DEALLOCATE_LOCAL	/	>	>	>	>	>	>	>	/	1
R, DEALLOCATED_IND	/	/	/	/	/	/	/	/	1	/
S, GET_ATTRIBUTES	/	-	-	-	-	-	-	-	/	-

OUTPUT CODE	FUNCTION
A	Set MU.PS_TO_HS.TYPE to PREPARE_TO_RCV_FLUSH and send the MU record to HS.
B	Call FSM_ERROR_OR_FAILURE (page 5.1-67) and pass it a RESET signal.

FSM_ERROR_OR_FAILURE

FUNCTION: This finite-state machine remembers if any error or failure MU records (either from HS to PS or RM to PS) have been received by PS. This knowledge is maintained until the information reflected by the records can be passed to the transaction program. The meanings of the states are as follows:

- NO_REQUESTS = the initial state of the FSM
- RCVD_ERROR = a RECEIVE_ERROR was received
- CONV_FAILURE_PROTOCOL_ERROR = a conversation protocol error record was received
- CONV_FAILURE_SON = a session outage notification for the conversation was received

NOTE: The inputs are the error and failure records from the HS and RM.

Referenced procedures, FSMs, and data structures:

RCB page A-6
MU page A-29

STATE NAMES----> / INPUTS STATE NUMBERS-->	NO REQUESTS 01	RCVD ERROR 02	CONV FAILURE PROTOCOL ERROR 03	CONV FAILURE SON 04
SIGNAL(CONV_FAIL_PROTOCOL)	3	3	/	/
SIGNAL(CONV_FAIL_SON)	4	4	/	/
RECEIVE_ERROR	2(A)	/	-	-
SIGNAL(RESET)	-	1	-	-

OUTPUT CODE	FUNCTION
A	If the send MU buffer exists then Set the values in the send MU to their initial values (purge the data in the MU)

FSM_POST

FUNCTION: This finite-state machine maintains the posting status of a conversation. The meanings of the states are as follows:

- RESET = the initial state of the FSM
- PEND_POSTED = state after the FSM received a POST_ON_RECEIPT input
- POSTED = state to show that post conditions were satisfied

NOTES:

1. If POST_ON_RECEIPT is issued after posting has already been activated (i.e., a prior POST_ON_RECEIPT has been issued), the post conditions used to test for post satisfied are reinitialized to those carried in the most recent POST_ON_RECEIPT.
2. RECEIVE_IMMEDIATE resets posting. If posting is activated and the conversation has been posted, this FSM is reset. If posting is activated and the conversation has not been posted, posting is canceled and this FSM is reset.
3. The initial state of this FSM is RESET.

STATE NAMES----> / INPUTS STATE NUMBERS-->	RESET 01	PEND POSTED 02	POSTED 03
POST_ON_RECEIPT	2	- [Note 1]	2 [Note 1]
TEST	-	-	1
WAIT	-	-	1
RECEIVE_IMMEDIATE	-	1 [Note 2]	1 [Note 2]
SIGNAL(POST)	/	3	-

CHAPTER 5.2. PRESENTATION SERVICES--MAPPED CONVERSATION VERBS

GENERAL DESCRIPTION

A transaction program (TP) requests LU services by issuing verbs. The verbs request several different kinds of services, and are therefore divided into several different categories (see SNA Transaction Programmer's Reference Manual for LU Type 6.2 for a complete description of the verbs). Each verb-processing component of PS processes the verbs of one specific category. Presentation services for mapped conversations (PS.MC) is the PS component that processes the verbs of the mapped conversation category (basic conversation verbs are processed by "Chapter 5.1. Presentation Services--Conversation Verbs" in Chapter 5.1).

The mapped conversation verbs are issued on mapped conversations, and basic conversation verbs are issued on basic conversations. Both the basic and the mapped conversation verbs request communication services for transaction programs. A mapped conversation, however, is easier for the communicating transaction programs to use because it also provides data formatting services that the programs would have to perform for themselves if they were using a basic conversation.

PS.MC FUNCTIONS

The primary function of PS.MC is reformatting the data contained in the DATA parameters of the MC_SEND_DATA and MC_RECEIVE_AND_WAIT verbs. Its subsidiary functions include the processing of errors related to this reformatting, and the translation of mapped conversation verbs into basic conversation verbs in support of services unrelated to formatting.

When the TP issues a mapped conversation verb, PS.MC processes the verb and performs the services that it requests. PS.MC does not, however, perform all the services requested by every mapped conversation verb. PS.MC performs only those services related to data formatting. If the verb requests additional conversation services that are not related to data formatting, then PS.MC, by issuing one or more basic conversation verbs, causes PS.CONV to perform those services.

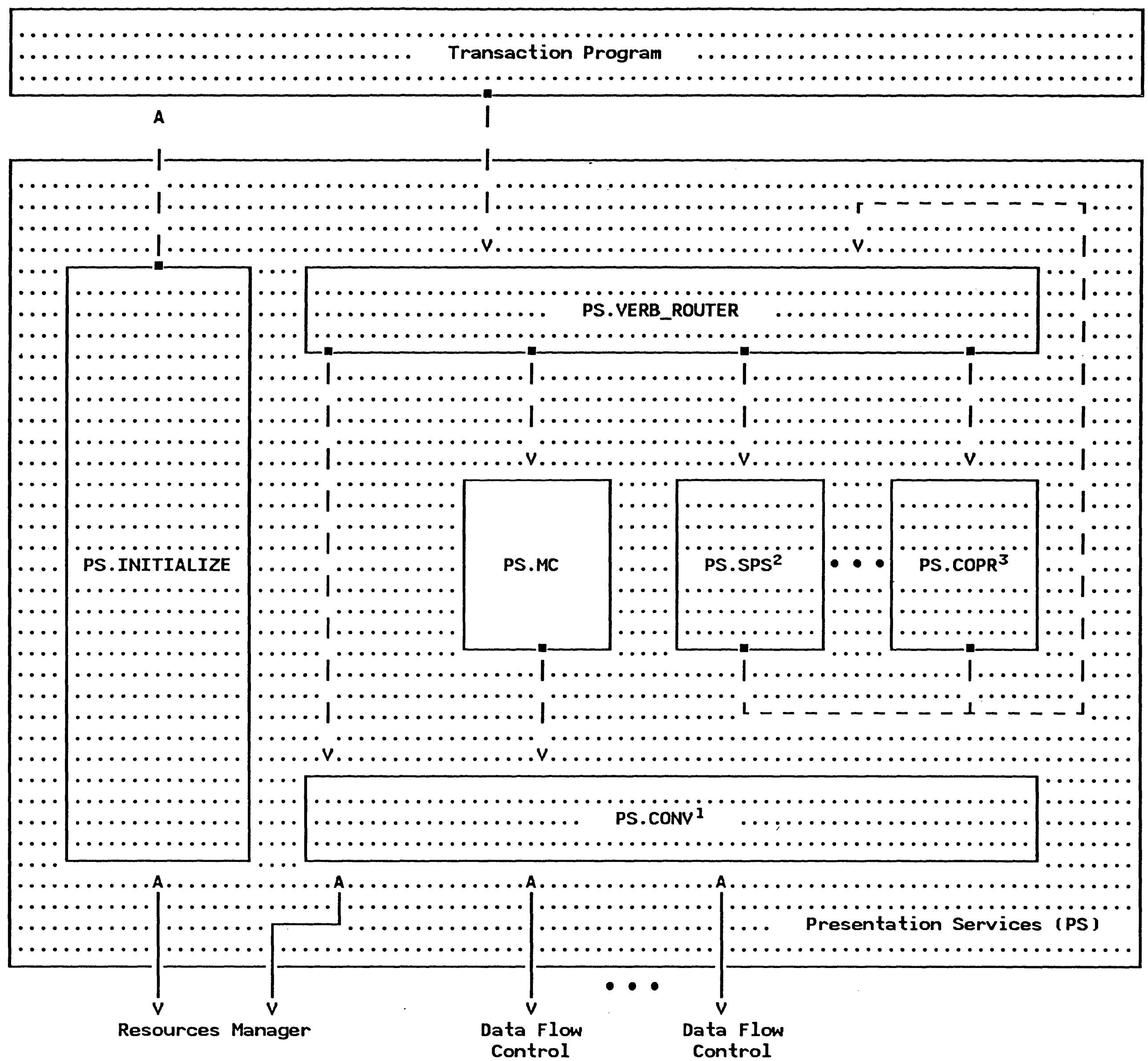
In general, the TP is faced with two formatting problems. The data format that it prefers for computational processing differs from the formats in which data is presented to (or by):

- Local end users and resources
- Half-sessions (for communication with remote end users and resources).

PS.MC solves the formatting problem for local end users and resources by routing all data presented to (or by) them through a component called "the Mapper" (UPM_MAPPER on page 5.2-46), which transforms data into (when receiving) or out of (when sending) formats preferred by end users. For communication with conversation partners, TP data must be made to conform to the format that SNA defines for the conversation data stream. On basic conversations, the conversing TPs must perform this formatting for themselves, but on mapped conversations, PS.MC adds (when sending) and strips (when receiving) the data-stream details required by the format.

The functions that PS.MC performs for the transaction program are summarized below:

- Adding and stripping conversation data-stream formatting details (see "Conversation Data Stream Formatting" on page 5.2-5)
- Data mapping (see "Data Mapping and the Mapper" on page 5.2-8)
- Allowing function management headers (FMHs) to flow on the mapped conversation (see "FM Header Data" on page 5.2-7)
- Detecting service errors committed by the partner transaction program (see "Service Errors Detected in Received Data" on page 5.2-14)
- Processing service errors committed by the local transaction program and detected by the partner LU (see "Processing of a Service Error Detected by Partner LU" on page 5.2-17).



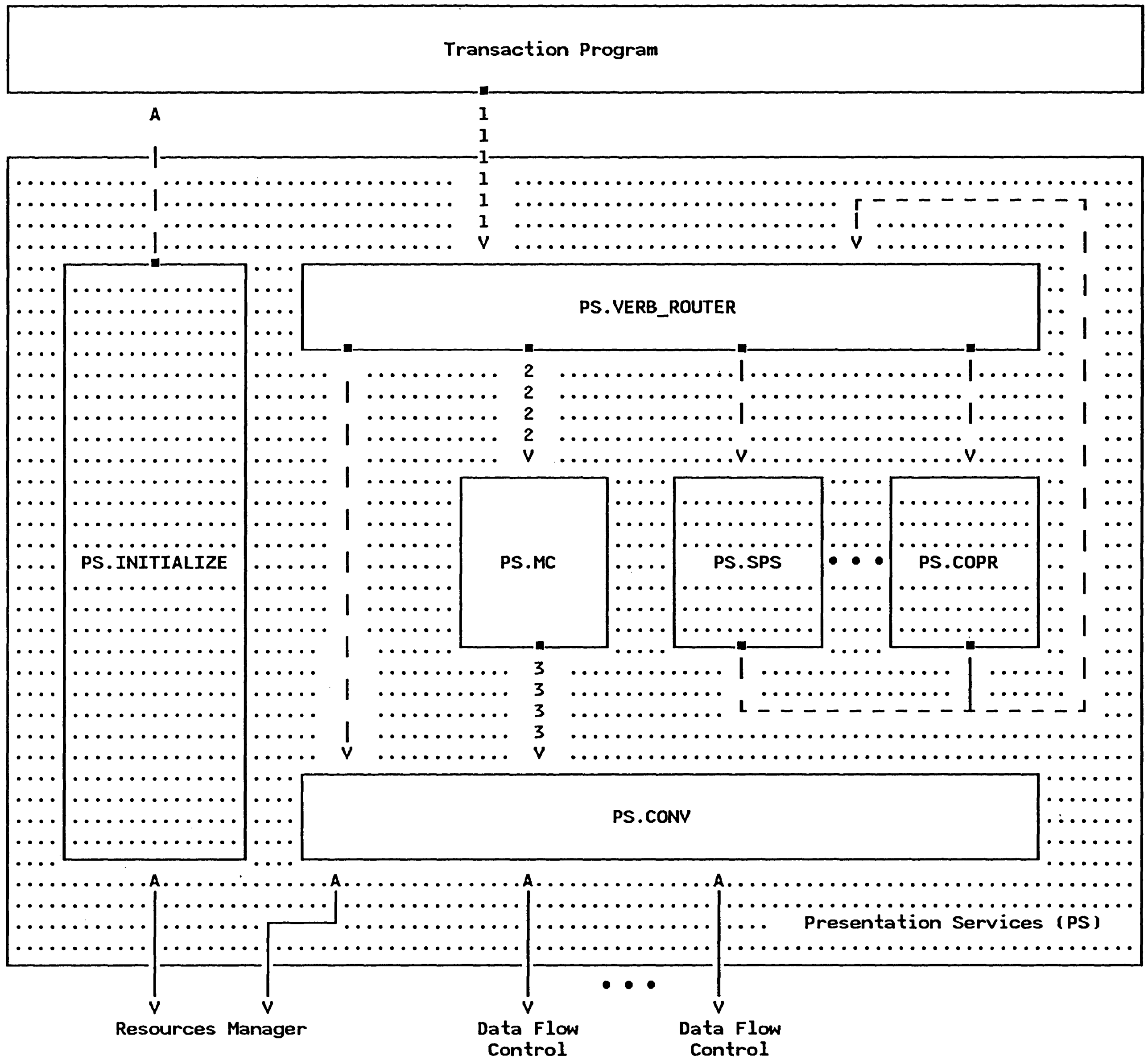
[1] See "Chapter 5.1. Presentation Services--Conversation Verbs"
[2] See "Chapter 5.3. Presentation Services--Sync Point Services Verbs"
[3] See "Chapter 5.4. Presentation Services--Control-Operator Verbs"

Note: A dashed line denotes a synchronous (or call/return) protocol boundary between PS components, while a solid line denotes an asynchronous (or send/receive) protocol boundary.

Figure 5.2-1. Overview of Presentation Services, Emphasizing Presentation Services for Mapped Conversations

COMPONENT INTERACTIONS

In terms of layering, non-basic-conversation verb-processing components (such as PS.MC) reside below the TP but above the PS.CONV sublayer of presentation services. PS.MC communicates primarily with the TP and PS.CONV. Figure 5.2-2 on page 5.2-3 illus-



Note: See "Component Interactions" on page 5.2-2 for an explanation of the flows shown in this figure.

Figure 5.2-2. PS.MC's Use of the Basic Conversation Protocol Boundary

trates the flow of processing. PS.MC accepts issuances of mapped conversation verbs from the TP, but issues basic conversation verbs to PS.CONV. Whenever a verb is issued by any component (for example, the TP), PS.VERB_ROUTER (Chapter 5.0) gains control and is responsible for routing the verb to the appropriate PS component for processing.

When the TP issues a mapped conversation verb (flow 1 in Figure 5.2-2), the verb is inspected by PS.VERB_ROUTER. PS.VERB_ROUTER determines that the verb is a mapped conversation verb and calls PS.MC, passing to it the received verb (flow 2). PS.MC may issue a basic conversation verb during its processing of the mapped conversation verb. If it does, then it calls PS.CONV and passes the verb to it (flow 3). PS.CONV processes the basic conversation verb, after which control returns along the same path to PS.MC.

A transaction program may support only mapped conversations or only basic conversations. Alternatively, it may support both types of conversation. In the latter case, the transaction program may have mapped conversations and basic conversations allocated concurrently. The PS.VERB_ROUTER requires the TP to issue only mapped conversation verbs on mapped conversations and only basic conversation verbs on basic conversations. However, PS.VERB_ROUTER allows PS.MC to issue basic conversation verbs on a mapped conversation.

PS.MC DATA BASE STRUCTURE

In order to perform its functions, PS requires information about the transaction program that it is serving and about the resources currently allocated to that transaction program. This information, which is described in "PS.CONV Data Base Structure" on page 5.1-1 , is stored in lists of control blocks in the LU (see Appendix A for complete definitions of the lists and of the entities that may be found in the lists). Some of the fields in these control blocks are especially important to PS.MC. Those fields are described below.

TRANSACTION CONTROL BLOCK (TCB)

Each transaction control block (TCB) contains information about one execution instance of a transaction program. PS.MC identifies the TCB describing the particular transaction program instance that it is serving by means of the TCB_ID that RM passed to PS when the transaction program instance was created. The TCB fields used by PS.MC contain such information as the name of the transaction program that PS is serving and the LU_ID of the LU in which the PS resides.

LU CONTROL BLOCK (LUCB)

PS.MC accesses the appropriate LUCB using a unique LU_ID, which is stored in the TCB to which PS.MC has access. The LUCB fields in which PS.MC is particularly interested contain information about whether the LU supports various mapped conversation options, such as handling of FM header data.

Transaction Program Control Block (TPCB)

Each LUCB also contains a pointer to a list of transaction program control blocks (TPCBs). For a given LU, the list contains a TPCB for each transaction program that is capable of running at the LU. The information contained in a TPCB includes the name of the transaction program and whether it supports various optional features. PS.MC, in particular, is interested in whether or not the TP supports mapped conversations.

RESOURCE CONTROL BLOCK (RCB)

PS.MC also requires information about all the mapped conversations allocated to the transaction program. This information is found in the resource control block (RCB), one for each resource associated with any transaction programs running at an LU. As in the case of the TCB, PS.MC is interested in only those RCBs containing information about mapped conversation resources allocated to its own transaction program. It does not need information about resources that are not mapped conversations or are allocated to other transaction programs.

PS.MC accesses an RCB by means of the RCB_ID. The transaction program supplies an RCB_ID in the RESOURCE parameter of a verb in order to indicate the particular conversation resource on which the verb is being issued. Whenever a new resource is allocated, the resources manager ("Chapter 3. LU Resources Manager" in Chapter 3) creates a new RCB_ID and returns it to the transaction program in the RESOURCE parameter of the MC_ALLOCATE verb. The RCB also contains the TCB_ID of the transaction program instance that has allocated the resource. RCB information is initialized when the conversation is allocated.

The following RCB fields are especially important to PS.MC.

MC_MAX_SEND_SIZE contains the length of the longest logical record that can be sent on the conversation, and is used to segment outgoing data (see "Construction of GDS Variables" on page 5.2-5).

MAPPER_SAVE_AREA contains information used in data mapping, such as the currently effective map names (see "Map Names" on page 5.2-8). The mapper may also, however, use data stored in this area to perform implementation-defined (as opposed to SNA-map-name-defined) data mapping. The mapper also uses this area to save any error data or indicators of events that occurred during data mapping.

MC_RECEIVE_BUFFER contains information that has arrived from a conversation partner but that has not yet been received by the transaction program.

CONVERSATION DATA STREAM FORMATTING

When a transaction program sends data on a basic conversation, it must ensure that that data conforms to the format of the conversation data stream. A transaction program that allocates a mapped conversation, however, does not need to perform this task, because PS.MC assumes responsibility for editing the data to make it conform to the format of the conversation data stream. Transaction programs communicating over a mapped conversation may supply their data in any format.

All data flowing on any conversation is formatted into logical records. A logical record consists of a 2-byte logical record length field (LL) followed by a data field. A transaction program sending data over a basic conversation must take care to include the LL fields in its data, and to complete the logical record that it is sending before leaving SEND state.

A TP sending data over a mapped conversation has neither of these concerns, because PS.MC computes and inserts the LLs for it. The TP simply supplies the data in the DATA parameter of MC_SEND_DATA. PS.MC then maps the data and formats the mapped data into one or more complete logical records.

CONSTRUCTION OF GDS VARIABLES

PS.MC formats all data flowing on a mapped conversation into general data stream (GDS) variables (see Figure 5.2-3). A GDS variable consists of one or more complete logical records. The data field of the first logical record in a GDS variable begins with a 2-byte GDS ID that identifies the type of information contained in the variable. The information itself begins in the third byte of the data field of the first logical record, and continues throughout the data fields of the variable's remaining logical records, which do not contain the GDS ID.

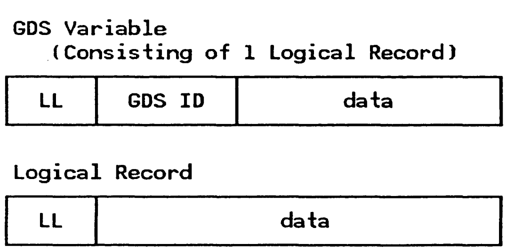


Figure 5.2-3. GDS Variables and Logical Records

The following types of GDS variables flow on mapped conversations:

- Map Name
- Application Data
- User Control Data
- Error Data
- Error Log
- Null Data

(See SNA Formats for descriptions of all the valid types of GDS variables and their GDS ID values.)

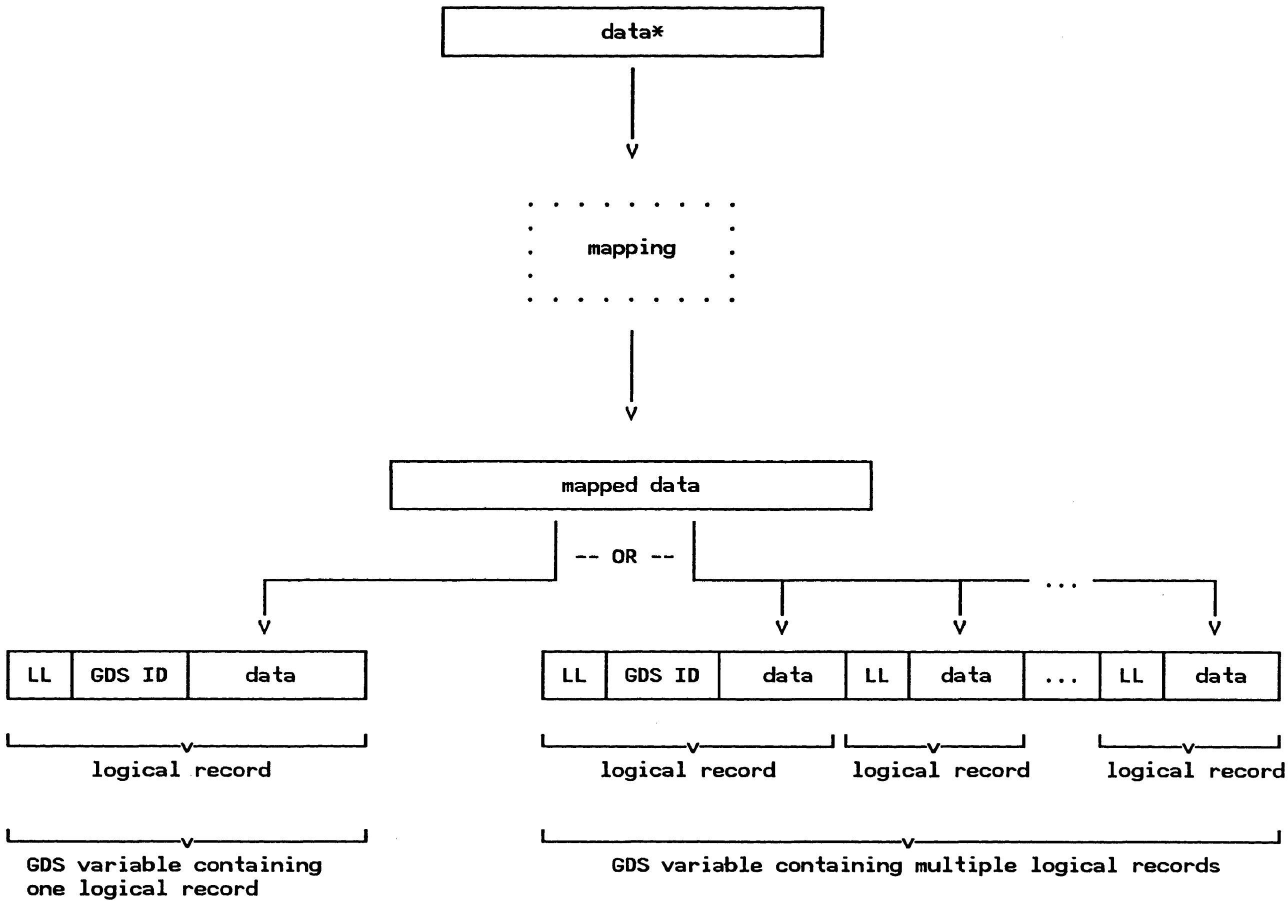
Map Name GDS variables are generated from the MAP_NAME parameter of MC_SEND_DATA (see "Map Names" on page 5.2-8 for details). Application Data and User Control Data GDS variables (collectively called data GDS variables) are generated from data supplied via the DATA parameter of MC_SEND_DATA. If this verb is issued with FMH_DATA(YES), the data is put into a User Control Data GDS variable; otherwise, it is put into an Application Data GDS variable. Error Data GDS variables are generated when the TP issues MC_SEND_ERROR or when PS detects an error (see "Mapped Conversation Errors" on page 5.2-12 for details).

Null Data GDS variables are optionally generated when the TP, after entering SEND state, leaves SEND state without sending any data. (Instead of issuing MC_SEND_DATA, the TP issues MC_CONFIRM, MC_PREPARE_TO_RECEIVE, or some other verb that removes the TP from SEND state.) The partner must be notified of this change in state. The RH that conveys this state-change notification (CD for MC_PREPARE_TO_RECEIVE, or RQD2 for MC_CONFIRM) can flow to the partner by means of a null-data FMD request, a LUSTAT request, or a Null Data GDS variable created by PS.MC. An Application Data GDS variable with a null data field cannot be used for this purpose, because that would erroneously indicate that the TP had issued MC_SEND_DATA with LENGTH(0), when the TP has not issued MC_SEND_DATA at all.

GDS Variables with Multiple Logical Records

Only data GDS variables may consist of multiple logical records; Error and Map Name GDS variables each consist of a single logical record. Whether a data GDS variable will have more than one logical record is determined by the value of MC_MAX_SEND_SIZE, which is the length of the longest logical record that may be sent on the mapped conversation. MC_MAX_SEND_SIZE may vary from mapped conversation to mapped conversation, or it may be the same for all mapped conversations. MC_MAX_SEND_SIZE is stored in the resource control block associated with the conversation (see "PS.CONV Data Base Structure" on page 5.1-1 and "PS.MC Data Base Structure" on page 5.2-4 for further details).

If the length of the data returned from the mapper does not exceed MC_MAX_SEND_SIZE, PS.MC creates a GDS variable containing a



* This data is supplied by the transaction program in an MC_SEND_DATA verb.

Note: The DATA field of the first, or only, logical record in a GDS variable begins with a 2-byte GDS ID. Subsequent logical records in the same GDS variable do not carry a GDS ID value.

Figure 5.2-4. Transformation of Data from MC_SEND_DATA to a GDS Variable

single logical record. MC_MAX_SEND_SIZE is used only to determine how many logical records to create from the data; it is not used to determine whether enough data exists to be sent to the partner LU. (See Figure 5.2-4.)

If PS.MC determines that multiple logical records are required, the LL fields of all but the last logical record have the high-order bit turned on to indicate that the data is continued in the next logical record. PS.MC continues to create logical records containing data returned from the mapper until the end of the data is reached. The high-order bit of the LL field of the last logical record in the outgoing GDS variable is turned off by the mapper. Figure 5.2-4 illustrates the transfer of outgoing data from its beginning in the DATA parameter of MC_SEND_DATA to its final position in a logical record in a GDS variable.

When the TP is receiving data, this process is reversed. PS.MC continues to receive data from PS.CONV until it receives a logical record in which the high-order bit of the LL field is off. At this point, PS.MC has a complete data GDS variable. Next, PS.MC strips the GDS ID and LLs from the received data, and maps the data according to the currently effective map name. The mapped data goes into application transaction program variables.

In either case, exactly one data GDS variable is created as a result of each issuance of MC_SEND_DATA, and exactly one GDS variable is received as a result of each issuance of MC_RECEIVE_AND_WAIT.

FM HEADER DATA

In LU 6.2, FM header data is normally processed by PS rather than by the transaction program. All FMHs except FMH-5 (to initiate a conversation) and FMH-7 (to report a PS or transaction program error) are trapped as errors. In LU 6.1, however, an FMH-5 could be used for transaction program functions (for example, transaction program parameters were sometimes encoded in an FMH-5), and could flow at any time during a conversation. Therefore, in order to allow transaction programs that were written for LU 6.1 to run on LU 6.2, PS.MC provides a way for transaction programs to prevent PS from intercepting the FM header data that they are trying to exchange.

If the TP wants to send application data containing FM headers to its partner, the TP issues MC_SEND_DATA with FMH_DATA(YES). This causes PS.MC to create a User Control Data GDS variable to contain the data. When FM header data is contained in a User Control Data GDS variable, the sending PS and the receiving PS do not process it; they allow it to flow directly to the receiving TP. PS.MC notifies the receiving TP of the presence of FM headers in the received data on the MC_RECEIVE_AND_WAIT verb (see <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>) that the receiving TP issues to receive the data.

Currently, the sole use of User Control Data GDS variables on mapped conversations is this processing of FM header data.

<u>EXAMPLES OF MAPPED CONVERSATION VERB PROCESSING</u>

As discussed in "PS.MC Functions" on page 5.2-1, one function of PS.MC is to translate mapped conversation verbs and their parameters into basic conversation verbs and parameters (the other functions relate specifically to the mapping of data). The functions of PS.MC that relate to verb translation are illustrated and described below. (The data-mapping-related functions are described in detail in "Data Mapping and the Mapper" on page 5.2-8.)

ESTABLISHING A MAPPED CONVERSATION

A mapped conversation is established when the transaction program issues MC_ALLOCATE. PS.MC, upon receipt of MC_ALLOCATE from the transaction program, performs some initial processing. If the processing succeeds, then PS.MC issues the basic conversation verb ALLOCATE, with TYPE(MAPPED_CONVERSATION), to PS.CONV. PS.CONV copies the supplied TYPE value into the Resource Type field in the FMH-5 that it creates as a result of the ALLOCATE. Then, after completing its normal ALLOCATE processing, returns control to PS.MC.

When the FMH-5 arrives at the target LU, it causes the conversation partner transaction program to be <u>attached</u> (or invoked). When the partner program is attached, it is only for the mapped conversation with the transaction program that has just invoked it. It may, however, request additional mapped conversations by issuing MC_ALLOCATE verbs of its own.

Once PS.MC returns control to the transaction program after processing of the MC_ALLOCATE is complete, the transaction program may issue mapped conversation verbs on the conversation whose ID was returned in the RESOURCE_ID parameter of the MC_ALLOCATE.

PS.VERB_ROUTER prohibits the transaction program from issuing basic conversation verbs specifying the resource ID of this mapped conversation. When the transaction program issues a mapped conversation verb, however, PS.VERB_ROUTER allows PS.MC, as part of its processing of that verb, to issue a basic conversation verb on the same mapped conversation. See Chapter 5.0 for a further discussion of this topic.

TERMINATING A MAPPED CONVERSATION

When the transaction program determines that its processing related to a mapped conversation has completed, or that the mapped conversation should be ended for other reasons, it causes the conversation to be terminated by issuing MC_DEALLOCATE. PS.MC processes this by issuing DEALLOCATE to PS.CONV. However, if the MC_DEALLOCATE specified a deallocation type of ABEND (see <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>), PS.MC first translates the ABEND value to ABEND_PROG before setting the type of deallocation. This reflects the fact that it is the transaction program, rather than PS.MC, that caused the DEALLOCATE to be issued. For all other types of deallocation, PS.MC sets the TYPE field of the DEALLOCATE to the value specified in that field of the MC_DEALLOCATE.

The transaction program sends data to its partner by issuing MC_SEND_DATA. The partner transaction program receives this data by issuing MC_RECEIVE_AND_WAIT. Whenever PS.MC processes either of these verbs, it passes the data through a component called the mapper (page 5.2-46). All mapped conversation data is thus mapped twice: once when sent, and once when received. PS.MC's processing of MC_SEND_DATA is called send mapping; its processing of MC_RECEIVE_AND_WAIT is called receive mapping. The particular mappings that the mappers perform are determined by the map name supplied by the sending transaction program.

BLOCK MAPPING

PS.MC performs block mapping, where a block is the amount of data contained in one data GDS variable (see "Construction of GDS Variables" on page 5.2-5 for definitions and descriptions of GDS variables). Typically, a data GDS variable (or block) resides in a transaction program buffer variable dedicated to network communication. The ultimate source or destination of the data, however, is usually one or more other transaction program variables that are significant to the transaction program application. A map provides an algorithm for transferring data between transaction program application variables and the transaction program buffer variable, and for performing any changes in the format or representation of the data that this transfer may require. Thus, in receive mapping, the received data is mapped out of a block and into application variables, while in send mapping, the data is mapped out of application variables and into a block before being sent to the conversation partner.

MAPPING EXAMPLE

Figure 5.2-5 on page 5.2-9 shows a high-level overview of the transformations that map name and data undergo during mapping by the sending and receiving transaction programs. Mapping is symmetric, in that receive mapping is basically the inverse of send mapping.

The transaction program sending data on a mapped conversation supplies a map name with each issuance of MC_SEND_DATA. The map name supplied by the sending transaction program determines the kind of mapping that occurs. In the figure, transaction program A issues MC_SEND_DATA, supplying MAP_NAME(map-name-1) and DATA(data-1). PS.MC, as part of its processing of this verb, then invokes the mapper. PS.MC passes to the mapper map-name-1 and data-1.

The output from the mapper is map-name-2 and data-2. Data-2 may be a different size from data-1 and may be in an entirely different format. After reformatting data-2 into a GDS variable (by breaking it into logical records according to MC_MAX_SEND_SIZE, and prefixing a GDS ID), PS.MC sends map-name-2 and data-2 to the partner LU.

When the data arrives, the PS.MC component at the partner LU processes the MC_RECEIVE_AND_WAIT by repeatedly issuing RECEIVE_AND_WAIT with a fill value of LL. PS.MC accumulates the data, one logical record at a time, until it receives a logical record whose LL field indicates that it is the final logical record of the incoming data GDS variable. At this point, PS.MC has one complete data GDS variable. It then strips the GDS ID and LLs, and invokes the mapper, passing it map-name-2 and data-2. Here, at the receiving LU, the map name and data once again go through a transformation. The receiving mapper transforms map-name-2 and data-2 into map-name-3 and data-3, and returns these to the receiving transaction program in the MAP_NAME and DATA parameters of MC_RECEIVE_AND_WAIT (only the amount of data requested by the transaction program is passed to it; any remaining data that is not requested and returned is discarded). Data-3 may again differ in size and format from data-2, or from data-1. Map-name-3, similarly, may be different from map-name-2 and map-name-1. In the simplest case, the three map names are identical.

"Send Mapping" on page 5.2-10 "Receive Mapping" on page 5.2-11 show more details of the processing of MC_SEND_DATA and MC_RECEIVE_AND_WAIT.

MAP NAMES

With every issuance of MC_SEND_DATA, the transaction program supplies a map name to PS.MC and the mapper. Similarly, on every issuance of MC_RECEIVE_AND_WAIT, the mapper returns a map name to the transaction program. The sending transaction program may supply the same map name repeatedly, and the same map name may be received repeatedly by the receiving transaction program, but the sending PS.MC does not send consecutive duplicate map names. Instead, the locally known map name supplied by the transaction program is translated into a globally known map name and stored in the MAPPER_SAVE_AREA as the currently effective map name. This map name is similarly stored by the receiving PS.MC. The sending PS.MC sends (and the receiving PS.MC receives) a new map name only when the currently effective map name changes. The currently effective map name changes when the map name supplied by the sending transaction program is translated into a globally known map name that differs from the currently effective one stored in the MAPPER_SAVE_AREA. When the mapper discovers this difference, it updates the cur-

```
 ┌──────────────────────────────────────────┐   ┌──────────────────────────────────────────┐
 │                          (Sending LU)    │   │  (Receiving LU)                          │
 │   TP(A)                     PS.MC        │   │      PS.MC                       TP(B)   │
 │                           (MAPPER)       │   │    (MAPPER)                              │
 └──────────────────────────────────────────┘   └──────────────────────────────────────────┘

       .                         .                      .                        .
       .   map-name-1, data-1    .                      .                        .
       o------------------------>o                      .                        .
       .                         .                      .                        .
       .                         .   map-name-2, data-2 .                        .
       .                         o--------------------->o                        .
       .                         .                      .                        .
       .                         .                      .  map-name-3, data-3    .
       .                         .                      o----------------------->o
       .                         .                      .                        .
```

Map-name-1 and data-1 are supplied by the sending transaction program on MC_SEND_DATA. Map-name-2 and data-2 flow from sending PS.MC to receiving PS.MC as GDS variables. Map-name-3 and data-3 are passed to the receiving transaction program via MC_RECEIVE_AND_WAIT .

See "Mapping Example" for an explanation of the flows shown in this figure.

Figure 5.2-5. An Example of Mapping

rently effective map name in its MAPPER_SAVE_AREA, and informs PS.MC of this change by returning an indicator and the new map name.

The mapper can translate map names in many different ways. It may, for example, translate the supplied map name to null, thereby preventing the data from being transformed. The mapper may also translate two different locally known map names to the same globally known map name. For instance, if the transaction program issues MC_SEND_DATA with map name A followed by another MC_SEND_DATA with map name B, the mapper may map both map names to map name C. Moreover, the mapper may translate the same locally known map name differently on different occasions. If the transaction program issues MC_SEND_DATA with map name A and the mapper translates it to map name B, then when the transaction program again issues MC_SEND_DATA with map name A, the mapper may, because of information known only to itself, translate this map name to map name C. Nevertheless, the translation of map names by the mapper is subject to some constraints. For example, the mapper never translates a null map name to a nonnull map name.

Map Name GDS Variables

To complete its processing of a change in the effective map name, the sending PS.MC must notify the receiving PS.MC of the change. It does this by sending to the receiving PS.MC a Map Name GDS variable containing the new effective map name. In this situation, a single MC_SEND_DATA causes two GDS variables to be created: a Map Name GDS variable and a data GDS variable.

Similarly, the receiving mapper saves, in its MAPPER_SAVE_AREA, the map name received in a Map Name GDS variable. When subsequent data GDS variables are received with no intervening Map Name GDS variables, the mapper uses the saved map name in mapping the new data. Once a Map Name GDS variable is received, that map name remains in effect until another map name is received or the mapped conversation ends.

When the effective map name is null (with a length of zero), mapping is said to be "off"; that is, any data passed to the mapper is returned unchanged. At the beginning of all mapped conversations, the effective map names are initialized to null. This happens prior to any flow of Map Name GDS variables. A Map Name GDS variable containing a null map name is sent to the partner only to change the effective map name back to null after it has not previously been null. If the transaction program always supplies a null map name, no Map Name GDS variable is ever sent to the partner LU.

MAPPER INVOCATION

PS.MC invokes the mapper whenever any of the following occurs:

- The transaction program sends or receives data (that is, issues MC_SEND_DATA or MC_RECEIVE_AND_WAIT). The data may be application data or FM header data; both of these types of data may be mapped.

- PS.MC receives, from PS.CONV, information indicating that the partner transaction program has received and processed all the recently sent map names. This includes information such as a positive reply to CONFIRM or to SYNCPT, or any information that the partner transaction program issued from SEND state (see explanation below).

The mapper is also invoked during the error processing triggered by the events listed below. This processing is more thoroughly described in "Mapper Errors" on page 5.2-12.

- The transaction program issues MC_SEND_ERROR.
- PS.MC issues SEND_ERROR with a type value of SVC (see SNA Transaction Programmer's Reference Manual for LU Type 6.2).
- The transaction program or the sync point manager ("Chapter 5.3. Presentation Services--Sync Point Services Verbs") issues BACKOUT.
- A return code of SVC_ERROR_* is received from PS.CONV.
- A return code of PROG_ERROR_* is received from PS.CONV.

A positive reply to CONFIRM or to SYNCPT informs the mapper that any map names it has caused to be sent to the partner have been received and processed by it. For example, if the mapper causes a Map Name GDS variable to be sent to the partner LU, and is informed that a positive reply to CONFIRM has been received, and is next invoked because the partner LU detected an error while in RECEIVE state, the mapper knows, because of the intervening confirmation, that the error processing at the partner did not cause the map name to be purged. The mapper does not cause a duplicate map name to be sent in this case.

In addition, receipt of data from the partner also indicates that all the recently sent map names have been processed, because the partner cannot have sent data unless it has entered SEND state, and it cannot have entered SEND state (from RECEIVE state, which is the state it was in when it was receiving and processing the data sent by the transaction program) unless it has finished receiving and mapping all the data that the transaction program was sending. Moreover, not only receipt of data, but receipt of any information whatsoever that the partner issued from SEND state (such as a SEND indicator, CONFIRM, or even an error notification) indicates to the mapper that the partner has received and processed the most recently sent map names.

MAPPER PARAMETERS

Each time PS.MC invokes the mapper, it supplies required information to the mapper. This information includes, in addition to the map name and the data to be mapped, such information as whether send or receive mapping is to be performed. Also, based upon the reason for the mapper invocation, information may be returned by the mapper to PS.MC. The mapper also uses other data structures in the RCB to store currently effective map names and incoming data. The information used and returned by the mapper is listed below. For a further description of mapper input and output, see the formal description of the UPM_MAPPER on page 5.2-46.

Supplied Information

- Reason for the mapper invocation
  - Data mapping
  - Errors
  - Positive confirmation
- Data polarity
  - Send
  - Receive
- FMH data indicator
- Input map name
- Input data
- Error code

Returned Information

- Output map name
- Output data (mapped data)
- Mapper return code

SEND MAPPING

When the transaction program is sending data (i.e., when PS.MC is processing MC_SEND_DATA), the mapper is responsible for:

- Mapping the data supplied by the transaction program (in the verb's DATA parameter) in accordance with the MAP_NAME parameter supplied by the transaction program
- Mapping the locally known map name supplied by the transaction program to a globally known map name corresponding to the format of the mapped data
- Determining whether to send a Map Name GDS variable to the partner LU, and preventing duplicate Map Name GDS variables from flowing consecutively to the partner LU

- Determining whether to resend a Map Name GDS variable to the partner LU, in the event of an error

PS.MC's processing of MC_SEND_DATA is described below. For example, the transaction program issues MC_SEND_DATA with MAP_NAME(A) and DATA(data-1). PS.MC invokes the mapper, informing it that send mapping is to be performed. PS.MC also passes to the mapper the supplied map name and data.

The mapper translates map name A to map name B and maps data-1 to data-2, to be sent to the partner LU. The translated map name, since it differs from the currently effective map name (which is stored in the MAPPER_SAVE_AREA and is initially null) is returned to PS.MC. The translated data is also returned.

When control is returned to PS.MC from the mapper call, PS.MC first determines whether the mapper succeeded in mapping the supplied data (it could have failed if the transaction program had provided a map name unknown to the mapper). Since the mapping was successful, PS.MC next determines whether a new map name has been returned. In this case, the mapper has returned the output map name, because the translated map name B differs from the currently effective map name. Therefore, PS.MC updates the currently effective map name to B and creates a Map Name GDS variable (to be sent to the partner) containing map name B. PS.MC next formats the data returned by the mapper as a an Application Data or User Control Data GDS variable, by segmenting it into logical records and prefixing the GDS ID. PS.MC uses the MC_MAX_SEND_SIZE field in the RCB to determine the size of the logical records.

Finally, PS.MC issues SEND_DATA, with a DATA parameter containing the Map Name and data GDS variables. When the SEND_DATA completes successfully, PS.MC returns control to the transaction program, indicating that the MC_SEND_DATA was also successful.

When the transaction program again issues MC_SEND_DATA, again specifying a map name of A, PS.MC again invokes the mapper. As in the previous invocation, the mapper translates map name A to map name B. Since it has already caused PS.MC to send map name B to the partner LU, it does not return an output map name to PS.MC.

Since no map name was returned from the mapper, PS.MC does not create a Map Name GDS variable. It processes the output data as above, creating an Application Data or User Control Data GDS variable containing the data. Finally, it issues SEND_DATA with a DATA parameter containing only the data GDS variable. An OK return code is returned on the SEND_DATA, and PS.MC returns a return code of OK on the MC_SEND_DATA.

RECEIVE MAPPING

When the transaction program is receiving data (i.e., when PS.MC is processing MC_RECEIVE_AND_WAIT), the mapper is responsible for

- Mapping the data received from the partner LU in accordance with the currently effective map name,
- Mapping the currently effective map name to a locally known map name corresponding to the format of the mapped data, and returning this map name and the mapped data to the transaction program, and
- Optionally, checking incoming Map Name GDS variables from the partner LU for duplication and symbol-string consistency.

An example of PS.MC's processing of MC_RECEIVE_AND_WAIT is described below.

First, PS.MC issues the basic conversation verb RECEIVE_AND_WAIT to PS.CONV, specifying a fill value of LL (see <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>) to request that PS.CONV return one logical record. After the RECEIVE_AND_WAIT completes successfully, PS.MC finds that the data received consists of a Map Name GDS variable. Knowing that a data GDS variable is to follow the map name, PS.MC again issues RECEIVE_AND_WAIT to PS.CONV, again retrieving one logical record. The data received in the second RECEIVE_AND_WAIT is application or FM header data, but the high-order bit of the LL field in the logical record indicates that more data follows that is to be associated with the data just received; that is, the data GDS variable consists of multiple logical records (see "Construction of GDS Variables" on page 5.2-5). PS.MC continues to request data from PS.CONV until the high-order bit of the LL field of the received logical record is <u>off</u>, indicating that the entire data GDS variable has been received. In the example, this occurs on the third RECEIVE_AND_WAIT.

PS.MC has now received a map name and data to be mapped. It invokes the mapper and receives from the mapper the map name and mapped data to be passed to the transaction program. PS.MC passes to the transaction program the amount of data that the transaction program has requested, and discards any remaining data.

<u>MC_TEST_PROC</u>

An implementation of the mapped conversation verbs includes an entry point, MC_TEST_PROC, which can be used to determine whether a complete data GDS variable has been received from the remote TP without causing the calling program to wait if data is not available immediately. This entry point is called by the implementation of the WAIT verb, which enables a TP to wait for arrival of data on any of a list of basic and mapped conversations.

An MC_POST_ON_RECEIPT verb must be issued before a call to MC_PROC_TEST is effective. Thus, MC_POST_ON_RECEIPT must be issued before a WAIT verb that includes a mapped conversation in its list. Then a sequence of calls can be made to MC_TEST_PROC, which returns the code OK when a complete data GDS variable is available.

In order to determine whether a complete data GDS variable has been received from the remote TP, MC_TEST_PROC has to issue a RECEIVE_AND_WAIT verb, so that it can examine the data. Several RECEIVE_AND_WAIT verbs may be required before a complete data GDS variable is received. As the pieces of the data GDS variable are received, they are placed in an RCB field, MC_RECEIVE_BUFFER, where they are held until the local TP issues an MC_RECEIVE_AND_WAIT verb.

To make sure that the RECEIVE_AND_WAIT verbs that it issues do not cause waits for data to be received from the remote TP, MC_TEST_PROC calls a similar entry point of PS.CONV, TEST_PROC, to determine whether a logical record has already been received. Only when such a record is available does it issue a RECEIVE_AND_WAIT verb.

An example of the use of MC_TEST_PROC is illustrated in Figure 5.2-6 on page 5.2-13 and described below. This figure begins with the TP issuing an MC_POST_ON_RECEIPT verb for a specified mapped conversation. It then issues a WAIT verb, which causes the PS.VERB_ROUTER to call MC_TEST_PROC for the specified conversation, as well as others. MC_TEST_PROC first checks the MC_RECEIVE_BUFFER in the RCB associated with the conversation to see if it holds a complete data GDS variable. In this example, PS.MC does not have a data GDS variable ready. Therefore, MC_TEST_PROC calls TEST_PROC to determine whether PS.CONV has any data ready to be received. PS.CONV returns to PS.MC with a code indicating that data is available, WHAT_RECEIVED = DATA_COMPLETE. PS.MC issues RECEIVE_AND_WAIT to retrieve the waiting data. After inspecting the data, PS.MC discovers that it is not sufficient to complete the current data GDS variable. PS.MC stores the received data in MC_RECEIVE_BUFFER, issues POST_ON_RECEIPT to request that PS.CONV reinitiate posting, and returns the code UNSUCCESSFUL to PS.VERB_ROUTER. PS.VERB_ROUTER resumes testing this resource and all others specified in the WAIT verb for receipt of a complete data GDS variable.

In this example, had the call to TEST_PROC returned any code other than OK--DATA, PS.MC would not issue RECEIVE_AND_WAIT but would return to PS.VERB_ROUTER the same code that it received from TEST_PROC. On the other hand, had the data returned by RECEIVE_AND_WAIT completed a data GDS variable, MC_TEST_PROC would not have issued POST_ON_RECEIPT but would have returned the code OK--DATA to PS.VERB_ROUTER.

When MC_TEST_PROC is called, MC_RECEIVE_BUFFER is in one of the following states:

- It is empty.
- It contains the initial logical records of a data GDS variable (perhaps preceded by an associated map name GDS variable), but does not yet contain the remaining logical records of the data GDS variable, which must be received before the data can be passed to the transaction program.
- It contains a complete data GDS variable that is ready to be mapped and passed to the transaction program.

Once a complete data GDS variable has been received, PS.MC requests no more information from PS.CONV until it passes to the transaction program the data already in MC_RECEIVE_BUFFER.

MC_RECEIVE_BUFFER may contain many different types of information. It may contain transient information, such as a return code or a SEND indicator, which is returned to the transaction program as soon as processing of the current verb is completed. It may contain part or all of a data GDS variable. These logical records remain in the list until the incoming data GDS variable is complete and is retrieved by RECEIVE_AND_WAIT.

MAPPED CONVERSATION ERRORS

MAPPER ERRORS

In send mapping, the supplied map name is not checked for symbol-string consistency; its symbol-string restrictions, if any, are implementation-defined. The mapper translates the supplied map name to a globally known map name that conforms to the symbol-string definitions in the SNA Transaction Programmer's Reference Manual for LU Type 6.2. PS.MC, therefore, also performs no checking of the globally known map name returned by the mapper; the mapper is responsible for supplying map names that conform to SNA-defined formats. In receive mapping, however, the mapper does check the map name received in a Map Name GDS variable for symbol-string consistency. The mapper informs PS.MC via a return code of MAP_NOT_FOUND when the map name violates SNA-defined symbol-string types, or when the map name conforms to defined symbol-string types but is unknown to the mapper (see SNA

```
 TP                 PS.VERB_ROUTER                   PS.MC                  PS.CONV
 .                        .                            .                        .
 .                        .                            .                        .
 .               MC_POST_ON_RECEIPT                    . POST_ON_RECEIPT (FILL = LL) .
 o---------------------------------------------------->o----------------------->o
 .                        .                            .                        .
 o<- - - - - - - - - - - - - - - - - - - - - - - - - - o<- - - - - - - - - - - - o
 .                        .                            .                        .
 .                        .                            .                        .
 .        WAIT            .      Call MC_TEST_PROC     .                        .
 o----------------------->o--------------------------->o                        .
 .                        .                            .                        .
 .                        .        (MC_RECEIVE_BUFFER does not hold             .
 .                        .          a complete data GDS variable)              .
 .                        .                            .                        .
 .                        .                            .      Call TEST_PROC    .
 .                        .                            o----------------------->o
 .                        .                            .                        .
 .                        .                            .  return code = OK--DATA .
 .                        .                            o<- - - - - - - - - - - - o
 .                        .                            .                        .
 .                        .                  (PS.CONV has data)                 .
 .                        .                            .                        .
 .                        .                            . RECEIVE_AND_WAIT (FILL = LL).
 .                        .                            o----------------------->o
 .                        .                            .                        .
 .                        .                            .WHAT_RECEIVED = DATA_COMPLETE.
 .                        .                            o<- - - - - - - - - - - - o
 .                        .                            .                        .
 .                        .               (more data to be received)            .
 .                        .                            .                        .
 .                        .                            . POST_ON_RECEIPT (FILL = LL) .
 .                        .                            o----------------------->o
 .                        .                            .                        .
 .                        .   return code = UNSUCCESSFUL .                      .
 .                        o<- - - - - - - - - - - - - -o<- - - - - - - - - - - - o
 .                        .                            .                        .
 .          Continue testing for posting by            .                        .
           any resource specified in the verb
```

See "MC_TEST_PROC" on page 5.2-11 for an explanation of the flows shown in this figure.

Note: Only those parameters pertinent to the example are shown.

Figure 5.2-6. MC_TEST_PROC

Formats for definitions of the valid symbol-string types).

The mapper also performs an optional receive check to determine if it has received a map name that is the duplicate of the map name last received. If it has, then the mapper informs PS.MC, which ends the mapped conversation. See "Protocol Violations" on page 5.2-14 for details.

If notification of an error is received, PS.MC passes the error notification to the transaction program as a return code. In addition, PS.MC invokes the mapper to inform it of the error. The mapper then determines whether a map name needs to be re-sent, since the MC_SEND_ERROR issued by the partner transaction program or PS.MC might have caused the map name to be purged on receipt. If notification of an error is received and the mapper has previously caused PS.MC to send a map name to the partner LU, the mapper checks to see if any information has been received that would indicate that the partner LU has received and processed the map name. Examples of the type of information that would indicate this are an affirmative reply to CONFIRM or to SYNCPT, received data, or a SEND indicator. If none of the above has been received, the mapper causes a map name to be re-sent to the partner LU. The map name that is sent is based upon the map name supplied by the transaction program on the next MC_SEND_DATA.

The mapper needs to be informed of any errors that occur on a mapped conversation, and of any issuances of BACKOUT that occur on a mapped conversation whose synchronization level is SYNCPT, because these events may require the mapper to re-send the currently effective map name. In the case of an error detected by the partner LU, a map name that the mapper has sent to the partner may have been purged by the partner as a result of its error processing. Therefore, the mapper has to determine whether it needs to re-send the map name that may have been purged. In the case of BACKOUT, the entire mapped conversation is required to revert to the status it had at the last issuance of SYNCPT. If the currently effective map name has changed since then, the mapper needs to resend the map name that was in effect at the last issuance of SYNCPT.

ERROR DATA GDS VARIABLES

A GDS variable that is not created as a direct result of action taken by the transaction program is the Error Data GDS variable. When PS.MC detects an error in the data being received from the partner LU, it issues a SEND_ERROR TYPE(SVC) followed by a SEND_DATA. The DATA parameter of the SEND_DATA contains the Error Data GDS variable, which describes the exact nature of the error encountered. The transaction program serviced by the PS.MC that received the data and detected the error is not informed of the error. The transaction program that issued the data in which an error was found is told of the error via a return code derived from the information contained in the Error Data GDS variable (see "Processing of a Service Error Detected by Partner LU" on page 5.2-17). An example of the type of error that PS.MC might encounter in received data is receipt of a User Control Data GDS variable when FM header data is not supported by the transaction program or the LU.

PROTOCOL VIOLATIONS

PS.MC performs optional receive checks to determine if the partner LU has committed a protocol violation. An example of a protocol violation PS.MC can detect is the receipt of a Map Name GDS variable followed by something other than a data GDS variable (map names have to be followed by data).

When PS.MC detects a protocol violation such as the one above, it issues DEALLOCATE with TYPE(ABEND_SVC) and returns a return code of RESOURCE_FAILURE_NO_RETRY to the transaction program. Correspondingly, when PS.MC receives a return code of DEALLOCATE_ABEND_SVC or DEALLOCATE_ABEND_TIMER from PS.CONV, it translates the return code to RESOURCE_FAILURE_NO_RETRY, and passes it to the transaction program.

If, however, the protocol violation occurred because the mapped conversation ended prematurely at the partner LU (i.e., the partner LU has issued a deallocation notification that indicates a protocol error), then PS.MC simply logs the error and passes the RESOURCE_FAILURE_NO_RETRY return code to the transaction program. Since the mapped conversation has already been deallocated at the partner LU, PS.MC cannot issue the DEALLOCATE (TYPE=ABEND_SVC) that it normally issues when it detects a protocol violation.

SERVICE ERRORS

The TP, upon detecting an error on a mapped conversation, issues MC_SEND_ERROR with TYPE(PROG). This indicates that the type of error detected was a program error (i.e., was an error discovered by a TP). Another category of errors may be detected by the LU rather than the TP. These errors are called service errors because they are detected by a presentation services component within the LU.

As a service component, PS.MC checks for certain types of service errors. If a partner TP requests a function, such as handling of function management header (FMH) data, that is not supported by the local LU or transaction program, PS.MC performs service error processing and advises the partner LU of the lack of support for that function.

Another service error that PS.MC may detect is receipt of a map name from the partner LU that is not known by the mapper. Similarly the mapper may find that the data and the map name it has received from the partner LU are incompatible, i.e., that the data cannot be mapped using the received map name.

PS.MC also handles receipt of a service error notification from a partner LU when it is the partner that discovered the service error.

The following sections describe the processing that PS.MC performs when it detects a service error, and the processing that results when PS.MC learns that the partner detected an error.

SERVICE ERRORS DETECTED IN RECEIVED DATA

As mentioned earlier, one type of error that PS.MC may detect is receipt of an invalid map name. Figure 5.2-7 on page 5.2-15 illustrates this service error. In the figure, PS.MC has issued a RECEIVE_AND_WAIT to PS.CONV as a result of the MC_RECEIVE_AND_WAIT issued by the TP. The data returned in the RECEIVE_AND_WAIT is a Map Name GDS variable. PS.MC stores the map name and issues another RECEIVE_AND_WAIT in order to receive the data that follows the map name. In this example, PS.MC receives a complete data GDS variable in the RECEIVE_AND_WAIT (and therefore does not retrieve any more data from PS.CONV).

PS.MC invokes the mapper, passing it the received map name and data. Instead of mapping the data, however, the mapper returns to

```
TP                        PS.MC                          MAPPER                        PS.CONV
 .                          .                               .                              .
 .   MC_RECEIVE_AND_WAIT    .        RECEIVE_AND_WAIT (FILL = LL)                          .
 o------------------------->o----------------------------------------------------------->o
 .                          .                               .                              .
 .                          .        WHAT_RECEIVED = DATA_COMPLETE                         .
 .                          o< - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - o
 .                          .                               .                              .
 .        (data is a map name GDS variable)                 .                              .
 .                          .        RECEIVE_AND_WAIT (FILL = LL)                          .
 .                          o----------------------------------------------------------->o
 .                          .                               .                              .
 .                          .        WHAT_RECEIVED = DATA_COMPLETE                         .
 .                          o< - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - o
 .                          .                               .                              .
 .       (data is a complete data GDS variable)             .                              .
 .                          .                               .                              .
 .                          .     INPUT_DATA=data-1         .                              .
 .                          .  INPUT_MAP_NAME=map-name-1    .                              .
 .                          o----------------------------->o                              .
 .                          .                               .                              .
 .                          .  RETURN_CODE=MAP_NOT_FOUND    .                              .
 .                          o< - - - - - - - - - - - - - - o                              .
 .                          .                               .                              .
 .                          .          SEND_ERROR (TYPE = SVC)                             .
 .                          o----------------------------------------------------------->o
 .                          .                               .                              .
 .                          .             RETURN_CODE = OK                                 .
 .                          o< - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - o
 .                          .                               .                              .
 .                          .   SEND_DATA (DATA = Error Data GDS variable)                 .
 .                          o----------------------------------------------------------->o
 .                          .                               .                              .
 .                          .             RETURN_CODE = OK                                 .
 .                          o< - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - o
 .                          .                               .                              .
 .                          .      PREPARE_TO_RECEIVE (TYPE = FLUSH)                       .
 .                          o----------------------------------------------------------->o
 .                          .                               .                              .
 .                          .             RETURN_CODE = OK                                 .
 .                          o< - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - o
 .                          .                               .                              .
 .                          .                               .                              .
 .                          .             RECEIVE_AND_WAIT                                 .
 .                          o----------------------------------------------------------->o
 .                          .                               .                              .
 .                          .        WHAT_RECEIVED = DATA_COMPLETE                         .
 .                          o< - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - o
 .                          .                               .                              .
 .                          .                               •                              .
 .                          .                               •                              .
 .                          .                               •                              .
```

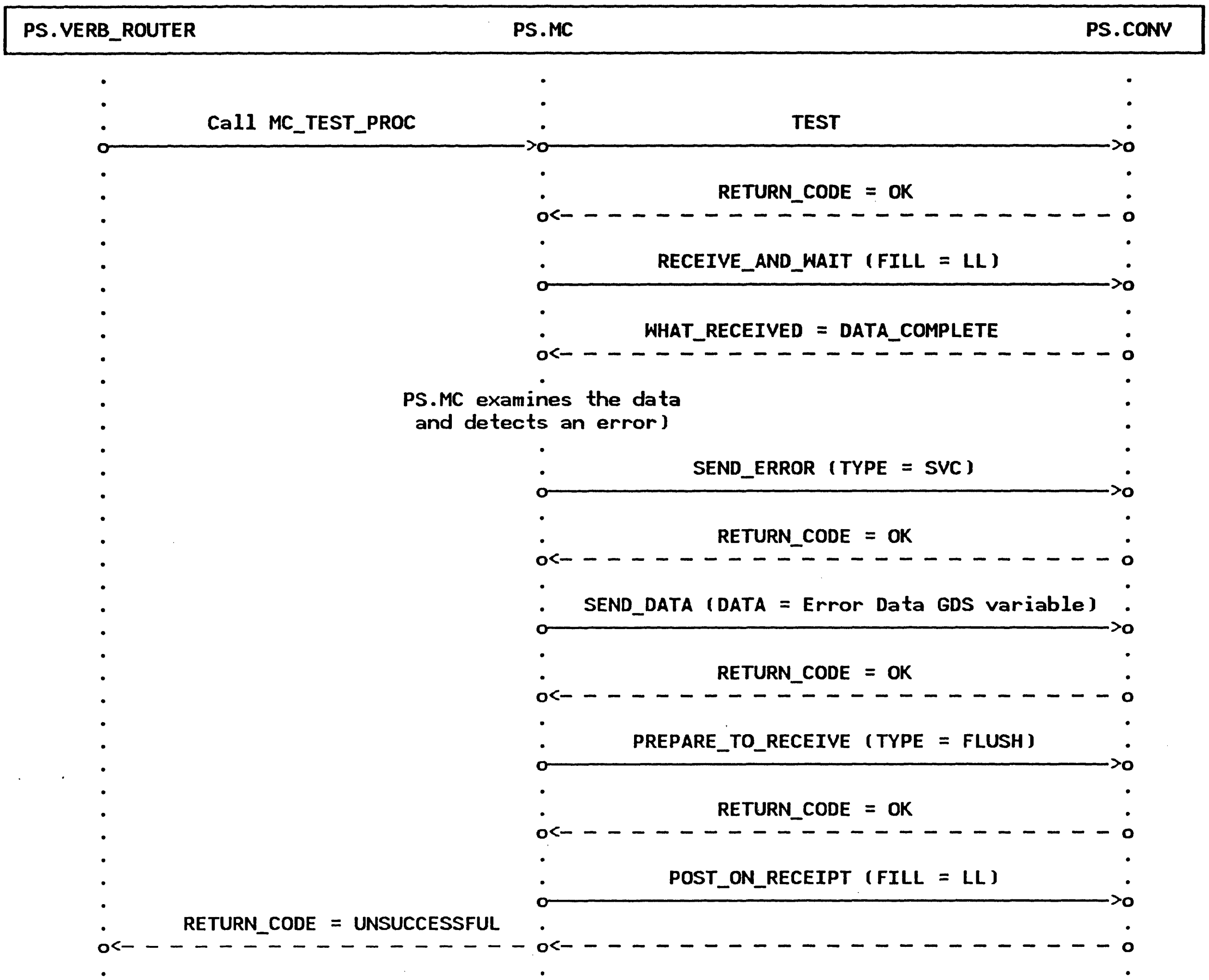
See "Service Errors Detected in Received Data" for an explanation of the flows shown in this figure.

Figure 5.2-9 on page 5.2-18 is the complement of this figure, showing the processing that occurs when an LU is informed of an error committed at that LU. Note: Only those parameters pertinent to the example are shown.

Figure 5.2-7. Detecting a Service Error as a Result of MC_RECEIVE_AND_WAIT Processing

PS.MC a return code indicating that the map name received is invalid. The mapper has detected a service error and informed PS.MC of the error.

PS.MC now has to inform the partner that a service error occurred and to return SEND control of the mapped conversation to the partner TP. PS.MC first issues SEND_ERROR with TYPE(SVC). This tells the partner LU only that an error occurred; it does not



See "Service Errors Detected in Received Data" for an explanation of the flows shown in this figure.

Note: Only those parameters pertinent to the example are shown.

Figure 5.2-8. Detecting a Service Error as a Result of a Call to MC_TEST_PROC

indicate to the partner the exact nature of the error. In order to convey this important information to the partner, PS.MC creates an Error Data GDS variable. The GDS variable carries an indication that the received map name was not found in the mapper's library of map names; the invalid map name is also returned to the partner LU in the Error Data GDS variable so that the partner LU will know exactly which map name was unknown. PS.MC then issues a SEND_DATA carrying the Error Data GDS variable to PS.CONV.

PS.MC completes its processing of the received service error by issuing PREPARE_TO_RECEIVE with TYPE(FLUSH), which returns SEND control of the mapped conversation to the partner TP.

PS.MC does not inform its local TP of the service error committed by the partner LU. It instead returns SEND control of the mapped conversation to the partner TP, which is informed of the error, and waits for the partner TP to recover from the error. The transaction program that committed the error is responsible for determining what error recovery is to take place. When the service error is detected as a result of an MC_RECEIVE_AND_WAIT, PS.MC immediately issues another RECEIVE_AND_WAIT to wait for information from the partner.

Figure 5.2-8 illustrates a slightly different situation in which a service error is detected. This time, the error is detected in data that was received as a result of a call to MC_TEST_PROC by the PS.VERB_ROUTER while it is processing a WAIT verb. Another difference is that instead of the mapper detecting the error, PS.MC discovers it. One cause of this type of error would be incoming data requesting a function that the receiving PS.MC did not support (for example, the function of handling FM header data when User Control Data GDS variables are not supported by the receiving PS.MC).

In handling this error during a call to MC_TEST_PROC, PS.MC, as in the MC_RECEIVE_AND_WAIT example, issues SEND_ERROR, followed by SEND_DATA with an Error Data GDS variable, followed by PREPARE_TO_RECEIVE with TYPE(FLUSH). PS.MC then continues, however, in a manner different from the MC_RECEIVE_AND_WAIT example: MC_TEST_PROC returns to the PS.VERB_ROUTER, after passing SEND control of the mapped conversation to the partner (and after causing posting to be re-activated). The PS.VERB_ROUTER is informed that its MC_TEST was unsuccessful, but not of the specific error.

PROCESSING OF A SERVICE ERROR DETECTED BY PARTNER LU

PS.MC also handles service errors that are detected by the partner LU. The error could have been detected in data sent to the partner LU by the local TP. Alternatively, the partner LU may have detected an error while sending data to PS.MC. Figure 5.2-9 on page 5.2-18 and Figure 5.2-10 on page 5.2-19 illustrate these two cases of error notification.

In Figure 5.2-9 on page 5.2-18, the transaction program is in the midst of sending data to the partner transaction program. However, a return code of SVC_ERROR_PURGING is returned on one of the SEND_DATAs that PS.MC issues to PS.CONV. The SVC_ERROR_PURGING return code indicates that the partner LU has detected an error in the data it has received. PS.MC, upon receipt of the SVC_ERROR_PURGING return code, issues a RECEIVE_AND_WAIT to learn the type of service error the partner LU encountered. The data returned in the RECEIVE_AND_WAIT consists of an Error Data GDS variable specifying the type of service error. The return code that PS.MC returns to the transaction program is derived from the information carried in the Error Data GDS variable. Before returning to the transaction program, PS.MC issues another RECEIVE_AND_WAIT to retrieve the SEND indicator. As discussed in the previous section, the transaction program that caused a service error to be committed is responsible for determining what error recovery is to occur. PS.MC returns to the transaction program with a return code, in this example, of MAP_NOT_FOUND. The transaction program still has SEND control of the mapped conversation (the transaction program is placed in SEND state as a result of a remotely detected error, even if the transaction program was in RECEIVE state when it issued the verb on which the error is reported).

The example shown in Figure 5.2-7 on page 5.2-15 and described in "Processing of a Service Error Detected by Partner LU" is the complement of the example just discussed and shown in Figure 5.2-9 on page 5.2-18. The first figure mentioned shows a transaction program requesting to receive data on a mapped conversation and the LU detecting an error in the data received. The second figure shows a transaction program sending data on a mapped conversation and being notified that a problem with the data was encountered at the partner LU.

As was pointed out in "Block Mapping" on page 5.2-8, PS.MC never sends a service-error notification to its partner from SEND state. An LU providing implementation-defined mapping, however, could issue such an error. For example, the LU may have mapped some, but not all, of the data issued by the transaction program in an MC_SEND_DATA. The part of the data that has been mapped is sent on the mapped conversation. While mapping the remainder of the data, however, the mapper discovers a problem. It informs its PS.MC component, which then issues a service-error notification indicating that data truncation has occurred at the sending LU. An LU with implementation-defined mapping may also, at some point, need to notify its partner that an error was detected but no data truncation has occurred.

While PS.MC does not issue service errors from SEND state, it does handle receipt of notifications that the partner LU detected a service error while it was in SEND state. Figure 5.2-10 on page 5.2-19 illustrates the processing that PS.MC performs as a result of this error. If it has received any incomplete data prior to receiving the service-error notification, PS.MC purges the data and immediately begins to wait for new data to arrive. Again, the transaction program is not informed of the error.

```
TP                                 PS.MC                                    PS.CONV
.                                    .                                         .
.          MC_SEND_DATA              .               SEND_DATA                 .
o----------------------------------->o---------------------------------------->o
.                                    .                                         .
.        RETURN_CODE = OK            .            RETURN_CODE = OK             .
o< - - - - - - - - - - - - - - - - - o<- - - - - - - - - - - - - - - - - - - - o
                                     .
                                     .
                                     .
           MC_SEND_DATA                              SEND_DATA
o----------------------------------->o---------------------------------------->o
.                                    .                                         .
.                                    .      RETURN_CODE = SVC_ERROR_PURGING    .
.                                    o<- - - - - - - - - - - - - - - - - - - - o
.                                    .                                         .
.                                    .            RECEIVE_AND_WAIT             .
.                                    o---------------------------------------->o
.                                    .                                         .
.                                    .      WHAT_RECEIVED = DATA_COMPLETE      .
.                                    o<- - - - - - - - - - - - - - - - - - - - o
.                                    .                                         .
.                   (data is an error.data GDS variable)                       .
.                                    .                                         .
.                                    .            RECEIVE_AND_WAIT             .
.                                    o---------------------------------------->o
.                                    .                                         .
.   RETURN_CODE = MAP_NOT_FOUND      .          WHAT_RECEIVED = SEND           .
o< - - - - - - - - - - - - - - - - - o<- - - - - - - - - - - - - - - - - - - - o
.                                    .                                         .
```

See "Processing of a Service Error Detected by Partner LU" for an explanation of the flows that are shown in this figure.

Figure 5.2-7 on page 5.2-15 is the complement of this figure, showing the processing that occurs at the LU that detects an error in received data. The SVC_ERROR_PURGING return code can be returned on several verbs. SEND_DATA is used here as an example of one of the verbs possible.

Note: Only those parameters pertinent to the example are shown.

Figure 5.2-9. Receipt by PS.MC of a SVC_ERROR_PURGING Return Code

```
 TP                                 PS.MC                                   PS.CONV

  .                                   .                                       .
  .                                   .                                       .
  .      MC_RECEIVE_AND_WAIT          .      RECEIVE_AND_WAIT (FILL = LL)     .
  o---------------------------------->o-------------------------------------->o
  .                                   .                                       .
  .                                   .RETURN_CODE = SVC_ERROR_TRUNC/SVC_ERROR_NO_TRUNC
  .                                   o<- - - - - - - - - - - - - - - - - - - o
  .                                   .                                       .
  .                       (PS.MC purges any data that it has received         .
  .                        prior to the service error notification)           .
  .                                   .                                       .
  .                                   .            RECEIVE_AND_WAIT           .
  .                                   o-------------------------------------->o
  .                                                                           .
                                    • • •
```

See "Processing of a Service Error Detected by Partner LU" for an explanation of the flows that are shown in this figure.

The processing that occurs when a SVC_ERROR_TRUNC or SVC_ERROR_NO_TRUNC return code is received by PS.MC while processing a call to MC_TEST_PROC differs from this figure only in that PS.MC does not issue a RECEIVE_AND_WAIT after receiving the return code. PS.MC returns a code of UNSUCCESSFUL to the PS.VERB_ROUTER.

Note: Only those parameters pertinent to the example are shown.

Figure 5.2-10. Receipt by PS.MC of a SVC_ERROR_TRUNC orSVC_ERROR_NO_TRUNC Return Code

PS_MC

FUNCTION:	This procedure receives mapped conversation verbs issued by the transaction program, and routes each verb to the appropriate procedure for processing. PS.MC is called by PS.VERB_ROUTER (Chapter 5.0) as a result of the transaction program's issuing a mapped conversation verb.
INPUT:	The current transaction program verb is passed with parameters; PS_PROCESS_DATA is provided by the resources manager at creation time and may be accessed by all the procedures within PS.
OUTPUT:	Refer to the procedures that are called from this procedure for the specific outputs.

Referenced procedures, FSMs, and data structures:

MC_ALLOCATE_PROC	page 5.2-21
MC_CONFIRM_PROC	page 5.2-21
MC_CONFIRMED_PROC	page 5.2-22
MC_DEALLOCATE_PROC	page 5.2-23
MC_FLUSH_PROC	page 5.2-23
MC_GET_ATTRIBUTES_PROC	page 5.2-24
MC_POST_ON_RECEIPT_PROC	page 5.2-25
MC_PREPARE_TO_RECEIVE_PROC	page 5.2-26
MC_RECEIVE_AND_WAIT_PROC	page 5.2-27
MC_REQUEST_TO_SEND_PROC	page 5.2-37
MC_SEND_DATA_PROC	page 5.2-38
MC_SEND_ERROR_PROC	page 5.2-40
MC_TEST_PROC	page 5.2-28

```
Select based on the mapped conversation verb (issued by the TP):
  When MC_ALLOCATE
    Call MC_ALLOCATE_PROC (page 5.2-21).
  When MC_CONFIRM
    Call MC_CONFIRM_PROC (page 5.2-21).
  When MC_CONFIRMED
    Call MC_CONFIRMED_PROC (page 5.2-22).
  When MC_DEALLOCATE
    Call MC_DEALLOCATE_PROC (page 5.2-23).
  When MC_FLUSH
    CALL MC_FLUSH_PROC (page 5.2-23).
  When MC_GET_ATTRIBUTES
   Call MC_GET_ATTRIBUTES_PROC (page 5.2-24).
  When MC_POST_ON_RECEIPT
   Call MC_POST_ON_RECEIPT_PROC (page 5.2-25).
  When MC_PREPARE_TO_RECEIVE
   Call MC_PREPARE_TO_RECEIVE_PROC (page 5.2-26).
  When MC_RECEIVE_AND_WAIT
   Call MC_RECEIVE_AND_WAIT_PROC (page 5.2-27).
  When MC_REQUEST_TO_SEND
   Call MC_REQUEST_TO_SEND_PROC (page 5.2-37).
  When MC_SEND_DATA
    Call MC_SEND_DATA_PROC (page 5.2-38).
  When MC_SEND_ERROR
    Call MC_SEND_ERROR_PROC (page 5.2-40).
  When MC_TEST
    Call MC_TEST_PROC (page 5.2-28).
```

MC_ALLOCATE_PROC

FUNCTION:	This procedure handles the allocation of mapped conversations.
INPUT:	MC_ALLOCATE verb parameters (See SNA Transaction Programmer's Reference Manual for LU Type 6.2.)
OUTPUT:	A return code as described in SNA Transaction Programmer's Reference Manual for LU Type 6.2. Also, if the allocation is successful, PS.MC returns the ID of this RCB.
NOTES:	1. The SNASVCMG mode name is not allowed at the mapped conversation protocol boundary.
	2. A return code on ALLOCATE of PARAMETER_ERROR, PROGRAM_PARAMETER_CHECK, or UNSUCCESSFUL indicates that no resource has been allocated (and, therefore, no RCB has been created). When the ALLOCATE returns a RETURN_CODE value of OK or ALLOCATION_ERROR, an RCB has been created.

Referenced procedures, FSMs, and data structures:

ALLOCATE_PROC	page 5.1-11
RCB	page A-6

If the transaction program supports mapped conversations and the mode name is
 not SNASVCMG (See Note 1) then
   Call ALLOCATE_PROC(verb parameters) (Chapter 5.0) to issue an ALLOCATE verb with
    the MC_ALLOCATE verb parameters and specifying that the conversation type is mapped.
   Set the MC_ALLOCATE parameters to the values returned by the ALLOCATE verb.

Else (allocation of a conversation is not allowed)
   Set RETURN_CODE to PROGRAM_PARAMETER_CHECK.

MC_CONFIRM_PROC

FUNCTION:	This procedure processes MC_CONFIRM verbs.
INPUT:	MC_CONFIRM verb parameters (See SNA Transaction Programmer's Reference Manual for LU Type 6.2.)
OUTPUT:	A return code as described in SNA Transaction Programmer's Reference Manual for LU Type 6.2. If a request to send is received from the remote transaction program while processing a CONFIRM verb, this request is also indicated to the local TP.
NOTES:	1. PS.MC performs no check to determine if the conversation is in an appropriate state to receive an MC_CONFIRM verb. A state check is performed by PS.CONV (Chapter 5.1) during its processing of the CONFIRM verb.
	2. The processing that PS.MC performs as a result of receiving a return code of SVC_ERROR_PURGING involves issuing the necessary RECEIVE_AND_WAIT verbs. A request to send by the remote TP may be indicated by one of these RECEIVE_AND_WAIT verbs, as well as by the CONFIRM verb. In either case, the indication is passed to the local TP.

Referenced procedures, FSMs, and data structures:

CONFIRM_PROC	page 5.1-12
PS_SPS	page 5.3-35
RCVD_SVC_ERROR_PURGING	page 5.2-42
UPM_MAPPER	page 5.2-46
RCB	page A-6

Call CONFIRM_PROC(CONFIRM)(page 5.1-12) to issue a CONFIRM verb with
 the MC_CONFIRM verb parameters.
Set the MC_CONFIRM parameters and return code to the values returned by the CONFIRM verb.

```
Select based on the return code copied into MC_CONFIRM:
   When OK
      Call UPM_MAPPER (page 5.2-46) to record a positive confirmation.
   When PROG_ERROR_PURGING
      Call UPM_MAPPER (page 5.2-46) to record a remotely detected
       error of the type indicated by the return code from CONFIRM.
   When DEALLOCATE_ABEND_PROG
      Set RETURN_CODE to DEALLOCATE_ABEND.
   When DEALLOCATE_ABEND_SVC or DEALLOCATE_ABEND_TIMER
      Set the RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
   When BACKED_OUT
      Call PS_SPS (sync point manager, Chapter 5.3).
   When SVC_ERROR_PURGING
      Call RCVD_SVC_ERROR_PURGING (page 5.2-42) to
       get and process error data from the remote TP.
      Set RETURN_CODE to the value returned by RCVD_SVC_ERROR_PURGING.
If a request to send has been received from the remote TP and not
 indicated on a prior MC_CONFIRM, MC_RECEIVE_AND_WAIT, MC_SEND_DATA, or
 MC_SEND_ERROR verb then
   Return a request to send received indication to the local TP.
```

MC_CONFIRMED_PROC

FUNCTION:	This procedure processes MC_CONFIRMED verbs.
INPUT:	MC_CONFIRMED verb parameters (See SNA Transaction Programmer's Reference Manual for LU Type 6.2.)
NOTE:	PS.MC performs no check to determine if the conversation is in an appropriate state to receive an MC_CONFIRMED. A state check is performed by PS.CONV (Chapter 5.1) during its processing of the CONFIRMED verb.

Referenced procedures, FSMs, and data structures:

CONFIRMED_PROC	page 5.1-14

Call CONFIRMED_PROC(CONFIRMED)(page 5.1-14) to issue a CONFIRMED verb with
 the MC_CONFIRMED verb parameters.
Set the MC_CONFIRM return code to the value returned by the CONFIRMED verb.

MC_DEALLOCATE_PROC

FUNCTION:	This procedure handles the deallocation of mapped conversation resources.
INPUT:	MC_DEALLOCATE verb parameters (See SNA Transaction Programmer's Reference Manual for LU Type 6.2.)
OUTPUT:	A return code as described in SNA Transaction Programmer's Reference Manual for LU Type 6.2.
NOTE:	PS.MC performs no check to determine if the conversation is in an appropriate state to receive an MC_DEALLOCATE. A state check is performed by PS.CONV (Chapter 5.1) during its processing of the DEALLOCATE verb.

Referenced procedures, FSMs, and data structures:

UPM_MAPPER	page 5.2-46
RCVD_SVC_ERROR_PURGING	page 5.2-42
DEALLOCATE_PROC	page 5.1-15
RCB	page A-6

```
Find the RCB for the conversation identified in the RESOURCE parameter.
If the deallocation type is ABEND then
   Clear RCB.MC_RECEIVE_BUFFER.
   Call DEALLOCATE_PROC(DEALLOCATE)(page 5.1-15) to issue a DEALLOCATE verb with
    the MC_DEALLOCATE verb parameters, no error data, and deallocation type ABEND_PROG.
Else
   Call DEALLOCATE_PROC(DEALLOCATE)(page 5.1-15) to issue a DEALLOCATE verb with
    the MC_DEALLOCATE verb parameters, no error data, and deallocation type specified.
Set the MC_DEALLOCATE parameters and return code to the values returned by the DEALLOCATE verb.

Select based on the return code copied into MC_DEALLOCATE:
   When PROG_ERROR_PURGING
      Set RETURN_CODE to the code returned by DEALLOCATE.
      Call UPM_MAPPER (page 5.2-46) to record a
       remotely detected error of the type indicated by the return code
       from the DEALLOCATE verb.
   When DEALLOCATE_ABEND_PROG
      Set RETURN_CODE to DEALLOCATE_ABEND.
   When DEALLOCATE_ABEND_SVC or DEALLOCATE_ABEND_TIMER
      Set RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
   When SVC_ERROR_PURGING
      Call RCVD_SVC_ERROR_PURGING (page 5.2-42).
```

MC_FLUSH_PROC

FUNCTION:	This procedure processes MC_FLUSH verbs.
INPUT:	MC_FLUSH verb parameters (See SNA Transaction Programmer's Reference Manual for LU Type 6.2.)
NOTE:	PS.MC performs no check to determine if the conversation is in an appropriate state to receive an MC_FLUSH. A state check is performed by PS.CONV (Chapter 5.1) during its processing of the FLUSH verb.

Referenced procedures, FSMs, and data structures:

FLUSH_PROC	page 5.1-16

```
Call FLUSH_PROC(FLUSH)(page 5.1-16) to issue a FLUSH verb with
 the MC_FLUSH verb parameters.
Set the MC_FLUSH return code to the value returned by the FLUSH verb.
```

MC_GET_ATTRIBUTES_PROC

FUNCTION:	This procedure handles requests from the transaction program for information about a mapped conversation.
INPUT:	MC_GET_ATTRIBUTES verb parameters (See SNA Transaction Programmer's Reference Manual for LU Type 6.2.)
OUTPUT:	PS.MC issues a GET_ATTRIBUTES (See SNA Transaction Programmer's Reference Manual for LU Type 6.2) verb for the resource specified in MC_GET_ATTRIBUTES. PS.MC places the information returned in the GET_ATTRIBUTES verb into the appropriate fields in the MC_GET_ATTRIBUTES and returns control to the transaction program.

Referenced procedures, FSMs, and data structures:

GET_ATTRIBUTES_PROC page 5.1-17

Find the RCB for the conversation identified in the RESOURCE parameter.
Call GET_ATTRIBUTES_PROC(GET_ATTRIBUTES)(page 5.1-17) to issue a GET_ATTRIBUTES verb with the MC_GET_ATTRIBUTES verb parameters.
Set the MC_GET_ATTRIBUTES parameters and return code to the values returned by the GET_ATTRIBUTES verb, to the TP, such as the fully qualified LU names of both LUs of the conversation, the mode name, synchronization level, security profile, and security user ID.

MC_POST_ON_RECEIPT_PROC

FUNCTION: This procedure processes MC_POST_ON_RECEIPT verbs.

INPUT: MC_POST_ON_RECEIPT verb parameters (See SNA Transaction Programmer's Reference Manual for LU Type 6.2.)

OUTPUT: If the MC_RECEIVE_BUFFER is empty when the MC_POST_ON_RECEIPT is issued, PS.MC issues a POST_ON_RECEIPT verb. Otherwise, no POST_ON_RECEIPT is necessary (see below).

NOTES:
1. If the MC_RECEIVE_BUFFER is not empty, the transaction program has, prior to issuing the current MC_POST_ON_RECEIPT, issued one or more MC_POST_ON_RECEIPTs followed by one or more MC_TESTs. The MC_TEST processing caused PS.MC to receive data (via a RECEIVE_AND_WAIT) from PS.CONV (Chapter 5.1) and PS.MC has stored that data in the MC_RECEIVE_BUFFER. See "MC_TEST_PROC" on page 5.2-11 for a discussion of MC_TEST.

2. If the information stored in the MC_RECEIVE_BUFFER indicates that a complete Application Data or User Control Data GDS variable has been received (and that the data in that variable has been mapped), then PS.MC has already informed the transaction program via the RETURN_CODE on a previous MC_TEST that posting has been satisfied. The transaction program, however, has issued another MC_POST_ON_RECEIPT (after having issued an MC_TEST on which was returned a return code of OK--DATA). PS.MC remembers the fact that an MC_POST_ON_RECEIPT has been issued, in case the transaction program issues another MC_TEST, but does not issue a POST_ON_RECEIPT to PS.CONV.

3. If the data stored in the MC_RECEIVE_BUFFER is not complete (i.e., a Map Name GDS variable, but no data, has been received; or part, but not all, of the data in an Application or FMH Data GDS variable has been received), posting is still activated. PS.MC, therefore, does not issue a POST_ON_RECEIPT to PS.CONV. In this situation, the transaction program has issued one or more prior MC_TESTs, all of which have been unsuccessful.

4. PS.MC performs no check to determine if the conversation is in an appropriate state to receive an MC_POST_ON_RECEIPT. This state check is performed by PS.CONV (Chapter 5.1) during its processing of the POST_ON_RECEIPT verb, if PS.MC issues one. As described above, there are certain situations in which PS.MC receives an MC_POST_ON_RECEIPT from the transaction program but does not issue a POST_ON_RECEIPT to PS.CONV. In these situations, however, the MC_RECEIVE_BUFFER in the RCB is not empty. This indicates that the conversation is in RECEIVE state and therefore the MC_POST_ON_RECEIPT is valid at the present time.

Referenced procedures, FSMs, and data structures:

POST_ON_RECEIPT_PROC	page 5.1-17
RCB	page A-6

If the RCB.MC_RECEIVE_BUFFER for the current conversation is empty then
  Call POST_ON_RECEIPT_PROC(POST_ON_RECEIPT) (page 5.1-17) on this conversation,
   specifying the maximum length of the data to be received before posting,
   and that posting should be done after receiving a complete logical record.
Set the MC_POST_ON_RECEIPT parameters and return code to the values returned by the
 POST_ON_RECEIPT verb.

MC_PREPARE_TO_RECEIVE_PROC

FUNCTION:	This procedure processes MC_PREPARE_TO_RECEIVE verbs. PS.MC issues a PREPARE_TO_RECEIVE verb against the resource specified in the MC_PREPARE_TO_RECEIVE. It sets the return code field in the MC_PREPARE_TO_RECEIVE based upon the value returned in the PREPARE_TO_RECEIVE. Some return codes, such as OK, are placed in the MC_PREPARE_TO_RECEIVE unchanged. Others, such as DEALLOCATE_ABEND_PROG, are transformed to another value before being placed in the MC_PREPARE_TO_RECEIVE. In addition, some return codes cause PS.MC to perform further processing. For example, when PS.MC receives a return code of PROG_ERROR_PURGING to its PREPARE_TO_RECEIVE, it invokes the mapper to inform that procedure that an error was detected by the partner transaction program. (See "Mapper Invocation" on page 5.2-9.) When a return code of SVC_ERROR_PURGING is received, PS.MC performs the processing necessary to determine what type of service error the PS.MC component at the partner LU encountered. A return code reflecting the type of error is returned to the local transaction program in the MC_PREPARE_TO_RECEIVE. (See "Processing of a Service Error Detected by Partner LU" on page 5.2-17.)
INPUT:	MC_PREPARE_TO_RECEIVE verb parameters (See SNA Transaction Programmer's Reference Manual for LU Type 6.2.)
OUTPUT:	PS.MC issues a PREPARE_TO_RECEIVE verb and sets the return code field in the MC_PREPARE_TO_RECEIVE based upon the corresponding field in the PREPARE_TO_RECEIVE.
NOTE:	PS.MC performs no check to determine if the conversation is in an appropriate state to receive an MC_PREPARE_TO_RECEIVE. This state check is performed by PS.CONV (Chapter 5.1) during its processing of the PREPARE_TO_RECEIVE verb.

Referenced procedures, FSMs, and data structures:

UPM_MAPPER	page 5.2-46
PREPARE_TO_RECEIVE_PROC	page 5.1-18
RCVD_SVC_ERROR_PURGING	page 5.2-42
RCB	page A-6

```
Find the RCB for the conversation identified in the RESOURCE parameter.
Call PREPARE_TO_RECEIVE_PROC(PREPARE_TO_RECEIVE)(page 5.1-18) to issue a
 PREPARE_TO_RECEIVE verb with the MC_PREPARE_TO_RECEIVE verb parameters.
Set the MC_PREPARE_TO_RECEIVE parameters and return code to the values
 returned by the PREPARE_TO_RECEIVE verb.

Select based on the return code copied into MC_PREPARE_TO_RECEIVE:
   When (PROG_ERROR_PURGING)
      Call the UPM_MAPPER (page 5.2-46) to record
       the RETURN_CODE for the remotely detected error.
   When (DEALLOCATE_ABEND_PROG)
      Set RETURN_CODE to DEALLOCATE_ABEND.
   When (DEALLOCATE_ABEND_SVC, DEALLOCATE_ABEND_TIMER)
      Set RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
   When (SVC_ERROR_PURGING)
      Call RCVD_SVC_ERROR_PURGING (page 5.2-42) to
       do service error processing, specifying the return code and current RCB.
```

MC_RECEIVE_AND_WAIT_PROC

FUNCTION: This procedure processes MC_RECEIVE_AND_WAIT verbs.

PS.MC first determines the status of the MC_RECEIVE_BUFFER. Processing of the MC_RECEIVE_AND_WAIT continues based upon the status of the buffer.

The MC_RECEIVE_BUFFER contains any information that has been received from PS.CONV (Chapter 5.1) but has not yet been passed to the transaction program. It is in one of the following states: (1) the buffer is empty, (2) the buffer contains information, but the information is incomplete and more has to be received before it can be passed to the transaction program, or (3) the buffer contains information that is complete and ready to be passed to the transaction program.

If the MC_RECEIVE_BUFFER is not empty, the transaction program has issued one or more prior MC_TEST verbs. The processing that PS.MC performed as a result of the MC_TEST(s) involved receiving data from PS.CONV. It is the data that resulted from the MC_TEST(s) that is stored in the MC_RECEIVE_BUFFER.

INPUT: MC_RECEIVE_AND_WAIT verb parameters (See <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>.)

OUTPUT: Fields in the MC_RECEIVE_AND_WAIT are set based upon the type of information being returned to the transaction program.

If the MC_RECEIVE_BUFFER is empty or contains incomplete data, this procedure causes one or more RECEIVE_AND_WAIT verbs to be issued to PS.CONV. PS.MC continues to issue RECEIVE_AND_WAITs until it has a complete piece of information.

NOTES:

1. PS.MC performs no check to determine if the conversation is in an appropriate state to receive an MC_RECEIVE_AND_WAIT. This state check is performed by PS.CONV (Chapter 5.1) during its processing of the RECEIVE_AND_WAIT verb, if PS.MC issues one. If the MC_RECEIVE_BUFFER already contains complete information ready to be passed to the transaction program, PS.MC does not issue a RECEIVE_AND_WAIT. However, the fact that the MC_RECEIVE_BUFFER is not empty indicates that the mapped conversation is in RECEIVE state and that the MC_RECEIVE_AND_WAIT is valid at the present time.

2. RECEIVE_INFO_PROC on page 5.2-30 issues a RECEIVE_AND_WAIT to PS.CONV and causes processing of the information returned in the RECEIVE_AND_WAIT to occur. It is possible that when control is returned from this procedure, the MC_RECEIVE_BUFFER is empty, even though data was returned in the RECEIVE_AND_WAIT. This is the case when PS.MC detects an error in the data (e.g., the data specified a function not supported). Nothing is placed in the buffer during this invocation of RECEIVE_INFO_PROC. For more details, see "Service Errors Detected in Received Data" on page 5.2-14.

Referenced procedures, FSMs, and data structures:

RECEIVE_INFO_PROC	page 5.2-30
RCB	page A-6

```
If the RCB.MC_RECEIVE_BUFFER contains a null entry, map name, data-continued
 indicator, or map name and data-continued indicator then
   Call RECEIVE_INFO_PROC(RCB) (page 5.2-30)
    to issue a RECEIVE_AND_WAIT verb.
If the RCB.MC_RECEIVE_BUFFER does not contain a null entry, or contains
 mapped data or a return code entry then
   Select based on the contents of the RCB.MC_RECEIVE_BUFFER:
      When the buffer element contains a WHAT_RECEIVED indicator
         Put the WHAT_RECEIVED indicator in the MC_RECEIVE_AND_WAIT verb.
         Set RETURN_CODE to OK.
      When the buffer element contains a return code
         Set RETURN_CODE to the buffer return code.
      When the buffer element contains mapped data
         Retrieve the mapped data from the MC_RECEIVE_BUFFER and place the
          amount of data requested by the transaction program into the DATA
          field of the MC_RECEIVE_AND_WAIT.  Indicate whether data was complete
          or truncated, and indicate that FMH data, if present, was complete.
```

MC_RECEIVE_AND_WAIT_PROC

Clear the MC_RECEIVE_BUFFER for the current RCB.
If a request to send has been received from the remote TP and not returned on
 a prior MC_CONFIRM, MC_RECEIVE_AND_WAIT, MC_SEND_DATA, or MC_SEND_ERROR verb then
   Return a request-to-send-received indication to the local TP on the verb.

MC_TEST_PROC

FUNCTION: This procedure processes MC_TEST verbs.

INPUT: MC_TEST

OUTPUT: PS.MC sets the RETURN_CODE field in the MC_TEST based upon the outcome of the specified test. Depending upon the type of test specified and the information contained in the RCB, PS.MC may issue basic conversation verbs that are processed by PS.CONV. RCB.MC_RECEIVE_BUFFER, or a return code obtained by calling TEST_PROC (page 5.1-26).

NOTES:

1. If RCB.MC_RECEIVE_BUFFER is not empty when a return code of OK--NOT_DATA is received, the partner LU has committed a protocol violation. For example, the partner LU has sent data with an indication that the data is continued in the next logical record, but instead of sending the remaining data, the partner LU allowed a SEND indicator to flow.

2. RCB.MC_RECEIVE_BUFFER may be empty at this point. This occurs when the TEST verb just issued returns OK--DATA but an error is detected in the data by RECEIVE_INFO_PROC (page 5.2-30). For more details, see "Service Errors Detected in Received Data" on page 5.2-14.

3. An INDICATOR element cannot appear in RCB.MC_RECEIVE_BUFFER here. If the TEST verb just issued returns OK--NOT_DATA, the conversation indicator that caused this return code remains in PS.CONV's buffer. PS.MC does not issue a RECEIVE_AND_WAIT to PS.CONV to get the indicator until the transaction program issues an MC_RECEIVE_AND_WAIT.

4. The RCB.MC_RECEIVE_BUFFER contains data ready to be returned to the transaction program as a result of one or more prior calls to MC_TEST (TEST=POSTED).

Referenced procedures, FSMs, and data structures:

TEST_PROC	page 5.1-26
RECEIVE_INFO_PROC	page 5.2-30
POST_ON_RECEIPT_PROC	page 5.1-17
PROTOCOL_ERROR_PROC	page 5.2-47
PROCESS_ERROR_OR_FAILURE_RC	page 5.2-31
RCB	page A-6

```
Select based on the specified type of test:
   When POSTED
      If RCB.MC_RECEIVE_BUFFER is empty or contains a map name or unmapped data then
         Call TEST_PROC (page 5.1-26) to determine whether the current
          conversation has been posted indicating that data, status, or a
          request for confirmation has been received from the remote TP.
         Select based on the return code from TEST_PROC:
            When OK--DATA
               Call RECEIVE_INFO_PROC (page 5.2-30) to receive
                the data and place it in RCB.MC_RECEIVE_BUFFER.
            When OK--NOT_DATA
               If RCB.MC_RECEIVE_BUFFER is empty then
                  Put the RETURN_CODE from TEST in RCB.MC_RECEIVE_BUFFER.
               Else (optional check when receiving data; See Note 1)
                  Call PROTOCOL_ERROR_PROC (page 5.2-47)
                   to deallocate the current conversation.
                  Replace the contents of RCB.MC_RECEIVE_BUFFER with the
                   RETURN_CODE RESOURCE_FAILURE_NO_RETRY.
            When POSTING_NOT_ACTIVE or UNSUCCESSFUL
               Put the RETURN_CODE from TEST in RCB.MC_RECEIVE_BUFFER.
            Otherwise
               Call PROCESS_ERROR_OR_FAILURE_RC (page 5.2-31)
                to process the RETURN_CODE from TEST.
         If RCB.MC_RECEIVE_BUFFER is empty or contains a map name or
          unmapped data (See Note 2) then
            Set RETURN_CODE to UNSUCCESSFUL.
            Call POST_ON_RECEIPT_PROC(POST_ON_RECEIPT)(page 5.1-17) to issue a
             POST_ON_RECEIPT verb specifying posting when a complete or truncated logical
             record is received.
         Else
            Select based on the type of information in RCB.MC_RECEIVED_BUFFER
             (See Note 3):
               When it is mapped data
                  Set RETURN_CODE to OK--DATA.
               When it is a RETURN_CODE
                  Set RETURN_CODE to that in RCB.MC_RECEIVE_BUFFER.
                  Clear RCB.MC_RECEIVE_BUFFER.
      Else
         If there is mapped data in RCB.MC_RECEIVE_BUFFER and the local
          TP has issued a MC_POST_ON_RECEIPT verb since this data was
          mapped then (See Note 4)
            Set RETURN_CODE to OK--DATA.
         Else
            Set RETURN_CODE to POSTING_NOT_ACTIVE.

   When REQUEST_TO_SEND_RECEIVED
      If a request to send has been received from the remote TP and not
       yet returned to the local TP then
         Return a request-to-send-received indication to the local TP.
      Else
         Call TEST_PROC (page 5.1-26) to determine whether
          a request to send has been received from the remote TP and is
          being held by PS.CONV.
         If a request to send was held by PS.CONV then
            Return a request-to-send-received indication to the local TP.
```

RECEIVE_INFO_PROC

FUNCTION:	The purpose of this procedure is to receive information from PS.CONV (Chapter 5.1) and to place that information in the MC_RECEIVE_BUFFER. This procedure issues a RECEIVE_AND_WAIT for the mapped conversation corresponding to the passed RCB. PS.MC continues the processing of the RECEIVE_AND_WAIT in other procedures, depending upon the return code carried in the RECEIVE_AND_WAIT.
INPUT:	The RCB corresponding to the mapped conversation specified in the TRANSACTION_PGM_VERB currently being processed
OUTPUT:	See the procedures called for the specific outputs.

Referenced procedures, FSMs, and data structures:

PROCESS_ERROR_OR_FAILURE_RC	page 5.2-31
PROTOCOL_ERROR_PROC	page 5.2-47
PROCESS_DATA_COMPLETE	page 5.2-33
PROCESS_DATA_INCOMPLETE	page 5.2-36
UPM_MAPPER	page 5.2-46
RCB	page A-6

```
Issue a basic RECEIVE_AND_WAIT verb for a complete logical record
 specifying the maximum length of the data.
If a request to send data was received from the remote TP then
   Save an indication of the request to be returned later.
If the  RECEIVE_AND_WAIT was successful then
   Select based on the WHAT_RECEIVED field on the RECEIVE_AND_WAIT verb:
      When the data received is complete
         Call PROCESS_DATA_COMPLETE(RCB, RECEIVE_AND_WAIT) (page 5.2-33).
      When the data received is incomplete
         Call PROCESS_DATA_INCOMPLETE(RCB) (page 5.2-36).
      When the RCB.MC_RECEIVE_BUFFER is empty
         Put the WHAT_RECEIVED indicator in the MC_RECEIVE_BUFFER
           of the current RCB.
         Call the UPM_MAPPER (page 5.2-46) to save an
          indication that the end of the logical message was received.
      When the RCB.MC_RECEIVE_BUFFER is not empty, but does not contain data,
         Clear the MC_RECEIVE_BUFFER in the current RCB.
         Call PROTOCOL_ERROR_PROC (page 5.2-47)
          to deallocate the current conversation.
         Put the RESOURCE_FAILURE_NO_RETRY RETURN_CODE in the
          MC_RECEIVE_BUFFER of the current RCB.
Else
    Call PROCESS_ERROR_OR_FAILURE_RC (page 5.2-31)
```

PROCESS_ERROR_OR_FAILURE_RC

FUNCTION:	This procedure is invoked after PS.MC has issued a RECEIVE_AND_WAIT to which has been returned a RETURN_CODE value other than OK. Processing of the return code continues in other procedures, depending upon the return code.
INPUT:	The RCB corresponding to the conversation specified in the verb being processed, and the RECEIVE_AND_WAIT return code to be processed
OUTPUT:	A return code value is placed in RCB.MC_RECEIVE_BUFFER.
NOTES:	1. Certain return codes are invalid if RCB.MC_RECEIVE_BUFFER is not empty, and, if received at such a time, indicate that the partner LU has committed a protocol violation. Depending upon the return code, PS.MC may end the mapped conversation.
	2. A return code on RECEIVE_AND_WAIT of ALLOCATION_ERROR cannot occur if prior information has been received on the specified mapped conversation.
	3. A return code on RECEIVE_AND_WAIT of PROG_ERROR_PURGING or SVC_ERROR_PURGING cannot occur if MC_RECEIVE_BUFFER is not empty. It can occur only if the RECEIVE_AND_WAIT was issued by PS.MC while the mapped conversation was in SEND state. (The partner transaction program or LU that issued the *_ERROR_PURGING information was in RECEIVE state.) Since the mapped conversation was in the SEND state locally, no information can be in RCB.MC_RECEIVE_BUFFER.
	4. The return codes that reference this note can be received at any time and are valid regardless of the status of RCB.MC_RECEIVE_BUFFER.
	5. A return code of *_ERROR_TRUNC cannot be received on the RECEIVE_AND_WAIT issued by this procedure because it can only be received following a RECEIVE_AND_WAIT in which a WHAT_RECEIVED value of DATA_INCOMPLETE is returned. (This procedure is not invoked after a DATA_INCOMPLETE indicator has been received.)

Referenced procedures, FSMs, and data structures:

PS_SPS	page 5.3-35
RCVD_SVC_ERROR_TRUNC_NO_TRUNC	page 5.2-41
RCVD_SVC_ERROR_PURGING	page 5.2-42
UPM_MAPPER	page 5.2-46
PROTOCOL_ERROR_PROC	page 5.2-47
RCB	page A-6

```
Select based on the RECEIVE_AND_WAIT return code being processed:
   When ALLOCATION_ERROR (See Note 2), PROGRAM_STATE_CHECK
      Put the RETURN_CODE in RCB.MC_RECEIVE_BUFFER.
   When DEALLOCATE_NORMAL
      If RCB.MC_RECEIVE_BUFFER is empty then
         Put the RETURN_CODE in RCB.MC_RECEIVE_BUFFER.
      Else (optional check when receiving data; See Note 1)
         Replace the contents of RCB.MC_RECEIVE_BUFFER by the
          RETURN_CODE value RESOURCE_FAILURE_NO_RETRY.
         Optionally log implementation-dependent error data.
   When DEALLOCATE_ABEND_PROG
      If RCB.MC_RECEIVE_BUFFER is empty then
         Put the RETURN_CODE DEALLOCATE_ABEND in RCB.MC_RECEIVE_BUFFER.
      Else (optional check when receiving data; See Note 1 )
         Replace the contents of RCB.MC_RECEIVE_BUFFER by the
          RETURN_CODE RESOURCE_FAILURE_NO_RETRY.
         Optionally log implementation-dependent error data.
   When PROG_ERROR_PURGING (See Note 3)
      Put the RETURN_CODE in RCB.MC_RECEIVE_BUFFER.
      Call UPM_MAPPER (page 5.2-46) to record a remotely detected
       error of the type indicated by the return code parameter.
```

```
When PROG_ERROR_NO_TRUNC
   If RCB.MC_RECEIVE_BUFFER is empty then
      Put the RETURN_CODE in RCB.MC_RECEIVE_BUFFER.
      Call UPM_MAPPER (page 5.2-46) to record a remotely detected
       error of the type indicated by the RETURN_CODE.
   Else (optional check when receiving data; See Note 1)
      Call PROTOCOL_ERROR_PROC (page 5.2-47)
       to deallocate the current conversation.
      Replace the contents of RCB.MC_RECEIVE_BUFFER by the
       RETURN_CODE RESOURCE_FAILURE_NO_RETRY.
When DEALLOCATE_ABEND_SVC, DEALLOCATE_ABEND_TIMER (See Note 4)
   Replace the contents of RCB.MC_RECEIVE_BUFFER by the
    RETURN_CODE RESOURCE_FAILURE_NO_RETRY.
When RESOURCE_FAILURE_RETRY, RESOURCE_FAILURE_NO_RETRY (See Note 4)
   Replace the contents of RCB.MC_RECEIVE_BUFFER by the RETURN_CODE.
When BACKED_OUT
   If RCB.MC_RECEIVE_BUFFER is empty then
      Call PS_SPS (sync point manager, Chapter 5.3).
      Put the RETURN_CODE in RCB.MC_RECEIVE_BUFFER.
   Else (optional check when receiving data; See Note 1)
      Call PROTOCOL_ERROR_PROC (page 5.2-47)
       to deallocate the current conversation.
      Replace the contents of RCB.MC_RECEIVE_BUFFER by the
       RETURN_CODE RESOURCE_FAILURE_NO_RETRY.
When SVC_ERROR_NO_TRUNC (See Note 4)
   Clear the RCB.MC_RECEIVE_BUFFER.
   Call RCVD_SVC_ERROR_TRUNC_NO_TRUNC (page 5.2-41)
    to process the RETURN_CODE.
When SVC_ERROR_PURGING (See Note 3)
   Call RCVD_SVC_ERROR_PURGING (page 5.2-42) to get
    and process error data from the partner LU.
   Put the RETURN_CODE in RCB.MC_RECEIVE_BUFFER.
```

PROCESS_DATA_COMPLETE

FUNCTION:	This procedure is invoked when PS.MC issues a RECEIVE_AND_WAIT and a value of DATA_COMPLETE is returned in the WHAT_RECEIVED field of the RECEIVE_AND_WAIT. The purpose of this procedure is to process the received data. The data received in the RECEIVE_AND_WAIT is a logical record. It may be the first or only logical record in a GDS variable. Alternatively, it may be a subsequent logical record in a GDS variable containing multiple logical records. A subsequent logical record does not carry a GDS ID field. If the MC_RECEIVE_BUFFER is empty, the data in the RECEIVE_AND_WAIT is the initial or only logical record in a GDS variable. This procedure checks the GDS ID in the logical record and calls the appropriate procedure to process the data carried in the DATA field of the logical record. If the MC_RECEIVE_BUFFER contains a map name but no data, the data in the RECEIVE_AND_WAIT is again the initial or only logical record in a GDS variable. The GDS variable following a Map Name GDS variable has to contain application or user control data. If the MC_RECEIVE_BUFFER contains incomplete data or a map name and incomplete data (i.e., the last logical record in a GDS variable that contains multiple logical records has not been received), the appropriate procedure is called to add the data carried in the DATA field of the subsequent logical record to the data already contained in the MC_RECEIVE_BUFFER. If the subsequent logical record is the last logical record in the GDS variable, additional processing is performed.
INPUT:	The RCB associated with the mapped conversation specified in the current verb issued by the transaction program and the RECEIVE_AND_WAIT (issued by PS.MC) that contains the data to be processed
OUTPUT:	Depending upon the data received, the MC_RECEIVE_BUFFER may be updated. See the procedures called for specific outputs.

Referenced procedures, FSMs, and data structures:

SEND_SVC_ERROR_PURGING	page 5.2-45
PROTOCOL_ERROR_PROC	page 5.2-47
PROCESS_MAPPER_RETURN_CODE	page 5.2-35
UPM_MAPPER	page 5.2-46
RCB	page A-6

```
If the MC_RECEIVE_BUFFER for the current conversation is empty (no map name) then
    Select based on the type of GDS variable in the passed data (first record):
       When a Map Name GDS variable
          If the LU receiving the map name supports mapping and the TP for this
           conversation supports mapping then
             Put the unmapped map name in the MC_RECEIVE_BUFFER (data incomplete).
          Else (the LU or TP doesn't support mapping)
             Call SEND_SVC_ERROR_PURGING (page 5.2-45) to
              handle the invalid map name and mapping request.
       When an Application Data GDS variable
          Put the passed unmapped data and an indication that FM headers are
           not included in the data in the MC_RECEIVE_BUFFER.
          If data is not continued in the next logical record (only one record) then
             Call the UPM_MAPPER(RCB.MAPPER_SAVE_AREA) (page 5.2-46)
              to map the received data, specifying that FMH data is not included.
              (No mapping will occur if no map name is found.)
             Call PROCESS_MAPPER_RETURN_CODE (page 5.2-35).
```

```
            When a User Control Data GDS variable
               If the LU for the current conversation supports FMH data and
                the TP for the current conversation supports FMH data then
                  Put the passed unmapped data and an indication that FM headers are
                   included in the data in the MC_RECEIVE_BUFFER.
                  If the data is not continued in the next record (one logical record) then
                     Call the UPM_MAPPER(RCB.MAPPER_SAVE_AREA) (page 5.2-46)
                      to get the map name and to map the received data, specifying that FMH
                      data is included.  (No mapping will occur if no map name is found.)
                     Call PROCESS_MAPPER_RETURN_CODE(RCB) (page 5.2-35).
               Else (the LU or TP doesn't Support FMH-data)
                  Call SEND_SVC_ERROR_PURGING  (page 5.2-45)
                   to perform service error purging, and to notify the partner LU.
            When a Null Structured Data GDS variable
               Do nothing.
            When an Error Data GDS variable, optionally
               Call PROTOCOL_ERROR_PROC (page 5.2-47)
                to deallocate the current conversation.
               Put the return code in the MC_RECEIVE_BUFFER of the current RCB.
            When the GDS ID is invalid
               Call SEND_SVC_ERROR_PURGING (page 5.2-45) to
                handle the invalid GDS ID (no such variable type).
      Else (the MC_RECEIVE_BUFFER is not empty)
         If the buffer element in the MC_RECEIVE_BUFFER is a map name then
            Select based on the contents of the passed RECEIVE_AND_WAIT data:
               When the GDS ID indicates an Application Data variable
                  Add the passed data and an indication that FM headers are
                   not included in the data to the unmapped map name in the
                   MC_RECEIVE_BUFFER.
                  If the data is not continued in the next record (one record) then
                     Call the UPM_MAPPER(RCB.MAPPER_SAVE_AREA) (page 5.2-46)
                      to map the received data in the MC_RECEIVE_BUFFER.
                     Call PROCESS_MAPPER_RETURN_CODE (page 5.2-35).
               When the GDS ID indicates a User Control Data GDS variable
                  If the LU for the current conversation supports FMH data and
                   the TP for the current conversation supports FMH data then
                     Add the passed data and an indication that FM headers are included
                      in the data to the unmapped map name in the MC_RECEIVE_BUFFER.
                     If the data is not continued in the next record (only one record) then
                        Call the UPM_MAPPER(RCB.MAPPER_SAVE_AREA) (page 5.2-46)
                         to map the received data in the MC_RECEIVE_BUFFER.
                        Call PROCESS_MAPPER_RETURN_CODE (page 5.2-35).
                  Else (the LU or TP doesn't support FMH data)
                     Call SEND_SVC_ERROR_PURGING  (page 5.2-45)
                      to perform service error purging, and to notify the partner LU.
               When the GDS ID is invalid for a map name buffer element, optionally
                  Purge the MC_RECEIVE_BUFFER for the current RCB.
                  CALL PROTOCOL_ERROR_PROC (page 5.2-47) to
                   deallocate the conversation.
                  Put the return code in the MC_RECEIVE_BUFFER of the current RCB.
         Else (the buffer element indicates continued data, with or without a map name)
            Add the passed data to the data contained in the MC_RECEIVE_BUFFER.
            If the data is not continued in the next logical record then
               Call the UPM_MAPPER (page 5.2-46) to map the contents
                of the MC_RECEIVE_BUFFER (a complete variable), specifying the map
                name, if any, and that FM header data is included.
```

PROCESS_MAPPER_RETURN_CODE

FUNCTION: This procedure determines whether the mapper was successful in mapping data. It is invoked after the mapper has been called to process data received from the partner transaction program.

INPUT: The RCB corresponding to the mapped conversation over which the data to be mapped flowed; and a structure containing information that is both supplied to, and returned from, the mapper

OUTPUT: If the mapper was able to successfully map the received data, the mapped data, along with a locally known map name provided by the mapper and an indication of the format of the mapped data, is placed in the MC_RECEIVE_BUFFER. If mapping was unsuccessful, PS.MC performs service error purging processing to notify the partner LU that the received data could not be mapped. (See "Service Errors Detected in Received Data" on page 5.2-14.)

NOTES:

1. If the mapper was successful in mapping the received data, it always provides to PS.MC a protocol boundary map name known to the local transaction program. The map name is supplied by the mapper even when it was invoked without a map name (in which case, the mapper uses a previously received map name). If mapping is off, the mapper supplies a null map name, which is passed to the transaction program.

2. If the mapper encountered an error in mapping the data, it provides to PS.MC the map name, known to the remote LU, that was in effect when the mapper was invoked. PS.MC places the map name in an Error Data GDS variable, which is sent to the partner LU to notify it of the mapping failure.

3. A return code of MAP_NOT_FOUND cannot be returned from the mapper if the mapper is invoked without a map name. If the mapper is invoked without a map name, it determines that it is to use a previously received map name. If the map name had been unknown to the mapper, this fact would have been discovered as a result of the earlier mapper invocation.

Referenced procedures, FSMs, and data structures:

SEND_SVC_ERROR_PURGING	page 5.2-45
PROTOCOL_ERROR_PROC	page 5.2-47
RCB	page A-6

```
Select based on the RETURN_CODE from the mapper:
   When mapping was successful
      Put the mapped map name, an indication that FM headers are included
       in the data, and the mapped data in the MC_RECEIVE_BUFFER.
   When mapping failed to execute successfully
      Call SEND_SVC_ERROR_PURGING (page 5.2-45)
       specifying the current RCB and the error type.
   When the provided map name was not found
      Call SEND_SVC_ERROR_PURGING (page 5.2-45)
       specifying the current RCB and the error type.
   When the map name was a duplicate (optional processing for receive only)
      Call PROTOCOL_ERROR_PROC (page 5.2-47) to
       deallocate the current conversation.
      Put a duplicate map name RETURN_CODE in the current MC_RECEIVE_BUFFER.
```

PROCESS_DATA_INCOMPLETE

FUNCTION:	This procedure is invoked when PS.MC issues a RECEIVE_AND_WAIT as a result of a mapped conversation verb issued by the transaction program. PS.MC has examined the value returned in the WHAT_RECEIVED field of the RECEIVE_AND_WAIT, determined that the value received is DATA_INCOMPLETE, and has discarded the incomplete logical record returned in the RECEIVE_AND_WAIT. This procedure purges the MC_RECEIVE_BUFFER of any data that has been received via one or more prior RECEIVE_AND_WAITs. It then issues a RECEIVE_AND_WAIT to determine the reason for the logical record being truncated. Processing continues based upon the RETURN_CODE value received in the RECEIVE_AND_WAIT.
INPUT:	The RCB corresponding to the resource specified in the RECEIVE_AND_WAIT in which DATA_INCOMPLETE was returned.
OUTPUT:	This procedure issues a RECEIVE_AND_WAIT. Depending upon the RETURN_CODE value returned on the RECEIVE_AND_WAIT, a return code buffer element may be inserted into the MC_RECEIVE_BUFFER.
NOTE:	RETURN_CODE values of DEALLOCATE_ABEND_PROG, PROG_ERROR_TRUNC, and BACKED_OUT following a DATA_INCOMPLETE notification indicate that the partner LU has committed a protocol violation by allowing the transaction program to truncate data. This should never occur at the mapped conversation protocol boundary. The PS.MC at the partner LU is allowed to truncate a logical record with SVC_ERROR_TRUNC, for instance; the transaction program is not.

Referenced procedures, FSMs, and data structures:

RCVD_SVC_ERROR_TRUNC_NO_TRUNC	page 5.2-41
PROTOCOL_ERROR_PROC	page 5.2-47
PS_VERB_ROUTER	page 5.0-16
RCB	page A-6

```
Clear the RCB.MC_RECEIVE_BUFFER.
Call the PS_VERB_ROUTER (Chapter 5.0) to issue a
 RECEIVE_AND_WAIT verb to get the RETURN_CODE that explains why the
 data was incomplete.
If a request to send data was received from the remote TP then
   Save an indication of the request to be returned later.
Select based on the RECEIVE_AND_WAIT return code:
   When the return code is SVC_ERROR_TRUNC
      Call RCVD_SVC_ERROR_TRUNC_NO_TRUNC to do service error processing
       (page 5.2-41).
   When the return code is DEALLOCATE_ABEND_SVC or DEALLOCATE_ABEND_TIMER
      Put the RETURN_CODE RESOURCE_FAILURE_NO_RETRY in the
       MC_RECEIVE_BUFFER of the current RCB.
   When the RETURN_CODE is RESOURCE_FAILURE_RETRY or RESOURCE_FAILURE_NO_RETRY
      Put the RETURN_CODE in the MC_RECEIVE_BUFFER of the current RCB.
   When the RETURN_CODE is DEALLOCATE_ABEND_PROG, optionally do the following:
      Put the RETURN_CODE RESOURCE_FAILURE_NO_RETRY in the
       MC_RECEIVE_BUFFER of the current RCB.
      Log implementation-dependent error data in the system error log.
   When the RETURN_CODE is PROG_ERROR_TRUNC or BACKED_OUT, optionally do the following:
      Call PROTOCOL_ERROR_PROC (page 5.2-47) to deallocate the
       current conversation.
      Put the RETURN_CODE in the MC_RECEIVE_BUFFER of the current RCB.
```

MC_REQUEST_TO_SEND_PROC

FUNCTION:	This procedure processes MC_REQUEST_TO_SEND verbs. PS.MC issues a REQUEST_TO_SEND verb against the resource specified in the MC_REQUEST_TO_SEND and returns control to the transaction program.
INPUT:	MC_REQUEST_TO_SEND verb parameters.
NOTE:	PS.MC performs no check to determine if the conversation is in an appropriate state to receive an MC_REQUEST_TO_SEND verb. A state check is performed by PS.CONV (Chapter 5.1) during its processing of the REQUEST_TO_SEND verb.

Referenced procedures, FSMs, and data structures:
REQUEST_TO_SEND_PROC page 5.1-23

Call REQUEST_TO_SEND_PROC(REQUEST_TO_SEND)(page 5.1-23) to issue
a REQUEST_TO_SEND verb with the MC_REQUEST_TO_SEND verb parameters.
Set the MC_REQUEST_TO_SEND return code to the value returned by the REQUEST_TO_SEND verb.

MC_SEND_DATA_PROC

FUNCTION:		This procedure processes MC_SEND_DATA verbs. This procedure causes the mapper to be invoked. If the mapper is successful in mapping the data contained in the MC_SEND_DATA, or if the mapper determines that mapping is not being performed, the output data from the mapper is placed in an Application Data or User Control Data GDS variable (the variable may contain one or more logical records). The mapper may also return to PS.MC a map name that is to be sent to the partner LU, in which case PS.MC also creates a Map Name GDS variable that precedes the data GDS variable. This procedure then issues a SEND_DATA containing the GDS variable(s). PS.MC sets the return code field in the MC_SEND_DATA based upon the value returned in the SEND_DATA. Some return codes, such as OK, are placed in the MC_SEND_DATA unchanged. Others, such as DEALLOCATE_ABEND_PROG, are transformed to another value before being placed in the MC_SEND_DATA. In addition, some return codes cause PS.MC to perform further processing. For example, when PS.MC receives a return code of PROG_ERROR_PURGING to its SEND_DATA, it invokes the mapper to inform that procedure that the partner transaction program detected an error. (See "Mapper Invocation" on page 5.2-9.) When a return code of SVC_ERROR_PURGING is received, PS.MC performs the processing necessary to determine what type of service error the PS.MC component at the partner LU encountered. A return code reflecting the type of error is returned to the local transaction program in the MC_SEND_DATA. (See "Processing of a Service Error Detected by Partner LU" on page 5.2-17.)
INPUT:		MC_SEND_DATA verb parameters (See SNA Transaction Programmer's Reference Manual for LU Type 6.2.)
OUTPUT:		PS.MC issues a SEND_DATA verb. It sets fields in the MC_SEND_DATA based upon the corresponding values returned in the SEND_DATA.
NOTES:	1.	PS.MC performs a check to determine if the conversation is in an appropriate state to receive an MC_SEND_DATA. This is unlike its processing of most mapped conversation verbs, in that PS.MC generally does not perform this state check, but instead allows it to be performed by PS.CONV (Chapter 5.1). PS.MC performs the state check, rather than deferring it, for the following reasons: unlike other verbs, the MC_SEND_DATA causes PS.MC to perform some amount of processing before issuing a basic conversation verb. By PS.MC performing the state check, any state errors are detected before the processing is performed. In addition, if the data provided in the MC_SEND_DATA could not be mapped by the mapper procedure, no basic conversation verb is issued; in order to catch any state errors, PS.MC has to perform the state check.
	2.	The processing that PS.MC performs as a result of receiving a return code of SVC_ERROR_PURGING involves issuing one or more RECEIVE_AND_WAIT verbs. REQUEST_TO_SEND_RECEIVED information may be returned on the RECEIVE_AND_WAIT(s), and, if this is the case, the MC_SEND_DATA is updated to reflect this information.

Referenced procedures, FSMs, and data structures:

UPM_MAPPER	page 5.2-46
RCVD_SVC_ERROR_PURGING	page 5.2-42
PS_SPS	page 5.3-35
SEND_DATA_PROC	page 5.1-24
SEND_BUFFER	page 5.2-48
RCB	page A-6

```
Find the RCB for the resource specified in the MC_SEND_DATA verb.
If the resource is in a state to receive data (Chapter 5.1) then
   Call the UPM_MAPPER(RCB.MAPPER_SAVE_AREA) (page 5.2-46)
    to map the data to be sent, specifying the map name and whether or not the data
    contains FM header data (all from the verb).
   Select based on the RETURN_CODE from the mapper:
     When the mapper RETURN_CODE is MAP_NOT_FOUND
        Set the MC_SEND_DATA RETURN_CODE to MAP_NOT_FOUND.
     When the mapper RETURN_CODE is MAP_EXECUTION_FAILURE
        Set the MC_SEND_DATA RETURN_CODE to MAP_EXECUTION_FAILURE.
        Optionally, log implementation-dependent error data in system error log.
     When the mapping was successful
        If a map name was returned from the mapper then
           Create a Map Name GDS variable for the map name and put it in the SEND_BUFFER.
        Create a GDS variable that contains the data passed with the verb,
         which has been successfully mapped.  The GDS variable, depending on the
         amount of data, may consist of one logical record or of multiple continued
         logical records.  Only the first logical record will carry the GDS ID
         indicating either a User Control Data or an Application Data GDS variable type.
        Put, or add, the data GDS variable in, or to, the SEND_BUFFER.
        Call SEND_DATA_PROC (page 5.1-24) to issue a
         SEND_DATA verb with the MC_SEND_DATA verb parameters.
        Set the MC_SEND_DATA parameters and return code to the values returned
         by the SEND_DATA verb.

        Select based on the return code copied into MC_SEND_DATA:
           When OK
              Do nothing.
           When DEALLOCATE_ABEND_PROG
              Set RETURN_CODE to DEALLOCATE_ABEND.
           When DEALLOCATE_ABEND_SVC or DEALLOCATE_ABEND_TIMER
              Set RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
           When PROG_ERROR_PURGING
              Call UPM_MAPPER(RCB.MAPPER_SAVE_AREA) (page 5.2-46)
               to notify the mapper of the remotely detected error.
           When BACKED_OUT
              Call PS_SPS (Chapter 5.3).
           When SVC_ERROR_PURGING
              Call RCVD_SVC_ERROR_PURGING passing the current RCB and the
               SEND_DATA return code (page 5.2-42).
        If a request to send has been received from the remote TP and not
         returned on a prior MC_CONFIRM, MC_RECEIVE_AND_WAIT, MC_SEND_DATA,
         or MC_SEND_ERROR verb then
           Return a request-to-send-received indication to the local TP on
            the MC_SEND_DATA verb.
```

MC_SEND_ERROR_PROC

FUNCTION:		This procedure processes MC_SEND_ERROR verbs.
INPUT:		MC_SEND_ERROR verb parameters (See SNA Transaction Programmer's Reference Manual for LU Type 6.2.)
OUTPUT:		A return code indicating the result of the verb execution. An indication that a request to send has been received from the remote TP may also be returned.
NOTES:	1.	PS.MC performs no check to determine if the conversation is in an appropriate state to receive an MC_SEND_ERROR. A state check is performed by PS.CONV (Chapter 5.1) during its processing of the SEND_ERROR verb.
	2.	The processing that PS.MC performs as a result of receiving a return code of SVC_ERROR_PURGING involves issuing one or more RECEIVE_AND_WAIT verbs. A request to send from the remote TP may be returned on a RECEIVE_AND_WAIT and, if this is the case, an indication of the request is passed to the local TP.

Referenced procedures, FSMs, and data structures:

UPM_MAPPER	page 5.2-46
RCVD_SVC_ERROR_PURGING	page 5.2-42
PS_SPS	page 5.3-35
SEND_ERROR_PROC	page 5.1-25
RCB	page A-6


```
Find the RCB for the conversation identified in the RESOURCE parameter.
Clear RCB.MC_RECEIVE_BUFFER.
Call SEND_ERROR_PROC(SEND_ERROR)(page 5.1-25) to issue a SEND_ERROR
 verb with the MC_SEND_ERROR verb parameters.

Select based on the return code in SEND_ERROR:
   When OK
      Set RETURN_CODE to the value returned on SEND_ERROR.
      If the conversation is in send state (Chapter 5.1) then
         Call UPM_MAPPER (page 5.2-46) to record a locally detected
          error of the type PROG_ERROR_NO_TRUNC.
      Else
         Call UPM_MAPPER (page 5.2-46) to record a locally detected
          error of the type PROG_ERROR_PURGING.
   When PROG_ERROR_PURGING
      Set RETURN_CODE to the value returned on SEND_ERROR.
      Call UPM_MAPPER (page 5.2-46) to record a remotely detected
       error of the type indicated by the return code from SEND_ERROR.
   When ALLOCATION_ERROR, DEALLOCATE_NORMAL, PROGRAM_STATE_CHECK,
    RESOURCE_FAILURE_RETRY, or RESOURCE_FAILURE_NO_RETRY
      Set RETURN_CODE to the value returned on SEND_ERROR.
   When DEALLOCATE_ABEND_PROG
      Set RETURN_CODE to DEALLOCATE_ABEND.
   When DEALLOCATE_ABEND_SVC or DEALLOCATE_ABEND_TIMER
      Set RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
   When BACKED_OUT
      Call PS_SPS (Chapter 5.3).
      Set RETURN_CODE to the value returned on SEND_ERROR.
   When SVC_ERROR_PURGING
      Call RCVD_SVC_ERROR_PURGING (page 5.2-42).
      Set RETURN_CODE to the value returned on RCVD_SVC_ERROR_PURGING.

If a request to send has been received from the remote TP and not
 indicated to the local TP on a prior MC_CONFIRM, MC_RECEIVE_AND_WAIT,
 MC_SEND_DATA, or MC_SEND_ERROR verb then
   Return a request-to-send-received indication to the local TP
    (see SNA Transaction Programmer's Reference Manual for LU Type 6.2).
```

RCVD_SVC_ERROR_TRUNC_NO_TRUNC

FUNCTION: This procedure is invoked when a return code of SVC_ERROR_TRUNC or SVC_ERROR_NO_TRUNC is returned by a RECEIVE_AND_WAIT verb. This return code indicates that the partner LU detected a map execution failure while sending data. All or only part of the data may have been sent. Any data that was received prior to the error is purged. Error information is optionally placed in the system error log, but the local transaction program is not informed of the error.

INPUT: The RCB associated with the mapped conversation on which the service error was detected and the SVC_ERROR_TRUNC or SVC_ERROR_NO_TRUNC return code

NOTES:

1. If the expected Error Data GDS variable is not received, or is received but indicates an error condition that is invalid in the present situation, the partner LU has committed a protocol violation. If the protocol violation occurred as a result of the partner LU allowing the mapped conversation to be prematurely ended without having sent the error data, PS.MC simply logs the error. Otherwise, PS.MC ends the mapped conversation. In either case, PS.MC inserts a return code of RESOURCE_FAILURE_NO_RETRY in RCB.MC_RECEIVE_BUFFER.

2. A return code of RESOURCE_FAILURE_RETRY or _NO_RETRY can occur at any time and does not indicate that the partner LU committed a protocol violation.

Referenced procedures, FSMs, and data structures:

UPM_MAPPER	page 5.2-46
RECEIVE_AND_WAIT_PROC	page 5.1-19
PROTOCOL_ERROR_PROC	page 5.2-47
RCB	page A-6
ERROR_DATA_STRUCTURE	page 5.2-48

```
Call UPM_MAPPER (page 5.2-46) to record a remotely
 detected error of the type SVC_ERROR_TRUNC or SVC_ERROR_NO_TRUNC
 as indicated by the input parameter.
Call RECEIVE_AND_WAIT_PROC(RECEIVE_AND_WAIT)(page 5.1-19) to issue a
 RECEIVE_AND_WAIT verb with the MC_RECEIVE_AND_WAIT verb parameters.
Select based on the RETURN_CODE from RECEIVE_AND_WAIT:
   When OK
      Interpret the data returned by the RECEIVE_AND_WAIT verb as
       an ERROR_DATA_STRUCTURE.
      If RECEIVE_AND_WAIT returns DATA_COMPLETE, the GDS_ID in
       ERROR_DATA_STRUCTURE indicates that the structure contains
       error data (see SNA Formats), and ERROR_DATA_STRUCTURE.ERROR_CODE
       indicates a map execution failure (see SNA Formats) then
         Optionally log implementation-dependent error data.
      Else (optional check when receiving data; See Note 1 )
         Call PROTOCOL_ERROR_PROC (page 5.2-47)
          to deallocate the current conversation.
         Put the RETURN_CODE RESOURCE_FAILURE_NO_RETRY in the
          MC_RECEIVE_BUFFER of the current RCB.
   When RESOURCE_FAILURE_RETRY or RESOURCE_FAILURE_NO_RETRY (See Note 2)
      Put the RETURN_CODE from the RECEIVE_AND_WAIT verb in the
       MC_RECEIVE_BUFFER of the current RCB.
   When PROG_ERROR_NO_TRUNC, SVC_ERROR_NO_TRUNC, or BACKED_OUT
    (optional check when receiving data; See Note 1)
      Call PROTOCOL_ERROR_PROC (page 5.2-47)
       to deallocate the current conversation.
      Put the RETURN_CODE RESOURCE_FAILURE_NO_RETRY in the
       MC_RECEIVE_BUFFER of the current RCB.
   When DEALLOCATE_NORMAL, DEALLOCATE_ABEND_PROG, DEALLOCATE_ABEND_SVC, or
    DEALLOCATE_ABEND_TIMER (optional check when receiving data; See Note 1)
      Put the RETURN_CODE RESOURCE_FAILURE_NO_RETRY in the
       MC_RECEIVE_BUFFER of the current RCB.
      Optionally log implementation-dependent error data.
```

RCVD_SVC_ERROR_PURGING

FUNCTION: This procedure is invoked when PS.MC issues a basic conversation verb in which a return code of SVC_ERROR_PURGING is returned. Unlike SVC_ERROR_TRUNC and SVC_ERROR_NO_TRUNC, the SVC_ERROR_PURGING return code can be returned on a verb issued while the mapped conversation is in either send or receive state.

INPUT: The RCB corresponding to the specified conversation.

OUTPUT: A return code reflecting the outcome of the service error processing.

NOTES:

1. If the expected Error Data GDS variable is not received, the partner LU has committed a protocol violation. The checks for these violations given below are optional. If the protocol violation occurred as a result of the partner LU allowing the mapped conversation to be prematurely ended without having sent the error data, PS.MC simply logs the error. Otherwise, PS.MC ends the mapped conversation. In either case, PS.MC returns the code RESOURCE_FAILURE_NO_RETRY.

2. A return code of RESOURCE_FAILURE_RETRY or RESOURCE_FAILURE_NO_RETRY can occur at any time and does not indicate that the partner LU committed a protocol violation.

Referenced procedures, FSMs, and data structures:

UPM_MAPPER	page 5.2-46
RECEIVE_AND_WAIT_PROC	page 5.1-19
PROCESS_ERROR_DATA	page 5.2-43
GET_SEND_INDICATOR	page 5.2-44
PROTOCOL_ERROR_PROC	page 5.2-47
RCB	page A-6
ERROR_DATA_STRUCTURE	page 5.2-48

```
Call UPM_MAPPER (page 5.2-46) to record a remotely
 detected error of the type SVC_ERROR_PURGING as indicated by the
 RETURN_CODE from the last verb issued.
Call RECEIVE_AND_WAIT_PROC(RECEIVE_AND_WAIT)(page 5.1-19) to issue a
 RECEIVE_AND_WAIT verb with the MC_RECEIVE_AND_WAIT verb parameters.

Select based on the return code in RECEIVE_AND_WAIT:
   When OK
      Interpret the data returned by the RECEIVE_AND_WAIT verb as
       an ERROR_DATA_STRUCTURE.
      If RECEIVE_AND_WAIT returns DATA_COMPLETE and the GDS_ID of
       ERROR_DATA_STRUCTURE indicates that the structure contains
       error data then
         Call PROCESS_ERROR_DATA (page 5.2-43) and
          pass it the ERROR_DATA_STRUCTURE.
         Set RETURN_CODE to the value returned on PROCESS_ERROR_DATA.
         If the RETURN_CODE is not RESOURCE_FAILURE_NO_RETRY then
            Call GET_SEND_INDICATOR (page 5.2-44).
      Else (See Note 1)
         Call PROTOCOL_ERROR_PROC (page 5.2-47)
          to deallocate the current conversation.
         Set RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
   When RESOURCE_FAILURE_RETRY or RESOURCE_FAILURE_NO_RETRY (See Note 2)
      Set RETURN_CODE to the value returned on RECEIVE_AND_WAIT.
   When PROG_ERROR_NO_TRUNC, SVC_ERROR_NO_TRUNC, or BACKED_OUT (See Note 1)
         Call PROTOCOL_ERROR_PROC (page 5.2-47)
          to deallocate the current conversation.
         Set RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
   When DEALLOCATE_NORMAL, DEALLOCATE_ABEND_PROG,
    DEALLOCATE_ABEND_SVC or DEALLOCATE_ABEND_TIMER (See Note 1)
      Optionally log implementation-dependent error data.
      Set RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
```

PROCESS_ERROR_DATA

FUNCTION:	This procedure is invoked during the processing that PS.MC performs as a result of receiving a return code of SVC_ERROR_PURGING. It is called after receiving the Error Data GDS variable that follows the service error notification. The purpose of this procedure is to process the information carried in the Error Data GDS variable.
INPUT:	The Error Data GDS variable received from the remote TP
OUTPUT:	If the Error Data GDS variable contains no invalid values, this procedure returns a code that reflects the information carried in the error data and logs the error information in the system error log. If the error data contains an invalid value, PS.MC ends the mapped conversation.
NOTE:	When the Error Data GDS variable indicates MAP_NOT_FOUND or MAP_EXECUTION_FAILURE, the map name that caused the error is carried in the ERROR_PARM field of the Error Data GDS variable. When the Error Data GDS variable indicates INVALID_GDS_ID, the GDS_ID that specifies a function not supported by the partner LU or transaction program is carried in the ERROR_PARM field.

Referenced procedures, FSMs, and data structures:

PROTOCOL_ERROR_PROC	page 5.2-47
ERROR_DATA_STRUCTURE	page 5.2-48

```
Select based on ERROR_DATA_STRUCTURE.ERROR_CODE:
   When it indicates an invalid GDS_ID (see SNA Formats)
      Select based on the GDS_ID in ERROR_DATA_STRUCTURE.ERROR_PARM:
         When it indicates user control data (see SNA Formats)
            Set RETURN_CODE to FMH_DATA_NOT_SUPPORTED.
            Optionally log implementation-dependent error data.
         When it indicates map name (see SNA Formats)
            Set RETURN_CODE to MAPPING_NOT_SUPPORTED.
            Optionally log implementation-dependent error data.
         Otherwise (optional check when receiving data)
            Call PROTOCOL_ERROR_PROC (page 5.2-47)
             to deallocate the current conversation.
            Put the RETURN_CODE RESOURCE_FAILURE_NO_RETRY in the
             MC_RECEIVE_BUFFER of the current RCB.
   When it indicates map not found (see SNA Formats)
      Set RETURN_CODE to MAP_NOT_FOUND.
      Optionally log implementation-dependent error data.
   When it indicates map execution failure (see SNA Formats)
      Set RETURN_CODE to MAP_EXECUTION_FAILURE.
      Optionally log implementation-dependent error data.
   Otherwise (optional check when receiving data)
      Call PROTOCOL_ERROR_PROC (page 5.2-47)
       to deallocate the current conversation.
      Put the RETURN_CODE RESOURCE_FAILURE_NO_RETRY in the
       MC_RECEIVE_BUFFER of the current RCB.
```

GET_SEND_INDICATOR

FUNCTION:	This procedure is invoked during the processing that PS.MC performs as a result of receiving a return code of SVC_ERROR_PURGING. This procedure is called after the Error Data GDS variable that follows the service error notification has been received and processed. The purpose of this procedure is to receive the SEND indication that follows the Error Data GDS variable.
INPUT:	The RCB that corresponds to the specified conversation
OUTPUT:	A return code reflecting the results of the processing

Referenced procedures, FSMs, and data structures:

RECEIVE_AND_WAIT_PROC	page 5.1-19
PROTOCOL_ERROR_PROC	page 5.2-47
RCB	page A-6

```
Call RECEIVE_AND_WAIT_PROC(RECEIVE_AND_WAIT)(page 5.1-19) to issue a
 RECEIVE_AND_WAIT verb with the MC_RECEIVE_AND_WAIT verb parameters.

Select based on the return code in RECEIVE_AND_WAIT:
   When OK (optional check when receiving data)
      If RECEIVE_AND_WAIT returns WHAT_RECEIVED other than SEND then
         Call PROTOCOL_ERROR_PROC (page 5.2-47)
          to deallocate the current conversation.
         Set RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
   When RESOURCE_FAILURE_RETRY or RESOURCE_FAILURE_NO_RETRY
      Set RETURN_CODE to the value returned on RECEIVE_AND_WAIT.
   When DEALLOCATE_NORMAL, DEALLOCATE_ABEND_PROG,
    DEALLOCATE_ABEND_SVC, or DEALLOCATE_ABEND_TIMER
    (optional check when receiving data)
      Set RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
      Optionally log implementation-dependent error data.
   When PROG_ERROR_NO_TRUNC, SVC_ERROR_NO_TRUNC, or BACKED_OUT
    (optional check when receiving data)
      Call PROTOCOL_ERROR_PROC (page 5.2-47)
       to deallocate the current conversation.
      Set RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
```

SEND_SVC_ERROR_PURGING

FUNCTION:	This procedure performs service error purging processing. It is invoked when PS.MC receives a GDS variable specifying a function not supported by either the LU or the transaction program associated with the mapped conversation over which the GDS variable flowed, or when PS.MC receives a GDS variable containing an unrecognized GDS ID, or when data mapping is being performed and the mapper procedure has encountered an error in mapping the received data.
INPUT:	The RCB for the conversation on which the service error occurred; an error code specifying the type of error encountered; and an error parameter that provides more information about the error
OUTPUT:	If any of the verbs issued by this procedure do not complete successfully, the procedure inserts into the RCB.MC_RECEIVE_BUFFER an appropriate return code.
NOTE:	If mapping is supported and the mapper does not already know about the error, the mapper is notified of the type of error encountered. The mapper is not invoked when the error encountered is a MAP_NOT_FOUND or MAP_EXECUTION_FAILURE condition--the mapper is already aware of the error. (The mapper discovered the error.) If the error encountered indicates MAPPING_NOT_SUPPORTED, no mapper exists.

Referenced procedures, FSMs, and data structures:

UPM_MAPPER	page 5.2-46
PREPARE_TO_RECEIVE_PROC	page 5.1-18
SEND_DATA_PROC	page 5.1-24
SEND_ERROR_PROC	page 5.1-25
PROTOCOL_ERROR_PROC	page 5.2-47
RCB	page A-6
ERROR_DATA_STRUCTURE	page 5.2-48

```
If the input error code indicates an invalid GDS_ID (see SNA Formats)
 and the GDS_ID in the input error parameter does not indicate a map
 name (see SNA Formats) then
   Call UPM_MAPPER (page 5.2-46) to record a remotely detected error
    of the type SVC_ERROR_PURGING, as indicated by the RETURN_CODE from
    the last verb issued.
Call SEND_ERROR_PROC(SEND_ERROR)(page 5.1-25) to issue a SEND_ERROR
 verb with the MC_SEND_ERROR verb parameters, specifying error type SVC
 and implementation-dependent error log data.
Select based on the return code in SEND_ERROR:
   When OK
      Create an ERROR_DATA_STRUCTURE (a single logical record) using the
       data in the parameters ERROR_CODE and ERROR_PARM.
      Call SEND_DATA_PROC (page 5.1-24) to issue a SEND_DATA
       verb to send the ERROR_DATA_STRUCTURE to the remote TP.
      Select based on the return code from SEND_DATA:
         When OK
            Call PREPARE_TO_RECEIVE_PROC (page 5.1-18) to issue
             a PREPARE_TO_RECEIVE verb for the current conversation with the type
             parameter set to FLUSH and locks set to SHORT.
         When RESOURCE_FAILURE_RETRY or RESOURCE_FAILURE_NO_RETRY
            Put the RETURN_CODE from SEND_DATA in the MC_RECEIVE_BUFFER
             of the current RCB.
         When PROG_ERROR_PURGING, SVC_ERROR_PURGING, or BACKED_OUT
            (this check is optional when receiving data)
            Call PROTOCOL_ERROR_PROC (page 5.2-47)
             to deallocate the current conversation.
            Put the RETURN_CODE RESOURCE_FAILURE_NO_RETRY in the
             MC_RECEIVE_BUFFER of the current RCB.
         When DEALLOCATE_ABEND_SVC, DEALLOCATE_ABEND_TIMER,
          or DEALLOCATE_ABEND_PROG (optional check when receiving data)
            Optionally log implementation-dependent error data.
            Put the RETURN_CODE RESOURCE_FAILURE_NO_RETRY in the
             MC_RECEIVE_BUFFER of the current RCB.
   When DEALLOCATE_NORMAL, RESOURCE_FAILURE_RETRY, or RESOURCE_FAILURE_NO_RETRY
      Put the RETURN_CODE from SEND_DATA in the MC_RECEIVE_BUFFER
       of the current RCB.
```

UPM_MAPPER

FUNCTION: This procedure, referred to elsewhere in this chapter as "the mapper," performs mapping of data in an implementation-defined way. The MAPPER_SAVE_AREA in the RCB for the current conversation contains information used in data mapping, such as the currently effective map names (see "Map Names" on page 5.2-8). Refer to "Data Mapping and the Mapper" on page 5.2-8 for a detailed description of the processing that occurs when data is mapped.

INPUT:

1. Reason why the mapper was invoked:

- Data is to be sent to, or was received from, the partner LU. The map name supplied by the sending TP determines the kind of mapping to occur.
- An error occurred and was detected either remotely or locally.
- A positive reply to CONFIRM or to SYNCPT was received. This positive confirmation informs the mapper that any map names sent to the partner have been received and processed by it, and were not purged during error processing.

2. The polarity indicates whether send mapping or receive mapping is to be performed. This parameter is used when the mapper invocation is for data mapping.

3. The RH Format indicator indicates whether the passed data includes FM headers. The mapper requires this information in the event that the same map name could cause a different mapping to take place depending upon whether the data being mapped includes FM headers. This parameter is used when the mapper invocation is for data mapping.

4. Input map name contains the locally known map name supplied by the TP on an MC_SEND_DATA, if send mapping is to be performed; or the map name that flows in a Map Name GDS variable between LUs, if receive mapping is to be performed. This parameter is used only if the mapper invocation is for data mapping.

5. Input data contains the data supplied by the TP on the MC_SEND_DATA verb for SEND mapping, or data that flows in a Data GDS variable for RECEIVE mapping. This parameter is used only in data mapping.

6. Error code informs the mapper of the type of error encountered (for example, SVC_ERROR_PURGING or PROG_ERROR_NO_TRUNC). This is needed when the mapper invocation is for an error occurrence.

OUTPUT:

1. Output map name contains the "mapped" (global) map name that is sent to the partner LU if send mapping is performed, or the locally known map name that is passed to the TP if receive mapping was performed. This output is returned when the mapper invocation was for data mapping, and always after receive mapping.

2. Output data contains the data that is sent to the partner LU for send mapping, or the data that is passed to the TP for receive mapping. This data is returned if the the mapper was called for data mapping.

3. Mapper return code indicates whether the mapper successfully performed the mapping or encountered problems, and is returned after data mapping invocations.

PROTOCOL_ERROR_PROC

FUNCTION:	This procedure handles protocol error processing. It is invoked when PS.MC detects an architectural protocol error committed at the partner LU.
INPUT:	The RCB corresponding to the mapped conversation over which the protocol violation occurred.
NOTE:	Error log data is entered into the system log by PS.CONV (Chapter 5.1) during its processing of the DEALLOCATE issued by this procedure.

Referenced procedures, FSMs, and data structures:

DEALLOCATE_PROC page 5.1-15
RCB page A-6

Call DEALLOCATE_PROC(DEALLOCATE)(page 5.1-15) to issue a DEALLOCATE verb with the MC_DEALLOCATE verb parameters, specifying a deallocation type of ABEND_SVC and indicating that the resource ID is to be discarded.
Optionally, implementation-dependent error data may be recorded in the system error log.

ERROR_DATA_STRUCTURE

ERROR_DATA_STRUCTURE: an instance of a GDS variable
   LL_LENGTH: the high-order bit is set to 0 indicating a single-segment record
   GDS_ID (see format of an Error Data GDS variable in *SNA Formats*)
   DATA
      ERROR_CODE (see *SNA Formats*)
      ERROR_PARM (see *SNA Formats*)

SEND_BUFFER

SEND_BUFFER: a buffer containing the mapped data to be sent.

CHAPTER 5.3. PRESENTATION SERVICES--SYNC POINT SERVICES VERBS

Recovery from errors and failures is a central consideration in the design of transaction programs. LU 6.2 provides optional services to aid transaction programs in recovery from errors. A synchronization service is selected by the SYNC_LEVEL parameter in the ALLOCATE verb. This chapter is primarily concerned with the sync point synchronization services.[1]

ERRORS, FAILURES, AND RECOVERY

Errors and failures can be classified as:

- Application errors--these errors may occur frequently; recovery is part of the application design. In data entry, for instance, field validation and requests for repeated input are normal portions of the application logic.
- Recoverable system errors--these errors occur frequently; recovery is part of the system logic. Bracket race errors are an example (see "Chapter 6.1. Data Flow Control"); link-level retransmission is another.
- Transaction program failures--transaction programs sometimes end abnormally. In a well-tested system, this will not occur frequently. Application-level recovery varies by application. See "Chapter 5.1. Presentation Services--Conversation Verbs" for details of abnormal termination processing.
- Conversation failures--conversations will sometimes fail as a result of failure of the underlying sessions caused by the physical components over which the sessions are carried. The reactivation of failed sessions is handled by system logic; see "Chapter 4. LU Session Manager" for details. Application-level recovery from conversation failure is discussed in more detail in SNA Transaction Programmer's Reference Manual for LU Type 6.2.
- LU failures--LUs will sometimes fail by themselves or as a result of the failure of underlying hardware or software. Much of the recovery from LU failures, as seen by other LUs, is handled by the recovery of sessions that have failed. Other aspects of this recovery are the concern of sync point services.
- Local resource failures--local resources (e.g., files) will sometimes fail. If the local resource that fails is not protected by the sync point service, recovery is an application-level responsibility.

Applications are often designed as a sequence of logical units of work, each unit consisting of some changes to the resources under the control of the transaction program. Each logical unit of work (LUW) is recoverable by itself. The simplest case occurs when there is one LUW for a transaction program; recovery can often then consist of running the transaction again from the beginning. LUWs are delimited by the start-up of a transaction program and by execution of each SYNCPT verb. The SYNC_LEVEL(SYNCPT) service simplifies the design of transaction programs that use protected resources, since changes to those resources will be seen by the application transaction program as having occurred only after one LUW completes and before the next LUW begins.[2]

Figure 5.3-1 on page 5.3-2 illustrates the relationships among failures and recovery.

[1] Full support of sync point services in actual implementations includes provisions for synchronizing local resources as well as distributed resources accessed through conversations. For completeness, this section sketches fully general sync point services. Details of sync point services for local resources are not specified by SNA, but are implementation defined.

[2] The sync point service is not always able to provide a consistent state for the protected resources. When this occurs, a heuristic decision is made. This sometimes damages the LUW by making the states of its protected resources inconsistent. More details about this are provided in "RESOURCE_FAILURE_*, Recovery, and Heuristic Decisions" on page 5.3-15.

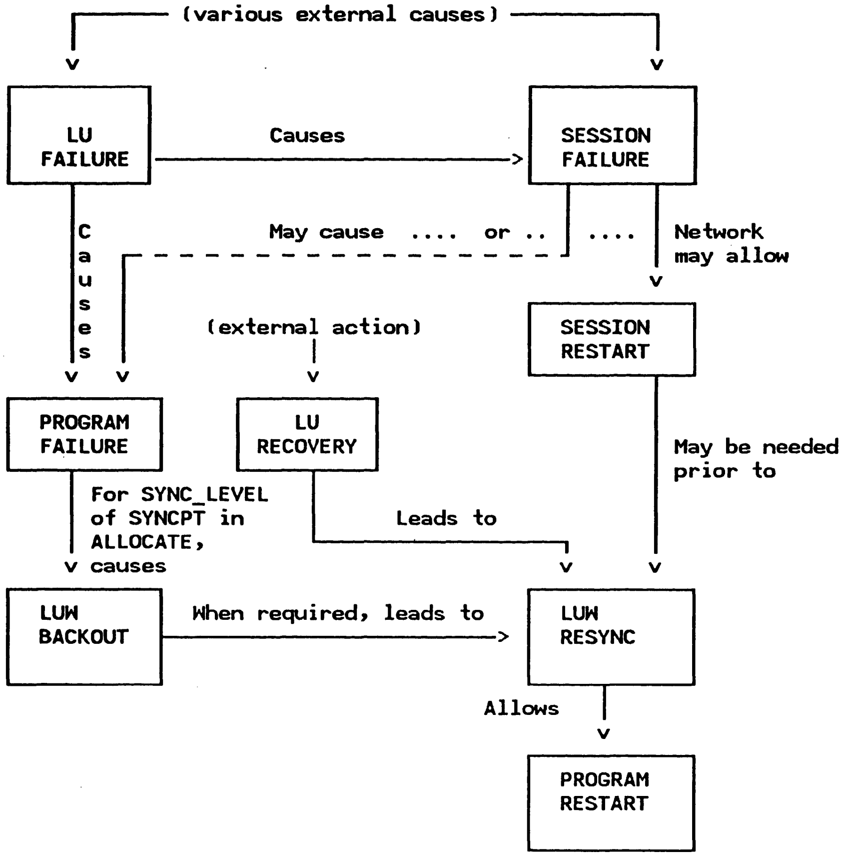


Figure 5.3-1. Relationships among Failures and Recovery

SYNC POINT CONCEPTS

The following are some terms that are used in this chapter:

- SYNCPT—A verb used by a transaction program (TP) to invoke sync point services. Sync point services coordinate the updates of distributed resources. Coordination is performed by the sending and receiving of presentation services (PS) headers by the sync point services component. The protocol allows recovery if messages are lost because of transaction program, conversation, or LU failures.

- INITIATOR—The role of the local sync point services component when the TP issues the SYNCPT verb that begins the coordinated update of distributed resources.

- AGENT—The role of the sync point services component that receives sync point requests from an initiator.

- CASCADED AGENT—An agent of an initiator that is itself an agent of another initiator; in other words, an agent may allocate other protected conversations. In this role an agent is responsible for propagating sync point requests to its cascaded agents.

- RESYNC—Recovery processing that is performed by sync point services after a failure of a session, transaction program, or LU. The resync exchange includes exchanging log names and comparing LUW states.

- PRESENTATION SERVICES (PS) HEADER—The requests and replies that sync point services components exchange to perform SYNCPT verb processing.

PROCESSING BY PS.SPS

The component of LU presentation services that provides the sync point service is called PS.SPS, also called the sync point manager. When all the resources used by a TP are at one LU, only one copy of PS.SPS is executed. Usually the situation is more complicated since every conversation allocated with the SYNC_LEVEL(SYNCPT) option connects two separate TPs, which cooperate to perform one or more distributed units of work. In the distributed cases, one TP is the first to issue the SYNCPT verb, and its local sync point manager becomes the sync point initiator for the current sync point, with respect to the sync point managers on the other ends of any conversation. These other sync point managers become agents with respect to the initiator, but may in turn become initiators with respect to additional, cascaded, sync point managers.

The sync point managers maintain consistency of the changes to protected resources by the propagation throughout the network of these sync point commands:

- Prepare--Solicits Request Commit. This command tells the agent to place its protected resources in a state that allows them to be fully committed to the changes that have been accumulated during this LUW, but that also allows these changes to be reversed, or backed out. The choice to commit or back out is made by the initiator after interaction with all agents.
- Request Commit--Solicits Committed. This command says that the issuer has succeeded in preparing all of its protected resources.
- Committed--Informs the soliciting sync point manager that all resources attached through this conversation are committed.
- Forget--Informs the sync point manager that sent Committed that its log record for this LUW can be erased.[3] Forget also tells the initiating sync point manager that the sync point is complete and that control can be returned to the TP.
- Backed Out--Informs the receiving sync point manager that the sending sync point manager has backed out the LUW.

The SNA encoding for transmission of these commands are described in SNA Formats under presentation services (PS) headers for the first four, and FMH-7 sense data for Backed Out.

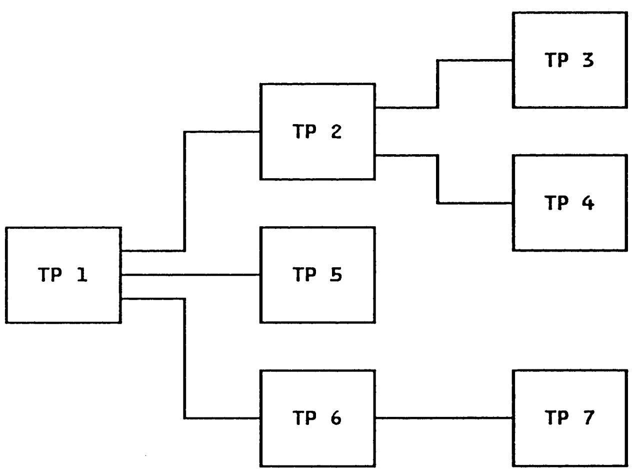


Figure 5.3-2. A Typical Sync Point Tree

[3] The sync point managers keep records about LUWs on logs, held on nonvolatile storage by the log manager, so that LUWs can be kept consistent across failures of LUs. The logical unit of work ID (LUWID) is comprised of three components: the fully qualified LU network name; the instance number, which is unique at the LU that creates it; and the sequence number, which is incremented by 1 following a successful sync point. In addition, a conversation correlator is used to further qualify LUWIDs. The LUWID is created by RM for a conversation whenever a conversation is allocated by a TP that does not already have an LUWID associated with it. A TP already has an LUWID associated with it, if it was the subject of an Attach by a TP that already has an LUWID. The LUWID and conversation correlator are carried in the FMH-5 (see SNA Formats).

LUW STATES

A distributed transaction program is a tree, with individual TPs as nodes on the tree, and conversations as branches. Distributed TPs support distributed LUWs, consisting of local LUWs at the individual TPs. The distributed LUW has a state made up of all the local LUW states. For the distributed transaction program shown in Figure 5.3-2 on page 5.3-3, the distributed LUW state is a vector with seven components:

```
LUW = [LUW1,LUW2, ... LUW7]
```

where LUWi is the local LUW state for TPi. The first TP to issue SYNCPT becomes the root of the tree for the global LUW that is ended by that verb. In the figure, the root, or initiator, is TP 1.

The sync point managers at each node of the tree cooperate to place all the LUW components into the same consistent state. They do this with four waves of sync point commands.

The Prepare wave starts at the root and spreads down the tree. The Request Commit wave starts at the leaves (nodes without subordinate nodes) and spreads up the tree to the root. The Committed wave returns down the tree, and the Forget wave flows up the tree to the root. Figure 5.3-3 shows these waves as they occur between the root and one of the nodes adjacent to the root.

```
              PS.SPS          PS.SPS
   TP         Initiator       Agent         TP

 SYNCPT
 ------->            Prepare
               ----------------------->   TAKE_SYNCPT *
                                          ------->
                                          SYNCPT
                                          <-------
                   Request Commit
               <-----------------------
                   Committed
               ----------------------->
                   Forget
               <-----------------------
 OK
 <-------
```

NOTE: TAKE_SYNCPT is returned in the WHAT_RECEIVED field of verbs that can receive data.

Figure 5.3-3. Basic Sync Point Flows

```
              PS.SPS          PS.SPS
   TP         Initiator       Agent         TP

 SYNCPT
 ------->            Prepare
               ----------------------->   TAKE_SYNCPT
                                          ------->
                                          SYNCPT
                                          <-------
                   Forget
               <-----------------------
 OK
 <-------
```

Figure 5.3-4. Optimized Flow: No Resource Changed

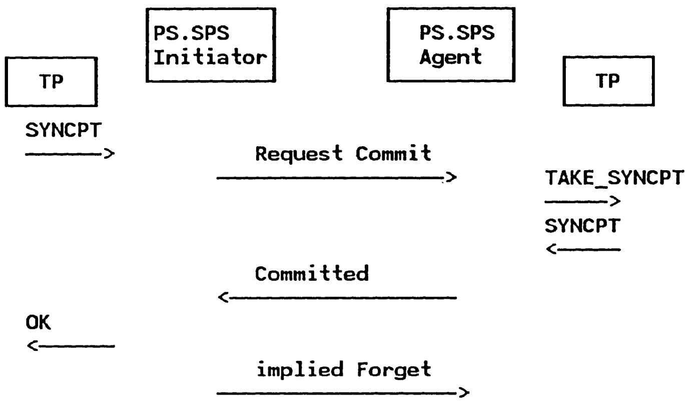


Figure 5.3-5. Optimized Flow: Last Resource

FLOW OPTIMIZATION

Since message flows are costly, the sync point managers attempt to reduce the number of flows. Figure 5.3-4 on page 5.3-4 illustrates one such case: when a sync point manager agent determines that the state of the local LUW is reset, that is, no protected resources have been changed, it answers Forget to Prepare. Intermediate agents can reply Forget only if all the local LUWs in their entire subtree are reset.

Figure 5.3-5 shows the other flow reduction that can be used. The initiator can pick one adjacent agent to receive Request Commit rather than Prepare. The Request Commit can be sent only after all the prepared agents have sent Request Commit up their subtree to the initiator, making the selected agent the last agent. This last agent is then free to select one of its cascaded agents also to be last, and so on.

Message flows are further reduced because the PS header that starts the sync point exchange indicates that one of three things should occur after the sync point message exchange is complete: the initiator is to be in send state, the initiator is to be in receive state, or the conversation is to be deallocated. This is shown in Figure 5.3-32 on page 5.3-38 to Figure 5.3-36 on page 5.3-40. The first PS header sent has a modifier field that indicates the setting of the CD and CEB indicators of the RH that completes the sync point exchange.

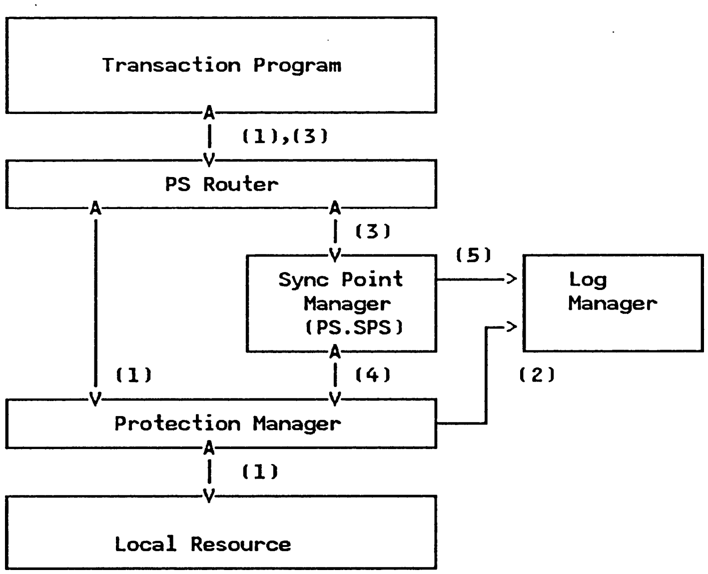


Figure 5.3-6. Sync Point Services for Local (Nonconversational) Resources, Such as Files

SYNC POINT AND OTHER LU COMPONENTS

The relationships among the transaction program, its resources, and the sync point manager are illustrated in Figure 5.3-6 through Figure 5.3-8.

The following notes correspond to the numbers in Figure 5.3-6 on page 5.3-5.

1. The transaction program issues a resource verb, which is passed, by the PS router, to the proper procedure to handle the local resource. See "Chapter 5.1. Presentation Services--Conversation Verbs" for details.

2. The local resource is protected, and so it has a protection manager, which examines the resource verb. If the resource is changed by the verb (e.g., it is a Write of some kind), the protection manager writes a log record containing the before-change data.[4]

3. Eventually the transaction program issues SYNCPT or BACKOUT. The PS router invokes the sync point manager, which coordinates the action of all sync point managers involved in the distributed LUW.

4. The sync point manager interacts with the protection manager for each protected resource, exchanging PS headers indicating Prepare, Request Commit, Committed, and Forget to coordinate commitment, or an FMH-7 indicating Backed Out to coordinate backout of changes, either as requested by the TP, or as required by a resource failure.

5. When all resources are prepared, the LUW is committed when the sync point manager writes Committed on the log, and forces the log.[5] The single force of the log is sufficient to commit the entire LUW because all local resources used by a single TP share a single log, which is also the log used by the TP's sync point manager.

   Recovery that uses the log records is discussed later in "Resynchronization Logic" on page 5.3-18.

[4] Logging before-change data is the technique suggested in the formal description. Other equivalent techniques are possible and permissible.

[5] Some writes to the log can be made to volatile log buffers. If these are lost because of a failure of the LU, no damage results. Other writes (called <u>forced</u> writes) to the log must be made to the nonvolatile log itself before the sync point protocol can proceed, if the LUW is to be kept synchronized even across LU failures. This use of the nonvolatile log is called <u>forcing</u> <u>the</u> <u>log</u>.

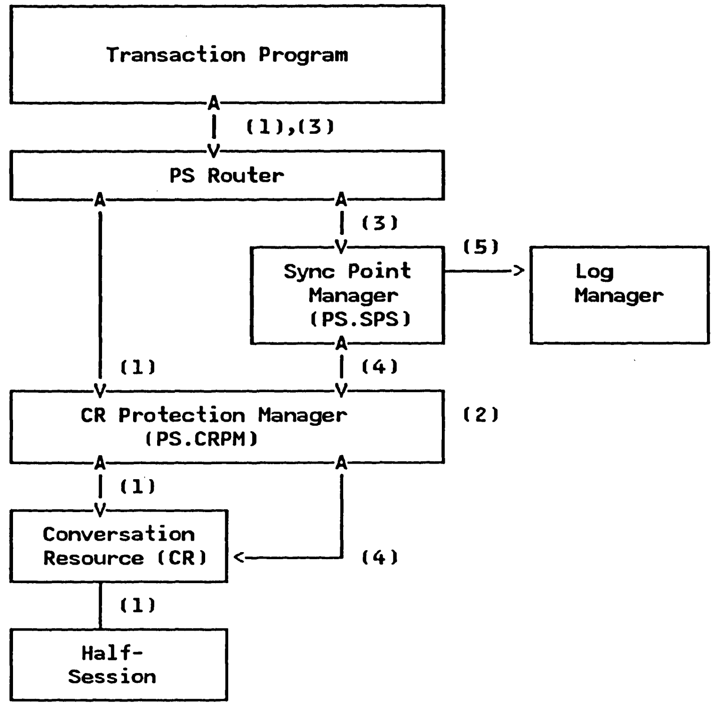


Figure 5.3-7. Sync Point Services for Conversation Resources

The following notes correspond to the numbers in Figure 5.3-7.

1. The transaction program uses a conversation. The conversation resource (CR) protection manager is not sensitive to any of the conversation verbs.

2. The CR protection manager does not write any log records. RM does write log records as part of ALLOCATE processing in order to be able to re-create the resource control blocks (RCBs) and their relationship to transaction control blocks (TCBs) following an LU failure. See RCB on page A-6 for details of the RCB and TCB.

3. Eventually the transaction program issues SYNCPT or BACKOUT. The PS router invokes the sync point manager to do the coordination.

4. The sync point manager interacts with the protection manager for each protected conversation, exchanging Prepare, Request Commit, Committed, and Forget PS headers to coordinate commitment, or Backed Out to coordinate backout of changes, either as requested by the TP, or as required by a resource failure.

   Protected conversations are treated somewhat differently from protected local resources; this difference is driven by a Backed Out FMH-7 can be received from nonlocal resources. Compare States GDS local/nonlocal[6] indicator in the RCB. A variables (also referred to as Compare States command or reply) can be exchanged with them to resynchronize following conversation failures.

   The local protection manager for the conversation communicates with its remote partner by exchanging PS headers and the Backed Out FMH-7 sense data. The half-session has no knowledge that a protected conversation is assigned to it.

5. The sync point manager has to do additional writes to the log whenever nonlocal resources are pointed to by a TCB. Also, additional forces of the log are required. Finally, the sync point manager attempts resynchronization by an exchange of Compare States GDS variables with its partner sync point manager after resource failures.

[6] Local resources are those that share the sync point manager's log.

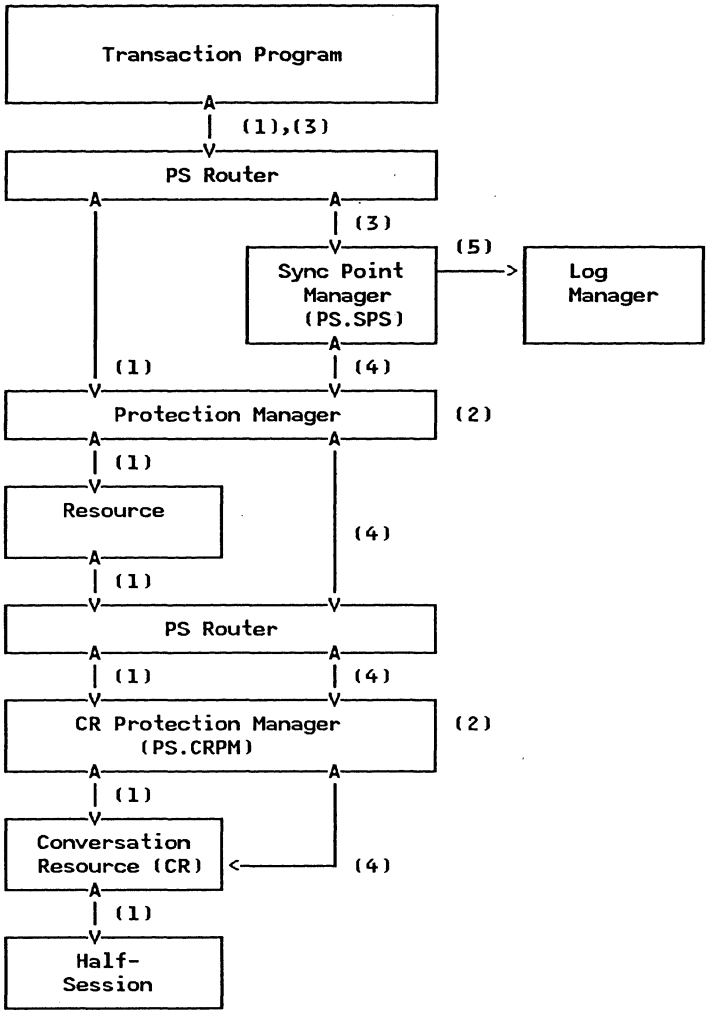


Figure 5.3-8. Sync Point Services for Function Shipping

The following notes correspond to the numbers in Figure 5.3-8.

1. The transaction program allocates a resource that is located remotely. The local resource manager uses a conversation to communicate to the remote resource.

2. Neither the local-resource protection manager nor the CR protection manager writes log records. The only logging is done by RM in order to be able to re-create the resource RCBs and their relationship to TCBs. The ALLOCATE issued by the local resource manager is understood to be for a function shipping situation, so the conversation's RCB is chained under the local resource's RCB rather than being chained directly to the TCB. At the same time, the local resource's RCB is marked nonlocal.

3. Eventually the transaction program issues SYNCPT or BACKOUT. The PS router invokes the sync point manager to do the coordination.

4. The sync point manager interacts with the protection manager for each protected resource, exchanging Prepare, Request Commit, Committed, and Forget PS headers to coordinate commitment, and Backed Out to coordinate backout of changes, either as requested by the TP, or as required by a resource failure.

   The nonlocal resources are treated the same as protected conversations: Backed Out can be received; Compare States GDS variables can be exchanged.

   The protection manager for the local resource, after dealing with local states (e.g., on a Prepare it may need to flush a local buffer), passes the PS headers that it receives from the sync point manager to the CR protection manager.

5. The sync point manager has to do additional writes to the log whenever nonlocal resources are pointed to by a TCB. Also, additional forces of the log are required to handle the extra error states introduced by the existence of remote logs. Finally, the sync point manager attempts resynchronization via exchange of Compare States GDS variables with partner sync point managers after resource failures.

SYNC POINT LOGIC

A transaction program can issue a SYNCPT verb as an initiator, or in reply to a WHAT_RECEIVED value of TAKE_SYNCPT, TAKE_SYNCPT_SEND, or TAKE_SYNCPT_DEALLOCATE on RECEIVE. After giving the TAKE_SYNCPT indication, the conversation resource rejects most verbs until SYNCPT, BACKOUT, or SEND_ERROR is issued. See SNA Transaction Programmer's Reference Manual for LU Type 6.2 for details.

PS.SPS processes the SYNCPT verb in the phases described below.

CLASSIFICATION PHASE

Since SYNCPT can be issued under many circumstances, PS.SPS begins by scanning the resources allocated to the transaction program in order to determine their states. Further PS.SPS processing varies according to the states of the local resources and TP:

1. PREPARE RECEIVED state--Prepare was received from an initiating sync point manager. The local TP did not initiate sync pointing. PS.SPS prepares its local and down-tree protected resources and replies up-tree with Request Commit if preparation succeeds. If it fails, it replies Backed Out.
2. REQUEST COMMIT RECEIVED state--Request Commit was received from an initiating sync point manager. The local TP did not initiate sync pointing. Since the initiating PS.SPS has used an optimized flow, which it can do only for the last resource that it is attempting to coordinate, the local PS.SPS coordinates the commitment of its local and down-tree resources and replies Committed if commitment succeeds. If it fails, it replies Backed Out.
3. SEND state--All protected conversations are verified to be in SEND state. Before issuing the SYNCPT verb, the transaction program puts all its protected resources into SEND state. If required, this can be done by issuing REQUEST_TO_SEND and waiting for the right to send.
4. Unprotected resource--Resource was allocated with SYNC_LEVEL(NONE | CONFIRM). The resource is not affected by the SYNCPT verb.

At the end of the scan, PS.SPS knows if a resource (i.e., the one in PREPARE RECEIVED state) must be sent Request Commit during its local coordination. Request Commit must be sent last, after all other resources have been prepared. If no last resource is identified, a UPM is used to select one. The UPM can consider things like minimizing session flows (which leads to making a remote conversation last whenever possible). It can also choose to prepare all resources, which allows all coordination to proceed in parallel, since Prepares can be sent simultaneously to several resources.

If any protected resources are in Receive state or more than one last resource is identified, the sync point manager recognizes a state error and abnormally terminates the TP. Since any TP may be sync point initiator, the design of the distributed TPs must be such that only one TP at a time is the initiator. For example, TPa is in conversation with TPb and TPb has a cascaded conversation with TPc. If TPa and TPc both initiate sync point with TPb at the same time, it is an error in the design of the transaction program. The sync point service at TPb recognizes this error and returns BACKED_OUT to TPb. TPb then issues the BACKOUT verb. Otherwise, PS.SPS advances to the Prepare phase.

PREPARE PHASE

PS.SPS now issues Prepare to all not-last resources. When Request Commit has been received from all of them, the next phase is entered. Other replies to Prepare are discussed in "Errors during Sync Point" on page 5.3-15. If no not-last resources exist, this phase is skipped and PS.SPS proceeds directly to the Request Commit phase.

REQUEST COMMIT PHASE

After receiving Request Commit from all not-last resources, PS.SPS issues Request Commit to the last resource, and waits for a reply, thus entering the Committed phase.

COMMITTED PHASE

PS.SPS completes sync point processing after receiving Committed from the last resource by sending Committed to all not-last resources, thus entering the Forget phase.

FORGET PHASE

In the Forget phase, PS.SPS waits for Forgets from all the not-last resources. When all

Forgets have been received, PS.SPS gives the SYNCPT verb that was issued by the local TP a return code of OK.

ILLUSTRATIVE SYNC POINT FLOWS

The following figures and comments illustrate the preceding discussion.

```
        +--------+        +-------+          +---------+
        |PS.SPS  |        |PS.SPS |          |Cascaded |
+--+    |Initiator        |Agent  |  +--+    |Agent    |
|TP|    +--------+        +-------+  |TP|    +---------+
+--+                                 +--+
          RESET             RESET
            V                 V
----> (1)                            <----
            A                 A
            |                 |
       PGM PENDING       PGM PENDING
            |                 |
SYNCPT      V                 |
----> (2)      Prepare*       |
            A  ----------->(3)|  TAKE_SYNCPT
            |                 |  ------->
            |                 |  SYNCPT
            |                 |  <-------
            |                 V
       SPM PENDING       (4)     Prepare*
            |                 A  ------------------>
            |                 |
            |            SPM PENDING
            |                 |
            |                 V
            |            (5)     Request Commit*
            V  Request Commit* A  <------------------
      (6)      <--------      |
            A  Request Commit |
            |  ------->       |
            |                 |
       IN DOUBT          IN DOUBT
            |  Committed      |
            |  <-----         |
            V                 |
      (7)                     |
               implied Forget |  (8)
            A  ------->       |
            |  Committed*     V
            |  ----------->(9)
            |                 A  Committed*
            |                 |  ------------------>
       COMMITTED         COMMITTED
            |                 |
            |                 |  Forget*
            |                 |  <------------------
            |  Forget*        V  RETURN_CODE
            V  <-----------(10)  ------->
      (11)
RETURN_CODE
<----
```

NOTE: The * indicates sending to, or receiving from, multiple agents.

Figure 5.3-9. Illustrative Sync Point Flow: General Case

The following notes correspond to the numbers in Figure 5.3-9.

1. The distributed LUW begins in RESET state. Any change to a local protected resource or receipt by PS of any message unit (including the initial Attach) over a protected conversation drives a local LUW from RESET to PGM PENDING.

2. The initiating TP issues SYNCPT. PS.SPS logs all affected conversations except the last as [INITIATOR, SPM PENDING]

while the last one is not logged yet.[7] The log is forced once. PS.SPS sends Prepare to all but the last agent (where the * at the end of Prepare means all the agents, except possibly the last).

3. Each agent PS.SPS returns to its transaction program, a WHAT_RECEIVED value of TAKE_SYNCPT. All TPs agree by issuing SYNCPT.

4. The agent PS.SPS logs [AGENT, SPM PENDING] for the conversation over which the Prepare is received. It logs [INITIATOR-CASCADE, SPM PENDING] for all the cascaded conversations, if any exist (there might only be local resources). The log is forced once if and only if any cascaded conversations exist.

5. All cascaded agents agree to commit. PS.SPS places [AGENT, IN DOUBT] on the log and forces the log.

6. All agents agree to commit. [INITIATOR, IN DOUBT] is placed on the log if and only if the last resource is being optimized with the last-resource sequence. If IN DOUBT is placed on the log, the log is forced and then Request Commit is sent to the last agent.

7. The last agent replies Committed (if the optimized flow is being used for the last agent). [INITIATOR, COMMITTED] is logged and the log is forced. Committed is sent to all agents (except the optimized last).

8. An implied Forget is sent to the last agent with the aid of RM and the session process. The implied Forget is the next normal-flow RU of any kind that flows from the initiator to the last agent. For instance, if the agent sent Committed as CEB, then the next RU might be a (BB, Attach); or it might be a (BB, LUSTAT); or BIS; or a data reply to a BB that came from the agent's half-session. Since the Committed can get lost, the agent retains the state of the LUW across session outage. Since the implied Forget can get lost, and since the initiator may have erased its log, the agent carries a resync responsibility for itself. Only in this way can it erase its log. "Resynchronization Logic" on page 5.3-18 describes resync in more detail.

9. PS.SPS logs [Initiator-Cascade, Committed] for all cascade agents and forces the log. It then sends Committed to the cascaded agents.

10. All cascaded agents return Forget. PS.SPS resets the LUW by erasing the log; then PS.SPS sends Forget to the initiator, and returns control to the agent TP.

11. All agents return Forget. PS.SPS erases the log and returns control to the initiating TP. The log does not have to be forced before PS.SPS sends Forget, since any Forgets lost during a failure can be reconstructed by resynchronizing with cascaded agents.

---

7 The log records are [state of local PS.SPS relative to remote PS.SPS, state of local LUW].

```
        +-----------+     +---------+                +----------+
        |PS.SPS     |     |PS.SPS   |                |Cascaded  |
+----+  |Initiator  |     |Agent    |    +----+      |Agent     |
| TP |  +-----------+     +---------+    | TP |      +----------+
+----+     RESET             RESET       +----+
             V                 V
------> (1)                              <------
             A                 A
             |                 |
       PGM PENDING       PGM PENDING
             |                 |
SYNCPT       V                 |
------> (2)     Request Commit |
             A  ------------->(3)  TAKE_SYNCPT
             |                 |   ---------->
             |                 |   SYNCPT
             |                 |   <----------
             |                 V
       IN DOUBT           (4)     Prepare*
             |                 A  ------------------------>
             |                 |
             |          SPM PENDING
             |                 |
             |                 V
             |            (5)     Request Commit*
             |                    <------------------------
             |            (6) A   Committed*
             |                |   ------------------------>
             |          INITIATOR-CASCADE, COMMITTED
             |                |   Forget*
             |                V   <-----------------------
             |            (7) A
             V  Committed     |   RETURN_CODE
        (8)     <------------ |   ---------->
             A                |
             |                |
       COMMITTED        COMMITTED
RETURN_CODE  |                |
<-----       V                |
           RESET              |
                              |
            implied Forget    |  (9)
            ------------->    V
```

NOTE: The * indicates sending to, or receiving from, multiple agents.

Figure 5.3-10. Illustrative Sync Point Flow: Last-Resource Optimization

The following notes correspond to the numbers in Figure 5.3-10.

1. The distributed LUW begins in RESET state. Any change to a local protected resource or receipt by PS of any message unit (including the initial Attach) over a protected conversation drives a local LUW from RESET to PGM PENDING.

2. The initiating TP issues SYNCPT. PS.SPS logs the last conversation as [INITIATOR, IN DOUBT]. It forces the log and sends Request Commit.

3. The agent PS.SPS presents TAKE_SYNCPT to the agent transaction program. The TP agrees by issuing SYNCPT.

4. The agent PS.SPS logs [AGENT, SPM PENDING] for the conversation over which the Request Commit is received. It logs [INITIATOR-CASCADE, SPM PENDING] for all the cascaded conversations, if any exist (there might be only local resources). It forces the log if and only if any cascaded conversations exist.

5. All cascaded agents agree to commit. The agent PS.SPS logs [INITIATOR-CASCADE, COMMITTED] and forces the log again (in the example, the agent is not using the last-resource optimization on cascaded resources). Then it sends Committed to all cascaded agents.

6. The agent PS.SPS waits for all cascaded agents to return Forget. This is done so that, in case of failures and resynchronization, it can return to the initiator an accurate report of any damage that may occur from heuristic decisions (discussed in "DEALLOCATE_ABEND_*" on page 5.3-15).

7. All Forgets are returned. The subtree for which this PS.SPS is responsible is

COMMITTED. The agent PS.SPS returns Committed to the initiator, even if no down-tree resources were changed, and then returns control to its TP.

8. The initiator sees the Committed. If there are no other participants, the initiator erases the log for the LUW and returns OK to the initiating transaction program. If there are other agents, [INITIATOR,COMMITTED] is placed on the log while the Forgets from the not-last agents are collected. See Figure 5.3-9 on page 5.3-11 for this type of sequence.

9. Implied Forget is sent to the last agent with the aid of the session process. That is, any conversation data that flows on the half-session is treated as an implied Forget. This includes a BB that begins a new conversation when Committed was sent with CEB.

```
           +---------+       +--------+           +---------+
           |PS.SPS   |       |PS.SPS  |           |Cascaded |
 +----+    |Initiator|       |Agent   |  +----+   |Agent    |
 | TP |    +---------+       +--------+  | TP |   +---------+
 +----+      RESET            RESET      +----+
               V                V
 ------> (1)                             <------
               A                A
               |                |
          PGM PENDING      PGM PENDING
               |                |
 SYNCPT        V                |
 ------> (2)       Prepare*     |
               A   ----------->(3)   TAKE_SYNCPT
               |                |    ---------->
               |                |    SYNCPT
               |                V    <---------
               |                A
          SPM PENDING           |
               |                |
               |           SPM PENDING
               |                |
               |                |
               |   Forget       V    RETURN_CODE
               V   <-----------(4)   ---------->
         (5)
```

NOTE: The * indicates sending to, or receiving from, multiple agents.

Figure 5.3-11. Illustrative Sync Point Flow: No Resources Changed

The following notes correspond to the numbers in Figure 5.3-11. The situation that the figure illustrates arises when a sync point is requested, but no remote resources have been altered during the LUW. In this case, the Request Commit and Committed flows are not necessary and are omitted.

1. The distributed LUW begins in RESET state. Any change to a local protected resource or receipt by PS of any message unit (including the initial Attach) over a protected conversation drives a local LUW from RESET to PGM PENDING.

2. The initiating TP issues SYNCPT. PS.SPS logs all affected conversations but the last as [INITIATOR, SPM PENDING], not logging the last one yet. It forces the log once, then sends Prepare to all but the last agent (represented by the * following Prepare).

3. The agent PS.SPS presents TAKE_SYNCPT to the agent TP, which agrees to commit. The rest of this flow illustrates the processing performed by a single agent where no resources have been changed. The generalization to cascaded LUWs is straightforward.

4. The agent PS.SPS sees (by receiving Forgets from the local resources) that no resources have been changed. It resets the LUW by erasing the log, sends Forget to the initiator, and returns control to the agent TP.

5. The agent returns Forget. The Request Commit and Committed flows were not needed; the initiator PS.SPS still processes the flows from other conversations that may or may not require the additional flows.

FORCING THE LOG

PS.SPS needs to force the log only once when all resources are local, while it uses at least two forces of the log as the initiator (SPM PENDING and COMMITTED states) and may use an additional force (IN DOUBT state) if the last resource is flow optimized.

PS.SPS uses at least one log force as the agent (IN DOUBT state), but if any cascaded conversations exist for this LUW, the agent PS.SPS has to appear to the cascaded agents as if it were the initiator. Therefore, the middle agent has to force the log (SPM PENDING state) in order to reliably assume the resync responsibility if it should terminate abnormally. The middle agents do not need to force the log to COMMITTED state since resync will re-establish this state if it is lost.

ERRORS DURING SYNC POINT

The preceding discussion assumed that sync point processing completed normally, without incident. This section shows how consistency can be maintained even when errors occur.

The errors addressed are those caused by many transaction programs operating independently of each other, communicating only when required. With this independence, unexpected return codes can occur after any verb. As the issuer of internal verbs to the conversation resource protection manager (PS.CRPM) in order to exchange sync point commands with partner sync point managers, PS.SPS has logic to deal with these return codes:

- PROG_ERROR_*, including SVC_ERROR_*
- BACKED_OUT
- DEALLOCATE_ABEND_*
- RESOURCE_FAILURE_*

Because recovery from conversation failure can require that a session be reactivated, PS.SPS gives special consideration to the case where this cannot be accomplished in a timely manner.

PROG_ERROR_*

PS.SPS treats PROG_ERROR_* as BACKED_OUT. It is the using transaction program's responsibility to avoid this by correct transaction design.

BACKED_OUT

BACKED_OUT is the return code given when the remote transaction program issues a BACKOUT verb. Unlike the case of PROG_ERROR_*, where the TP that issued SEND_ERROR gives the TP that receives the PROG_ERROR_* an option, on BACKOUT the issuing TP expects the entire distributed LUW to be backed out. The TP that receives BACKED_OUT therefore propagates the backout to all other resources by also issuing the BACKOUT verb.

DEALLOCATE_ABEND_*

PS.SPS may receive DEALLOCATE_ABEND_*. Since PS.SPS for the abnormally terminating TP will back out all of the TP's local resources, the local PS.SPS treats these return codes as BACKED_OUT.

RESOURCE_FAILURE_*, RECOVERY, AND HEURISTIC DECISIONS

Recovery from conversation failure depends upon the state of the conversation at the time of the outage:

1. If the conversation is under the control of the sync point manager, it attempts to recover from the failure by exchanging Compare States GDS variables with the remote sync point manager as part of resync processing. PS.SPS does this by issuing ALLOCATE specifying the LU resync service TP X'06F2' as the transaction program. See "Resynchronization Logic" on page 5.3-18 for the logic that is executed during this resynchronization effort.

   If resynchronization succeeds, PS.SPS absorbs the RESOURCE_FAILURE_* return code and returns from the SYNCPT or BACKOUT verb with the appropriate SYNCPT or BACKOUT return code. PS gives the RESOURCE_FAILURE_* return code to the TP on the next verb (other than SYNCPT and BACKOUT[8]) issued against the failing conversation, thus making the sync point verb and the resource failure appear to have occurred in the reverse order. This is done for the convenience of the TP writer. A TP that is using protected resources can take advantage of this by issuing SYNCPT or BACKOUT whenever a conversation failure return code is recognized. This gets the TP to a known

[8] If SYNCPT and BACKOUT returned RESOURCE_FAILURE_* there would be no way to resynchronize short of an IPL to drive the resync logic.

state: backed out to the last successful sync point call. Backed out state is arrived at when BACKOUT or SYNCPT is issued, after a resource failure, because a resync occurs. In this case, resync can only lead to backed out. The TP can then perform its own recovery logic from a known state, greatly simplifying the TP's recovery logic.

Because a new session may not be immediately available, the sync point manager and the lock manager have a protocol boundary that provides a capability to free locks on resources that may be needed by other TPs. When the lock manager needs to release locks, PS.SPS uses the guidance provided by the TP's entry in the transaction program list in RM, the LU control operator, or a programmed operator. The choices are either to hold the locks or to choose to do a partial commit or a partial backout of those resources with which communication has been maintained. The guidance (not shown in this book) indicates whether committing, backing out, or holding the locks is to be performed when the TP fails and the lock manager needs to release locks. As PS.SPS makes this decision with only partial information, it is called a <u>heuristic decision</u>.

PS.SPS reports the resource state (whichever is chosen, HEURISTIC COMMIT or HEURISTIC RESET) to the LU control operator (since the heuristic decision may result in a loss of synchronization among the distributed resources that has to be repaired by operator action) and saves the state for comparison during resynchronization. The PS.SPS that is responsible for resync continues resync attempts until resync completes. At this time, PS.SPS writes another message to the LU control operator and erases the LUW's log entries.

2. If the conversation is not under the control of the sync point manager, the responsibility for recovery is the transaction program's. However, if sync point is in use, the TP can typically turn the recovery processing over to the sync point manager by using the SYNCPT or BACKOUT verb as soon as any desired processing has been completed. Resources that are not protected are cleaned up according to application program logic. A failure by one TP or the other to return control to the sync point manager can lead to an extended holding of locks on shared resources. It may also lead to heuristic decisions if the locks have to be broken.

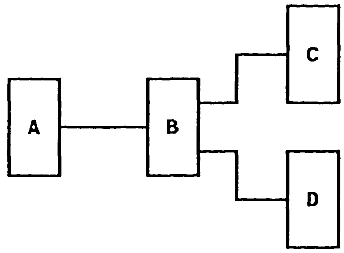
BACKOUT PROCESSING

When processing the BACKOUT verb, PS.SPS causes all protected resources in the LUW to be restored to their condition at the start of the LUW. The exception is that protected conversations are not deallocated, and the remote TPs that they started are not terminated by backout processing.

Like SYNCPT, BACKOUT is propagated to all TPs associated with the LUW. Also like SYNCPT, BACKOUT propagation requires all transaction programs that share a distributed unit of work to participate by issuing verbs, i.e., BACKOUT.

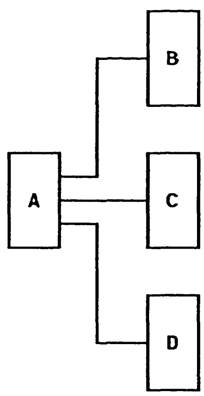
When a transaction program is notified of a BACKOUT initiated by another transaction program, the remote BACKOUT is complete. That is, the conversation resource that reports BACKED_OUT has already done so. The return code indicating this, BACKED_OUT, may be returned on several of the verbs. No backout of other resources in the local unit of work has been done. The TP must issue BACKOUT before it issues any other verb against protected resources.

Of particular interest is the case where BACKOUT is issued in the midst of SYNCPT processing. The locally issued BACKOUT takes precedence over the SYNCPT requested by the remote TP if the LUW stays intact. See Figure 5.3-12 and Figure 5.3-13 for examples that illustrate how this is accomplished. For brevity, the Forget commands are not shown.



Committed Sequence	Backed Out Sequence 1	Backed Out Sequence 2
1. A -> B - Prepare	1. A -> B - Prepare	1. A -> B - Prepare
2. B -> C - Prepare	2. B -> C - Prepare	2. B -> C - Prepare
3. B -> D - Prepare	3. B -> D - Prepare	3. B -> D - Prepare
4. C -> B - Request Commit	4. C -> B - Request Commit	4. C -> B - Request Commit
5. D -> B - Request Commit	5. D -> B - Backed Out	5. D -> B - Request Commit
6. B -> A - Request Commit	6. B -> C - Backed Out	6. B -> A - Request Commit
7. A -> B - Committed	7. B -> A - Backed Out	7. A -> B - Backed Out
8. B -> C - Committed		8. B -> C - Backed Out
9. B -> D - Committed		9. B -> D - Backed Out
STATUS = Committed	STATUS = Backed Out	STATUS = Backed Out

Figure 5.3-12. Back Out Example 1



Committed Sequence	Backed Out Sequence 1	Backed Out Sequence 2
1. A -> B - Prepare	1. A -> B - Prepare	1. A -> B - Prepare
2. A -> C - Prepare	2. A -> C - Prepare	2. A -> C - Prepare
3. B -> A - Request Commit	3. B -> A - Request Commit	3. B -> A - Request Commit
4. C -> A - Request Commit	4. C -> A - Backed Out	4. C -> A - Request Commit
5. A -> D - Request Commit	5. A -> B - Backed Out	5. A -> D - Request Commit
6. D -> A - Committed	6. A -> D - Backed Out	6. D -> A - Backed Out
7. A -> B - Committed		7. A -> B - Backed Out
8. A -> C - Committed		8. A -> C - Backed Out
STATUS = Committed	STATUS = Backed Out	STATUS = Backed Out

Figure 5.3-13. Back Out Example 2

HEURISTIC DECISIONS AND RELIABLE RESOURCES

Each implementation of the sync point option set makes available to transaction programs at least one protected resource that is fully reliable in that it is not subject to heuristic decisions. This can be done in a variety of ways; the simplest is to allow application designers to designate certain resources as not subject to heuristic decisions. However the reliable resource is provided, application designers can use data kept in the reliable resource to aid in recovery from any heuristic mismatches that may occur.

RESYNCHRONIZATION LOGIC

Resynchronization logic involves these steps:

- If an IPL has occurred, RM retrieves log records from the log manager and reconstructs the protected TCBs and RCBs that were active at the time of the failure. It then causes PS.SPS to gain control on the reconstructed TCB. PS.SPS uses the log to restore its relevant states. For instance, it restores the initiator/agent state for each resource. PS.SPS also supplies log records to the protection managers for each resource so that they can back out their resources if this is required.

- When PS.SPS finishes resynchronizing, RM deallocates the TCB.

- If the resync is occurring without an IPL, PS.SPS will return control to the TP or to the abnormal termination processor, depending on the caller. The abnormal termination processor, of course, will deallocate all resources as needed.

- Since it can happen that multiple conversations connect TCBs with the same LUWIDs in two separate LUs, resynchronization uses the value in the Conversation Correlator field carried in Attach (see SNA Formats) to uniquely identify the LUW whose states are to be compared. For example, this case occurs when TPa at LUa allocates a conversation with TPb at LUb. Then, as part of the same LUW, TPb allocates a conversation with TPc at LUa. The conversation correlator provides a way for PS.SPS at LUa to distinguish the part of the LUW that LUa initiated from the part that LUb initiated. The conversation correlator is unique in a network. To provide uniqueness, the fully qualified LU name of the LU that created the conversation correlator is concatenated to the conversation correlator when comparisons are made. The fully qualified LU name of the partner LU is known from the system definition of the PARTNER_LU data structure.

The decision to initiate resync by either end is depends upon the state of the unit of work. The following table reflects the action PS.SPS takes after a conversation failure or an IPL of the LU.

UNIT-OF-WORK STATE (in local log)	ACTION BY PS.SPS
Not Found.........	No Action
Agent, not last ...	Wait for resync
Agent, last	Resync after time-out
Initiator	Initiate resync

VALIDATION OF LOG IDS

The first level of resynchronization is the validation of the log IDs. PS.SPS accomplishes this by exchanging Log ID GDS variables. When this exchange validates the integrity of the LU pair's logs, PS.SPS exchanges Compare States. The following figures illustrate this resync logic.

```
 TP    PS.                                             PS.
       SPS                                             SPS

(1) SYNCPT
    ------->
           .
           .
           <--------Session Outage Notification

      ALLOCATE    same LUWID as the LUW that is being resynchronized
                  SYNC_LEVEL(CONFIRM),
                  TPN(X'06F2') ... ;

               [  BIND                                 ]
               [ ------------------------------------> ]  (optional)

                  Attach
                 ----------------------------------------------------->
     SEND_DATA    Exchange Log Name, log status (warm), log name (log name)
                 -----------------------------------------------------> RECEIVE_AND_WAIT
     SEND_DATA    Compare States command, CD                            (2)
                 -----------------------------------------------------> RECEIVE_AND_WAIT
                  Exchange Log Name, log status (warm), log name (log name) SEND_DATA
RECEIVE_AND_WAIT <-----------------------------------------------------
             (2)  Compare States reply, CD                              SEND_DATA
RECEIVE_AND_WAIT <-----------------------------------------------------

DEALLOCATE
  TYPE(SYNC_LEVEL) LUSTAT(X'0006'), RQD2, CEB
                 -----------------------------------------------------> RECEIVE_AND_WAIT
                                                                        (3)
             (3)  +DR2                                                  CONFIRMED
                 <-----------------------------------------------------
```

Figure 5.3-14. Resync after Conversation Failure

The following notes correspond to the numbers in Figure 5.3-14.

1. The TP issues SYNCPT or BACKOUT, giving PS.SPS control. Conversation failure results from the session outage. PS.SPS detects this and begins resynchronization by issuing ALLOCATE specifying the resync transaction program, X'06F2', as the TPN. The optional BIND may flow between LUs as a result of RM logic; RM will send BIND to activate a new session if an existing session is not available; PS.SPS does not know if it flows. PS.SPS retrieves the LUWID carried in this Attach from the TCB.

2. PS.SPS validates the log name and then executes resync logic. Each conversation to be resynchronized is processed in a separate resync conversation using a separate copy of the resync TP.

3. PS.SPS tells the log manager to erase the LUW's log records. The half-session sends an LUSTAT because there is no data to send. The LUSTAT carries the RH. PS.SPS is not aware of this detail.

```
 ┌────┐  ┌─────┐                                                 ┌─────┐
 │ RM │  │ PS. │                                                 │ PS. │
 │    │  │ SPS │                                                 │ SPS │
 └────┘  └─────┘                                                 └─────┘

(1)  LU fails
       .
       .
       .

     ALLOCATE    same LUWID as the LUW that is being resynchronized
                 SYNC_LEVEL(CONFIRM),
                 TPN(X'06F2') ... ;

                 BIND
                ---------------------------------------------------->

                 Attach
                ---------------------------------------------------->
     SEND_DATA   Exchange Log Name, log status (warm), log name (log name)
                ----------------------------------------------------> RECEIVE_AND_WAIT
     SEND_DATA   Compare States command, CD                            (2)
                ----------------------------------------------------> RECEIVE_AND_WAIT
                 Exchange Log Name, log status (warm), log name (log name) SEND_DATA
RECEIVE_AND_WAIT <----------------------------------------------------
          (2)    Compare States reply, CD                               SEND_DATA
RECEIVE_AND_WAIT <----------------------------------------------------

DEALLOCATE
TYPE(SYNC_LEVEL) LUSTAT(X'0006'), RQD2, CEB
                ----------------------------------------------------> RECEIVE_AND_WAIT
                                                                      (3)
          (3)    +DR2                                                  CONFIRMED
                <----------------------------------------------------
```

Figure 5.3-15. Resync after LU Failure

The following notes correspond to the numbers in Figure 5.3-15.

1. The LU fails. After the LU is IPLed, RM reads the sync point records from the log, rebuilds the TCB and RCBs, and gives PS.SPS control. After re-establishing the states of the local protected resources in cooperation with their protection managers, PS.SPS proceeds to resync each LUW in a separate conversation, since the reply can be delayed while cascaded resync occurs. If all the resync conversations are processed in parallel, multiple sessions will be used--up to one per LUW to be resynchronized. This can cause as many BINDs as LUWs that are in resynchronization. A UPM determines the degree of parallelism. The more parallelism, the more session resources will be used, but the resync may complete faster.

2. PS.SPS validates the log name and then executes resync logic.

3. PS.SPS erases the log. If a conversation or LU failure occurs during resynchronization, PS.SPS repeats resynchronization until both logs are erased.

SESSION OUTAGE DURING ATTACH

If session outage occurs, the Compare States command that is part of resync can arrive ahead of the session outage notification. When this occurs and the last-resource optimization is being used, and if no special steps were taken, the result could be that one partner backs out and the other partner commits. The resolution of this race condition is depicted in Figure 5.3-16 on page 5.3-21.

```
 TP     PS.SPS(1)                                     PS.SPS(2)     PC|LNS|RM

 ALLOCATE                                                                  (1)
 SEND_DATA    *BB, Attach, data, Request Commit, CD
 SYNCPT      ------------------------------------------+                   (2)
                                                       |
                          SON        SON               |                   (3)
             <--------------------X-----------+        |
             BB, Compare States(Session ID)   |        |
             ---------------------------------|--------|------->           (4)
                                              |        |   DEACTIVATE_SESSION
                                              |        |     TYPE(CLEANUP)
                       UNBIND(cleanup)        |        |     SESSION_ID(Session ID)
             <--------------------------------|--------|-------
                                              |        +---------------> (purged) (5)
                                              +------------------------> (purged)
                          Not Found
             <------------------------------------------------------               (6)
```

NOTES:

This shows how the failure is prevented by deactivating the session prior to processing the Compare States command.

PS.SPS(1) is the PS.SPS instance that is running on behalf of the application TP. PS.SPS(2) is the PS.SPS instance that is processing the Compare States command. It is using a different session from that used by the application TP.

Internal flows are not shown.

Figure 5.3-16. Avoiding Failure Resulting from an Attach-SON Race

The comments below correspond to the numbers in Figure 5.3-16.

1. A transaction is allocated with a sync level of sync point.

2. Data is sent and a sync point (in this case, for an optimized last resource) is requested.

3. After the data and Request Commit are sent, a session outage occurs. Both sides of the conversation are informed of the outage.

4. When the sync point manager receives the outage indication, it sends a Compare States command (Exchange Log Name also flows but it is not shown in the diagram). This flows on a different session from the transaction program data. This session can use any mode. However, the performance characteristics of the mode should be good enough to avoid undue delays in resynchronization. As a result of using a different session, the Compare States command can arrive ahead of the TP data.

   To resolve this race condition, the receiving sync point manager, PS.SPS(2), issues a DEACTIVATE_SESSION TYPE(CLEANUP) whenever Compare States is received. The Session Instance Identifier field of the Compare States command has the session identifier, to allow the sync point manager to deactivate the affected session. When the session deactivation is complete, any Attaches that are in transit are discarded and RM purges any records received from that half-session. Then the Compare States processing can proceed. If the Attach has been processed and the attached TP executed before the deactivation is complete, a log entry for the LUW will be found. If the Attach is discarded, no log entry will be found. In either case, both data bases will remain synchronized.

5. Depending upon when the Attach and SON arrive, either path control, RM or LNS purges them because the session has been deactivated.

6. The receiving sync point manager, PS.SPS(2), checks the log for awareness of the LUW for which the Compare States was sent. Since the Attach has not arrived yet, or it was purged, no log entry exists. The reply to Compare States is therefore Not Found.

It is possible that the incoming Attach has been processed and the PS process for the application TP was created, but the TP has never been dispatched when the Compare States arrives. In this case, the Attach arrives ahead of the Compare States, but as a result of timing conditions in the node, the Compare States TP (shown above as PS.SPS(2)) executes before the attached application TP. Then, before the application TP runs and the LUW is logged, the half-session is deactivated because of the Compare States processing. When the DEACTIVATE_SESSION processing is completed, PS.SPS(2) checks the log to find the state of the LUW. The LUW is not logged yet, so the reply to the Compare States is Not Found. In order to avoid having the application TP execute later and commit the LUW, the sync point manager that is running on behalf of an application TP checks that the LUW was not previously backed out, because of a session deactivation, before it commits. This is accomplished by having RM inform the sync point manager that the half-session it is using was deactivated. The sync point manager cannot perform a Commit if it is informed of a session deactivation.

LOST SYNC POINT MESSAGES

The logic for resync is summarized in Figure 5.3-21 on page 5.3-26 through Figure 5.3-25 on page 5.3-30. This logic is derived from Figure 5.3-19 on page 5.3-24, which shows the sync point messages that can be lost because of session outage, as viewed by the initiator. For example, when a Prepare is lost because of SON, the state of the LUW at the sender of the Prepare (the initiator), is either SPM PENDING or HEURISTIC RESET; the state of the LUW at the receiver (the agent) is either RESET or PGM PENDING. In the case when SEND_ERROR and Prepare are lost from one TP and Prepare is lost from the partner, the state of LUW on either side of the conversation is SPM PENDING or HEURISTIC RESET. Given this, it is possible to construct the tables that guide the resynchronization actions that the sync point managers must take.

Prepare vs. Prepare races (when both sides issue Prepare, and the flows cross), Prepare vs. Request Commit races, and Request Commit vs. Request Commit races also can occur as a result of races between session outage notification and SEND_ERRORs.

An example is shown in Figure 5.3-17.

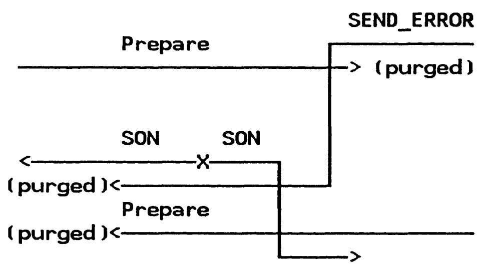


Figure 5.3-17. SEND_ERROR and Prepare vs. Prepare Race during Session Outage

When one TP issues SEND_ERROR followed by SYNCPT and messages are lost because of session outage, it is possible that both partners are the sync point initiator. In this case, each reports its state on the Compare States reply as if it were an agent.

The one exception to this rule is in the case shown in Figure 5.3-18 on page 5.3-23. In this case, when resynchronizing, the original sender of Prepare (sync point services[1]) recognizes that the partner is resynchronizing (following SON, the partner (sync point services[2]) sent a Compare States command with a state indicator of IN DOUBT). The sender of Prepare then replies with a RESET state indicator on the Compare States reply.

The details of resync, based on the state of the LUW is shown in the matrix in Figure 5.3-19 on page 5.3-24 and the logic depicted in Figure 5.3-23 and Figure 5.3-24.

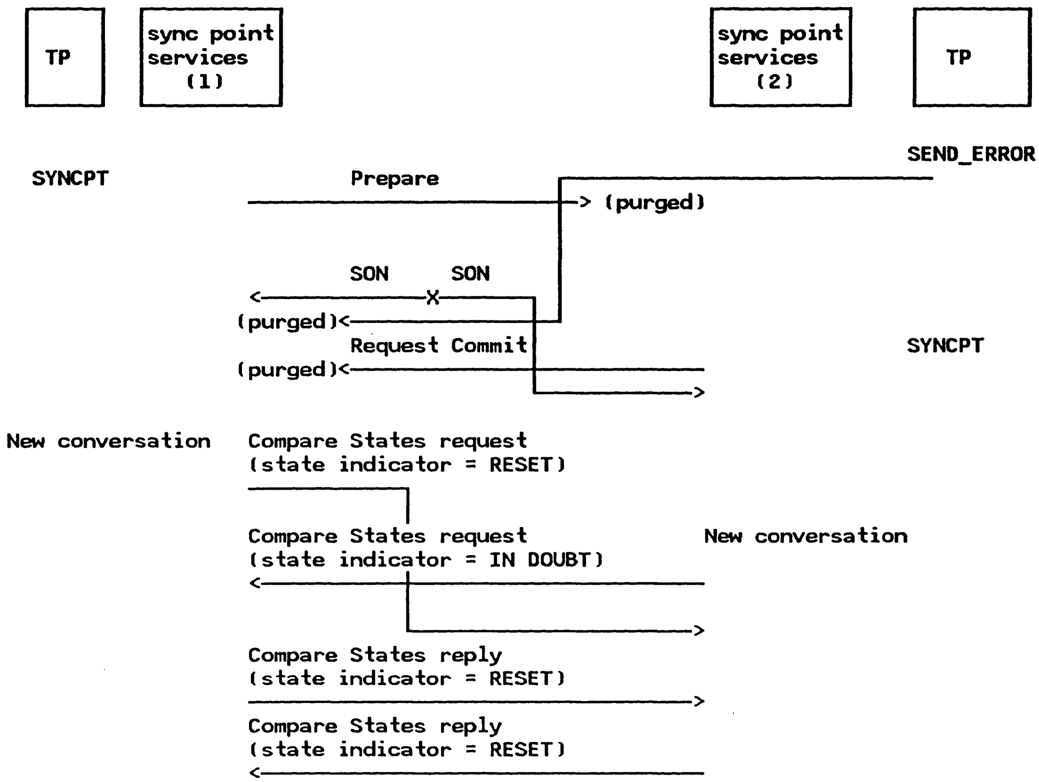


Figure 5.3-18. SEND_ERROR and Request Commit vs. Prepare Race during Session Outage

Sync Point Message Lost By Session Outage	Initiator's State When It Initiates Resync	Agent's State When Resync Occurs
Prepare [3]	SPM PENDING | HEURISTIC RESET [1]	RESET | PGM PENDING [2]
Prepare vs. SEND_ERROR and Prepare [5]	SPM PENDING | HEURISTIC RESET [1]	SPM PENDING | HEURISTIC RESET [1]
Prepare vs. SEND_ERROR and Request Commit(last) [5]	SPM PENDING | HEURISTIC RESET [1]	IN DOUBT | HEURISTIC RESET [4] | HEURISTIC COMMITTED [4]
Request Commit	SPM PENDING | HEURISTIC RESET [1]	IN DOUBT | HEURISTIC RESET [4] | HEURISTIC COMMITTED [4]
Committed	COMMITTED	IN DOUBT | HEURISTIC RESET [4] HEURISTIC COMMITTED [4]
Forget	COMMITTED	RESET
Request Commit(last) [3]	IN DOUBT | HEURISTIC RESET [4] | HEURISTIC COMMITTED [4]	RESET | PGM PENDING [2]
Request Commit(last) vs. SEND_ERROR and Prepare [5]	IN DOUBT | HEURISTIC RESET [4] | HEURISTIC COMMITTED [4]	SPM PENDING | HEURISTIC RESET [1]
Request Commit(last) vs. SEND_ERROR and Request Commit(last)	IN DOUBT | HEURISTIC RESET [4] | HEURISTIC COMMITTED [4]	IN DOUBT | HEURISTIC RESET [4] | HEURISTIC COMMITTED [4]
Committed(last)	IN DOUBT | HEURISTIC RESET [4] | HEURISTIC COMMITTED [4]	COMMITTED

Notes:

1. The LUW has been backed out as a result of a heuristic request from the lock manager. The initiator still owes resync to the agents.
2. The PGM PENDING state is never visible during resync, since it is converted to RESET.
3. These rows assume that no Prepare vs. Prepare type races have occurred.
4. Either HEURISTIC RESET or HEURISTIC COMMITTED can occur from IN DOUBT.
5. The agent issues SEND_ERROR and a sync point message, making the agent also an initiator for this race.

Figure 5.3-19. Lost Sync Point Messages: Initiator's View

Lost Message	Last Agent's State when it sends Compare States command	State of receiver of Compare States command
Implied Forget	COMMITTED [Note]	RESET | COMMITTED | HEURISTIC COMMITTED

Note: The last agent does not have to send this Compare States command unless it needs to erase its log. The last agent does have to remember the COMMITTED state if it does not receive Forget, either implied or as part of the initiator's resync or following its own resync. (See Figure 5.3-25 on page 5.3-30 for further details.) It is not possible to eliminate entirely this responsibility of the last agent without coupling the sync point resync to a single session instance. If the sync point resync were restricted to a single session instance, session data on that session could be treated as an Implied Forget.

Figure 5.3-20. Lost Messages for Sync Point: Last Agent's View

RESYNCHRONIZATION ACTION

Figure 5.3-21 on page 5.3-26 through Figure 5.3-25 on page 5.3-30 show the indicated states that the sync point manager is either sending or receiving in the Compare States command, in relation to the action taken. The logic depicted in these diagrams is the logic needed to implement resynchronization when an IPL has taken place or after an LU or a session fails.

The top left-hand corner of the tables indicates the role the sync point manager has in the sync point exchange. In Figure 5.3-21 on page 5.3-26, Figure 5.3-23 on page 5.3-28, and Figure 5.3-25 on page 5.3-30 the left-hand column shows the state indication that is sent on the Compare States command. The top row shows the state that is indicated in the Compare States reply. In Figure 5.3-22 on page 5.3-27 and Figure 5.3-24 on page 5.3-29, the left-hand column shows the state that is indicated in the Compare States command. The top row indicates the state of the LUW at the receiver; this state is indicated in the returned Compare States reply.

The matrix entry for the column-row pair indicates the action to be taken based on the pair of state indications exchanged. The action to be taken includes changing the state of the LUW at the LU and/or sending a message to the control operator. (See "Resynchronization Operator Messages" on page 5.3-30 for a description of the messages sent to the LU control operator.) For example, in Figure 5.3-21 on page 5.3-26, if the initiator finds the state of an LUW to be IN DOUBT on its log, it sends a Compare States command to the last agent, with the state indicator in the command set to IN DOUBT. If the initiator receives a Compare States reply with the state indicator set to RESET, it backs out the LUW. If the Compare States reply indicates COMMITTED, it commits the LUW. If the Compare States reply indicates HEURISTIC MIXED, it either commits or resets the LUW, based on a heuristic decision. The heuristic decision taken depends on the transaction program's defined characteristics. When the heuristic decision is taken, the LU control operator receives message 3.

Figure 5.3-21 on page 5.3-26 and Figure 5.3-22 on page 5.3-27 show the actions when resynchronizing with the last agent. The last agent must be resynchronized before any not-last agents are resynchronized because the state indication the last agent returns on the Compare States command controls the state indication sent to any not-last agents. When a Request Commit is sent to the last agent, the state of the LUW at the initiator with respect to the last agent is IN DOUBT. No other resynchronization is possible until it is known whether the state of the LUW at the last agent is RESET or COMMITTED.

\ INITIATOR \ RECEIVES \ \ INITIATOR\ SENDS \ \	RESET | LOG ENTRY NOT FOUND 1	SPM PENDING [3] 2	IN DOUBT [3] 3	COMMITTED 4	HEURISTIC RESET [3] 5	HEURISTIC COMMITTED [3] 6	HEURISTIC MIXED 7
RESET [3] 1	......[1]	....	....	....	....	....	....
SPM PENDING [3] 2	....	....	....	....	....	....	....
IN DOUBT 3	BACKOUT	....	....	COMMIT	....	....	HR or HC [2] and MSG 3
COMMITTED [3] 4	....	....	....	....	....	....	....
HEURISTIC RESET 5	MSG 2	....	....	MSG 3	....	....	MSG 3
HEURISTIC COMMITTED 6	MSG 3	....	....	MSG 2	....	....	MSG 3
HEURISTIC MIXED [3] 7	....	....	....	....	....	....	....

Notes:

1. All intersections with dots should not occur. States 1, 2, 4, and 7 are not possible at the initiator. Message 4 is generated for the control operator if indications of states 2, 3, 5, or 6 are returned by the last agent on the Compare States reply.

2. The heuristic direction is taken from the TP definition table. The TP definition table is created when the TP is defined to the LU. The HR (HEURISTIC RESET) or HC (HEURISTIC COMMITTED) action applies to local resources and cascaded resources. HEURISTIC MIXED (HM) state is reported to the resync initiator on the Compare States reply. In the case where the resync initiator was a cascaded agent, HM is reported to the sync point initiator in a PS header (Heuristic Mixed)

3. These state indications should never occur.

Figure 5.3-21. Resynchronization Action: At Initiator, When Resynchronizing with the Last Agent

\ LAST \ AGENT LAST \SENDS AGENT \ RECEIVES \	RESET | LOG ENTRY NOT FOUND 1	SPM PENDING [2] 2	IN DOUBT [2] 3	COMMITTED 4	HEURISTIC RESET [5] 5	HEURISTIC COMMITTED [5] 6	HEURISTIC MIXED 7
RESET [5] 1	......[1]	....	....	....	....	....	....
SPM PENDING [5] 2	....	....	....	....	....	....	....
IN DOUBT 3	—— [3]	....	....	—— [3]	....	....	MSG 3
COMMITTED [5] 4	....	....	....	....	....	....	....
HEURISTIC RESET 5	—— [3]	....	....	MSG 3 [4]	....	....	MSG 3
HEURISTIC COMMITTED 6	MSG 3 [4]	....	....	—— [3]	....	....	MSG 3
HEURISTIC MIXED [5] 7	....	....	....	....	....	....	....

Notes:

1. All intersections with dots should not occur. States 1, 2, 4, and 7 are not possible at the initiator. Message 4 is generated if indications of states 2, 3, 5, or 6 are returned by the last agent.

2. If resync occurs while the last agent is still in SPM PENDING or IN DOUBT state, the agent defers sending the reply until it completes its cascaded protocol, which may include resynchronization with cascaded agents. Eventually the last agent's state will resolve to RESET, COMMITTED, or HEURISTIC MIXED. The HEURISTIC MIXED state is reported when SON occurs on sessions with at least two cascaded agents and the control operator at one of the LUs causes the LUW to be put into a HEURISTIC RESET state, while the operator at the other LU causes the LUW to be put into a HEURISTIC COMMIT state. If there are no cascaded resources, the last agent changes the state of the LUW to reflect the state reported on the Compare States request.

3. In these cases, the agent takes no action, except to erase its log (if the log entry was found) upon the completion of the resync flows. The end of the resync flows is defined by receipt of LUSTAT(X'0006'), RQD2, CEB from the initiator, or receipt of +RSP(LUSTAT) from the agent. See Figure 5.3-14 on page 5.3-19 and Figure 5.3-15 on page 5.3-20 for more details.

4. A HEURISTIC MIXED situation (described in Note 2) has been detected and is reported to the resync initiator on the Compare States reply. The HEURISTIC MIXED state indicator is not propagated to the cascaded agents. See Figure 5.3-23 on page 5.3-28 for this case.

5. These state indications should never occur.

Figure 5.3-22. Resynchronization Action: At Last Agent, When Resynchronizing with the Initiator

\ INITIATOR \ RECEIVES \ \ \ INITIATOR \ SENDS \	RESET | LOG ENTRY NOT FOUND 1	SPM PENDING [2] 2	IN-DOUBT [2] 3	COMMITTED 4	HEURISTIC RESET 5	HEURISTIC COMMITTED 6	HEURISTIC MIXED [4] 7
RESET 1	—— [3]	....	....	....	—— [3]	MSG 3	MSG 3
SPM PENDING [5] 2	..... [1]	....	....	....	....	....	....
IN DOUBT [6] 3	....	....	....	....	....	....	....
COMMITTED 4	....	....	....	—— [3]	MSG 3	—— [3]	MSG 3
HEURISTIC RESET 5	MSG 2	....	....	....	MSG 2	MSG 3	MSG 3
HEURISTIC COMMITTED 6	MSG 3	....	....	MSG 2	MSG 3	MSG 2	MSG 3
HEURISTIC MIXED [6] 7	....	....	....	....	....	....	....

Notes:

1. All intersections with dots should not occur. States 2, 3, and 7 are not possible at the initiator. Message 4 is generated if an indication of state 2 or 3 is returned by the agent.

2. If resync occurs while the not-last agent is still in SPM PENDING or IN DOUBT state, the agent defers sending the reply until it completes its cascaded protocol, which may include resynchronization with cascaded agents. Eventually the not-last agent's state will resolve to RESET, COMMITTED, or HEURISTIC MIXED. The HEURISTIC MIXED state is reported when SON occurs on sessions with at least two cascaded agents and the control operator at one of the LUs causes the LUW to be put into a HEURISTIC RESET state, while the operator at the other LU causes the LUW to be put into a HEURISTIC COMMIT state. If there are no cascaded resources, the last agent changes the state of the LUW to reflect the state reported on the Compare States request.

3. In these cases, the agent takes no action, except to erase its log (if the log entry was found) upon the completion of the resync flows.

4. In all the HEURISTIC MIXED (HM) cases displayed in this matrix, while HEURISTIC MIXED is reported as a return code on the SYNCPT verb, HM is not propagated to agents. Rather, they are told the initial state of the initiator so that Message 3 is not issued for those agents that are synchronized with the initiator. This is illustrated by the following case: the initiator of resynchronization reports COMMITTED on the Compare States command, the agent has three protected cascaded conversations. The agent finds SPM PENDING on its log, so it initiates resynchronization with its cascaded agents. One of the cascaded agents reports HEURISTIC COMMITTED and the other reports HEURISTIC RESET on the Compare States reply. Rather than send a Compare States command indicating HEURISTIC MIXED to the third protected conversation, COMMITTED is reported.

5. When the initiator is in SPM PENDING state, it has resync responsibility. However, it reports its state as RESET on the Compare States command. The SPM PENDING state indicates that a resync is expected. If the state is actually RESET, no resync is needed.

6. These state indications should never occur.

Figure 5.3-23. Resynchronization Action: At Initiator, When Resynchronizing with the Not-Last Agent

\ ¬LAST \ AGENT ¬LAST\SENDS AGENT \ RECEIVES \	RESET (NOT FOUND) 1	SPM PENDING [2] 2	IN DOUBT [2] 3	COMMITTED 4	HEURISTIC RESET 5	HEURISTIC COMMITTED 6	HEURISTIC MIXED 7
RESET 1	—— [3]	....	....	....	MSG 2	MSG 3	MSG 3
SPM PENDING [4] 2	......[1]	....	....	....	....	....	....
IN DOUBT [4] 3	....	....	....	....	....	....	....
COMMITTED 4	....	....	....	—— [3]	MSG 3	MSG 2	MSG 3
HEURISTIC RESET 5	—— [3]	....	....	....	MSG 2	MSG 3	MSG 3
HEURISTIC COMMITTED 6	....	....	....	—— [3]	MSG 3	MSG 2	MSG 3
HEURISTIC MIXED [4] 7	....	....	....	....	....	....	....

Notes:

1. All intersections with dots should not occur. States 2, 3, and 7 are not possible at the initiator. Message 4 is generated if they are received.

2. If resync occurs while the agent is still in SPM PENDING or IN DOUBT state, the agent defers sending the reply until it completes its cascaded protocol, which may include resynchronization with cascaded agents. Eventually the not-last agent's state will resolve from SPM PENDING to RESET or HEURISTIC MIXED; or IN DOUBT will resolve to RESET, COMMITTED, or HEURISTIC MIXED. If there are no cascaded resources, the last agent changes the state of the LUW to reflect the state reported on the Compare States request.

3. In these cases, the agent takes no action, except to erase its log upon the completion of the resync flows.

4. These state indications should never occur.

Figure 5.3-24. Resynchronization Action: At Not-Last Agent, When Resynchronizing with the Initiator

\ LAST \ AGENT LAST \ RCVS AGENT \ SENDS \	RESET | LOG ENTRY NOT FOUND 1	SPM PENDING[2] 2	IN DOUBT [2] 3	COMMITTED [2] 4	HEURISTIC RESET[2] 5	HEURISTIC COMMITTED [2] 6	HEURISTIC MIXED [2] 7
RESET [3] 1	ERASE LOG [1]	NO ACTION	NO ACTION	NO ACTION	NO ACTION	NO ACTION	NO ACTION

Notes:

1. The last agent erases its log upon receipt of the initiator's state indication.
2. If the initiator is in any state other than RESET, both the last agent and the initiator ignore the state exchange that started from the last agent. It was started, at the last agent, by the sending of a Compare States command. The state indication exchange started by the initiator will perform the actual resynchronization.
3. The last agent actually finds COMMITTED on its log. In this case, it is possible that the initiator has already sent the implied Forget, but the implied Forget was lost. As a result, the initiator has no log record that indicates that the initiator is responsible for the resync. After a period of time, if the initiator has not attempted to resync, the last agent takes responsibility. It sends RESET. If the initiator replies with any state indication other than RESET, it means there has been a resync race. The initiator has resync responsibility and the resync is forthcoming.

Figure 5.3-25. Resynchronization Action: Resync from Last Agent

RESYNCHRONIZATION OPERATOR MESSAGES

The sync point manager issues several messages to the LU control operator. Messages can be sent to the operator at the following times: following session outage, during the log name exchange, and during the resync exchange. The parameters shown in parenthesis (such as TPN, for transaction program name) are passed in the message. In the messages that follow, reference is made to a 'waiting unit of work'. A waiting unit of work is an LUW for which resynchronization is necessary. Until the resynchronization is complete, resources may be locked and unavailable for other computations. The order that the messages may occur in are shown in Figure 5.3-26 on page 5.3-31.

- MSG 1: A heuristic decision has been made for transaction program name (TPN), process ID (PID), at time (TIME), and logical unit of work ID (LUWID). As a result, resources in LUs (LU name-1, LU name-2, ... , LU name-X) may be in inconsistent states with respect to the local resource states.

  Operator Action: Take user-defined action, if any, to protect data integrity until the local and remote data can be synchronized.

- MSG 2: Resources in LUs (LU name-1, ... , LU name-X), previously reported to be exposed to state inconsistency with respect to local resources for transaction program name (TPN), process ID (PID), at time (TIME), logical unit of work ID (LUWID), have been found to be synchronized.

  Operator Action: Reverse the user-defined action taken when MSG 1 was acted upon.

- MSG 3: Resources in LUs (LU name1, ... , LU nameX), previously reported to be exposed to inconsistency with respect to local resources for transaction program name (TPN), process ID (PID), at time (TIME), logical unit of work ID (LUWID), have been found to be out of synchronization.

  Operator Action: Take user-defined action to resynchronize the local and remote resources.

- MSG 4: Protocol failure in resynchronization logic during attempted resynchronization of transaction program name (TPN), process ID (PID), at time (TIME), logical unit of work ID (LUWID), conversation correlator (CID). The local state was state (state name), the remote state was state (state name).

  Operator Action: Make inquiries to determine the state of the resources. Take user-defined action to resynchronize the resources. Submit APAR report.

- MSG 5: Session failure (with LU name-i, mode name-j) at time (TIME), has resulted in a waiting unit of work for transaction program name (TPN), process ID (PID), with logical unit of work ID (LUWID) and conversation correlator (CID). This LUW is coupled to other LUs (LU name-m,..., LU name-n).

  Operator Action: Reactivate the session as soon as possible.

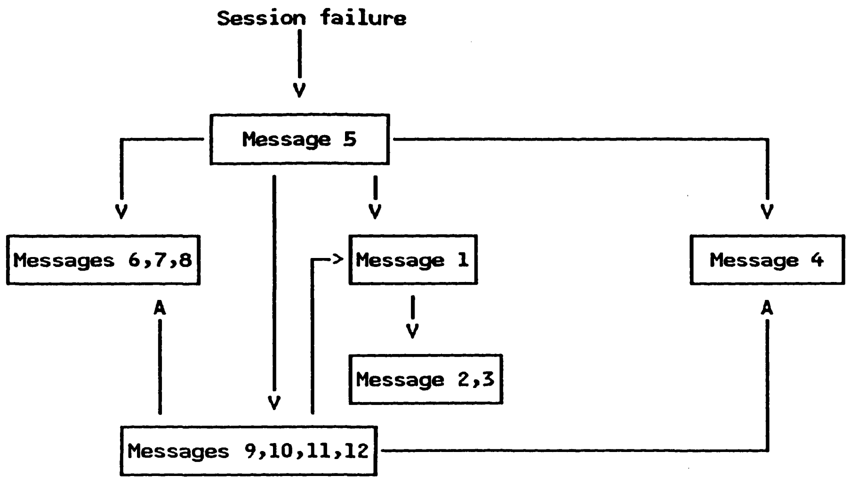


Figure 5.3-26. The Sequence of LU Control Operator Messages Generated by Sync Point Resynchronization

- MSG 6: The waiting unit of work identified by transaction program name (TPN), process ID (PID), and logical unit of work ID (LUWID), reported at time (TIME), is being commited.

  Operator Action: none.

- MSG 7: The waiting unit of work identified by transaction program name (TPN), process ID (PID), and logical unit of work ID (LUWID), reported at time (TIME), is being backed out.

  Operator Action: none.

- MSG 8: Resources in LUs (LU name-1, ..., LU name-X), previously reported to be waiting for resynchronization with respect to local resources for transaction program name (TPN), process ID (PID), at time (TIME), logical unit of work ID (LUWID), have been found to be out of synchronization.

  Operator Action: Take user-defined action to resynchronize the local and remote resources.

- MSG 9: The logical unit of work identified by transaction program name (TPN), process ID (PID), at time (TIME), logical unit of work ID (LUWID), has been waiting for HHH hours and MMM minutes. Locks held by this resynchronization are enqueued by NN transactions.

  Operator Action: If desired, abnormally terminate the PID specified process. This will release locks and may result in heuristic mismatches when resynchronization does complete.

- MSG 10: LU (LU name) has returned an abnormal reply to the Exchange Log Name command. This LU has detected a warm/cold mismatch, or a log name mismatch.

  Operator Action: Coordinate activation with the operator at the other LU. It may be necessary to abnormally terminate processes for some waiting units of work.

- MSG 11: A cold start has been attempted by LU (LU name), but the local LU has logical units of work that are awaiting resynchronization from the previous activation.

  Operator Action: Coordinate activation with the operator at the other LU. It may be necessary to abnormally terminate processes for the waiting units of work.

- MSG 12: LU (LU name) does not have the same memory as does the local LU of the previous activation between them.

  Operator Action: Coordinate activation with the operator at the other LU. It may be necessary to abnormally terminate processes for the waiting units of work.

ORDER OF RESYNCHRONIZATION

When a distributed unit of work fails because of a session or LU failure, more than one resynchronization exchange may be needed before resynchronization is complete. Figure 5.3-27 on page 5.3-32illustrates what can happen.

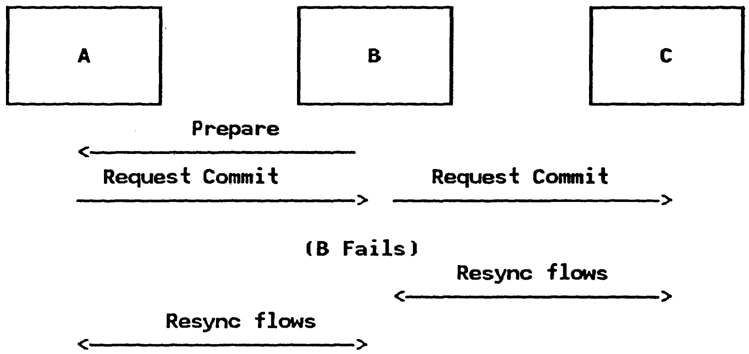


Figure 5.3-27. Cascaded Resynchronization Example

The rule illustrated in the figure is the following: The initiator resolves in-doubt resources before it resynchronizes with the other resources involved in the logical unit of work. One result can be that some participants in the distributed LUW will make heuristic decisions.

ERRORS AND FAILURES DURING RESYNCHRONIZATION

Errors and additional failures can occur during attempted resynchronization. Repeated conversation failures are handled by the resynchronization logic, since log records are not erased until after the state indications have been exchanged; see Figure 5.3-14 on page 5.3-19 and Figure 5.3-15 on page 5.3-20 for examples. Errors that occur while the sync point manager has control, such as completion of the receive timer that is started when resynchronization begins, are mapped into conversation failures, created by UNBIND, thereby falling back into an error recovery loop (described below).

The conversation failures, created by UNBIND, to recover from errors detected while the sync point manager has control are UNBIND(X'FE08640002' | X'FE08640001') depending on the error—timer or logic error, respectively—rather than DEALLOCATE(ABEND), since the latter is not guaranteed to work under double failures (the receiver of DEALLOCATE has to continue to issue RECEIVE in order for it to work).

The error recovery process is described as a loop because it iterates until one of the loop exit conditions is satisfied. The error recovery loop has two exits: Either (1) the resync completes, or (2) the control operator, after being informed that the resync attempt has been going on for a long time (the resync timer completes), decides to abnormally terminate processes that are holding locks for this LUW rather than continue with the sync point manager resync attempts. The cleanup transaction will erase the log record after writing suitable messages to the error log. It may additionally force (unilaterally change, without agreement among partners) the states of any pending resources, to HEURISTIC RESET or HEURISTIC COMMITTED, in the same way that heuristic decisions may force states.

RESET STATE AND ERASING OF LOG RECORDS

Reset state of the LUW (equivalent to unit of work backed out) is denoted by the absence of a log record.

The initiator's side can erase its log when all Forget flows have been completed, because, at this point, it is known that resync will never be required. Therefore the question of ambiguity of no record found never arises.[9]

The slaves erase their logs before sending Forget, so that a subsequent failure that results in the loss of the Forget will result in a resync that finds no log record.

LOG NAME PROCESSING

The following two figures illustrate the processing of log names so that log mismatches do not occur. A log mismatch occurs if the LU control operator mounts the wrong

[9] The ambiguity is that not finding a log record could mean that the LUW has either not been logged yet (not started), or is committed (completely finished).

sync-point-log tape or instructs the LU to use the wrong log-dataset, or the LU IPLs and no log exists (this is referred to as a cold start). When this happens, the sync-point log is not available for resync processing. The Exchange Log Name command is sent before the Compare States command, to detect a log mismatch.

```
 RM    PS.                                          PS.
       SPS                                          SPS

(1)   LU IPLs Cold
        .
        .
        .
       ALLOCATE    new LUWID,
                   SYNC_LEVEL(CONFIRM),
                   TPN(X'06F2') ...

          BIND
          ---------------------------------------->

          Attach
          ---------------------------------------->
(2)       Exchange Log Name, log status(cold), CD
          ---------------------------------------->
(3)       Exchange Log Name, log status(cold), RQD2, CEB
          <----------------------------------------
(4)       +DR2
          ---------------------------------------->
```

Figure 5.3-28. Cold Start of an LU

The following notes correspond to the numbers in Figure 5.3-28.

1. The LU IPLs cold, that is, with a new log tape or new log dataset. No resync attempt occurs, since the log is empty. If the name of the LU's log is changed, a cold IPL is required.

   PS.SPS is given control before any conversations with SYNC_LEVEL(SYNCPT) are allocated in order to exchange log names with PS.SPS in the partner LU. The sync point manager needs to know the partner's log name so log mismatches can be detected during resynchronization.

2. The resync TP, X'06F2', accepts the cold log name and returns its own LU's log name. Message 11 might also be returned on an error reply, as shown in Figure 5.3-29 on page 5.3-34.

3. Upon logging the log name of the partner PS.SPS, PS.SPS tells RM that SYNC_LEVEL(SYNCPT) conversations can now be allocated to the partner LU.

4. The partner PS.SPS similarly informs its RM. Race conditions can cause this transaction to be executed twice, once in each direction.

```
  RM     PS.                                                   PS.
         SPS                                                   SPS

(1)    LU IPLs Warm, with wrong log (log can be a disk dataset or tape volume)
         .
         .
         .
        ALLOCATE     same LUWID as the LUW that is being resynchronized,
                     SYNC_LEVEL(CONFIRM),
                     TPN(X'06F2') ...

           BIND
           ------------------------------------------------------------>

           Attach
           ------------------------------------------------------------>
           Exchange Log Name, log status (warm), log name (log name)
           ------------------------------------------------------------>
(2)        Compare States, CD
           ------------------------------------------------------------>
(3)        Exchange Log Name(error reply), RQE1, CEB
           <------------------------------------------------------------
```

Figure 5.3-29. Log Name Mismatch during Resync

The following notes correspond to the numbers in Figure 5.3-29.

1. The LU IPLs warm, but the wrong log volume is active. However, RM and PS.SPS do not know this at first, and proceed with resync processing.

2. The partner PS.SPS detects the mismatch of log names, notifies its control operator with Message 12, and returns an error reply.

3. PS.SPS sees the error reply and notifies its control operator of the mismatch with Message 10. Conversations with SYNC_LEVEL(SYNCPT) cannot be allocated between these LUs until the mismatch has been fixed. Perhaps the correct volume can be activated; or the operator can use a cold IPL, although this may damage the consistency of protected resources.

PS_SPS

FUNCTION:	To coordinate sync point processing, as described in this chapter. Details are not formally specified.

The sync point service is made up of a controlling subcomponent (PS.SPS) that determines when presentation services headers should be sent, and a subcomponent (in conversation resources protection manager [PS.CRPM]) that builds and sends the sync point headers. The subcomponents that build and send sync point headers are:

1. PREPARE
2. REQUEST_COMMIT
3. COMMITTED
4. FORGET
5. HEURISTIC_MIXED

The calling tree to show the relation of the components of sync point services is shown in Figure 5.3-30 on page 5.3-36.

A high-level overview of these subcomponents is described below.

PREPARE

The presentation services header contains a field (i.e., Sync Point Control Modifier) by which the receiver is requested to take a specific action upon completion of the sync point flows. This is done because the initiator issues SYNCPT when it is in either SEND state or DEFER state (see FSM_CONVERSATION on page 5.1-65). DEFER state is reached two ways: by issuing PREPARE_TO_RECEIVE or DEALLOCATE with TYPE(SYNC_LEVEL) when the sync level is SYNCPT. At the completion of the sync point flow, the sender of the last sync point command has send control; however, the TP may not need send control. Therefore, on the first command of the sequence, the Sync Point Control Modifier field of the PS header indicates the side of the conversation that is to have send control, or deallocation responsibility, after the last sync point command is sent. The Sync Point Control Modifier can be set to the following values:

Request RECEIVE: The sync point initiator requests to be in RECEIVE state upon completion of the sync point flow.

Request DEALLOCATE: The sync point initiator requests that a DEALLOCATE be issued upon completion of the sync point flow.

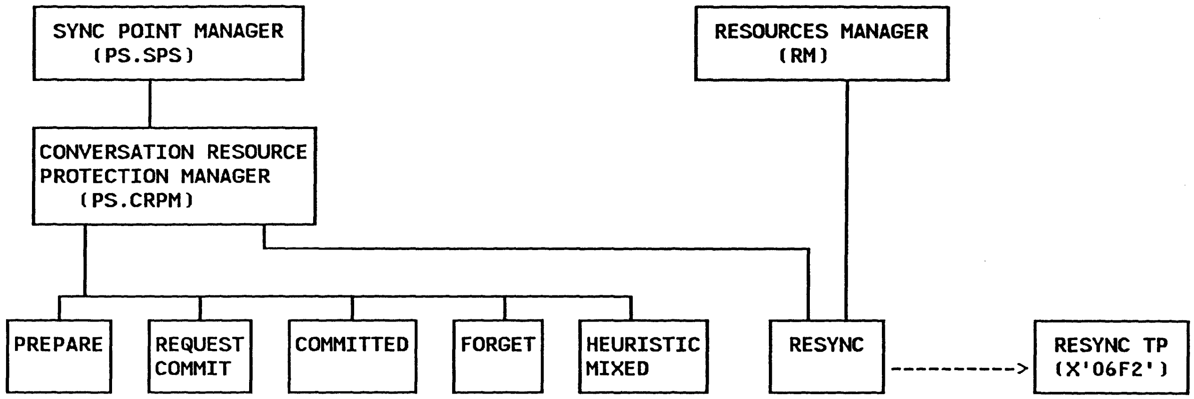
Request SEND: The sync point initiator requests to be in SEND state upon completion of the sync point flow.

When PREPARE is issued, the CD and CEB indicators in PS's send buffer (see Chapter 5.1) may be in one of three combinations of settings:

1. ¬CD and ¬CEB—neither PREPARE_TO_RECEIVE nor DEALLOCATE has been issued.
2. CD and ¬CEB—PREPARE_TO_RECEIVE has been issued.
3. ¬CD and CEB—DEALLOCATE has been issued.

If in state 1 (¬CD and ¬CEB), a PS header (Prepare) with modifier Request SEND is placed in the send buffer. The RQE1, CD, and EC indicators are turned <u>on</u> and the send buffer is sent to DFC. The Prepare then requires a reply. The reply will be either a PS header (Request Commit | Forget) or a -RSP. If a PS header is received, PREPARE subcomponent returns with a REQUEST_COMMIT or FORGET return code. It can also return RESOURCE_FAILURE (it is not a resource-specific verb). If a PS header, -RSP(0846), or resource failure is not present, a fatal error has occurred and the session is deactivated (using X'FE' reason code and appropriate UNBIND sense code). If a -RSP(0846) is received, the next data to arrive on the session is an FMH-7 and the return code is set according to the FMH-7 sense data.

If in state 2 (CD and ¬CEB), a PS header (Prepare) with modifier Request RECEIVE is placed in the send buffer. The RQE1, CD, and EC indicators are turned <u>on</u> and the send buffer contents are transmitted to DFC. The PREPARE subcomponent then requires a reply. A PS header indicates a successful Prepare;



Note: All relationships are via Call and Return except for the RESYNC TP, which is invoked as a remote service transaction program.

Figure 5.3-30. Sync Point Services Calling Tree

the return code is set accordingly. If a -RSP(0846) is received, the next data to arrive on the session is an FMH-7 and the return code is set accordingly.

If in state 3 (¬CD and CEB), a PS header (Prepare) with modifier Request DEALLOCATE is placed in the send buffer. The RQE1, CD, and EC indicators are turned on and the send buffer contents are transmitted to DFC. The PREPARE subcomponent then requires a reply. A PS header indicates a successful Prepare; the return code is set. If a -RSP(0846) is received, the next data to arrive on the session is an FMH-7 and the return code is set accordingly.

- REQUEST_COMMIT

As for Prepare, PS's send buffer may be in the same three states when Request Commit is sent. Additional information is also known. PS.SPS and PS.CRPM know whether or not the current Request Commit is being sent in reply to a Prepare. PS.CRPM uses the information to build the PS header. PS.SPS knows whether or not changes have occurred in other resources for this LUW.

When Prepare has not been received, these cases apply:

1. When in state 1 (¬CD and ¬CEB), the REQUEST_COMMIT subcomponent causes PS header (Request Commit, Request SEND) to be transmitted and waits for the reply. If Committed is received, REQUEST_COMMIT completes normally; however, if a -RSP(0846) is received, REQUEST_COMMIT processing waits for the FMH-7 and completes with the appropriate return code. Session outage is indicated in the return code for REQUEST_COMMIT as resource failure.

2. When in state 2 (CD and ¬CEB), the REQUEST_COMMIT subcomponent causes a PS header (Request Commit, Request RECEIVE) to be sent. The reply will be either Committed, SON, or a -RSP(0846) and FMH-7. The appropriate return code is set.

3. When in state 3 (¬CD and CEB), the REQUEST_COMMIT subcomponent causes a PS header (Request Commit, Request DEALLOCATE) to be sent. The COMMITTED, SON, or -RSP(0846) and FMH-7 reply sets the return code.

When Prepare has been received, one of these cases applies (PS.SPS chooses):

1. Changes have occurred in local or cascaded resources. PS header (Request Commit, Request SEND) is sent and the return code set according to the reply.

2. No changes have occurred in either local or cascaded resources. The sync point manager does not send Request Commit; Forget is sent next.

COMMITTED

Committed is sent by PS.SPS as a reply to Request Commit. Committed is sent with RQE1, CD. No Sync Point Control Modifier is sent. If Committed is sent from the last resource, CD and CEB are set by PS.CRPM as specified in Sync Point Control Modifier from the previous Request Commit.

FORGET

Unlike the case for the PREPARE and REQUEST_COMMIT subcomponents, the send buffer is in a known state when Committed and Forget are sent. The FORGET subcomponent uses the information passed on the Sync Point Control Modifier field of Prepare to leave the conversation in the state desired by the transaction that initiated SYNCPT.

HEURISTIC_MIXED

As in the case for FORGET, the send buffer is in a known state when Heuristic Mixed is sent. The HEURISTIC_MIXED subcomponent builds and sends the PS header(Heuristic Mixed) using the information passed on the Sync Point Control Modifier field of the Prepare to leave the conversation in the state desired by the transaction that initiated SYNCPT.

The Heuristic Mixed PS header is sent when a sync point agent discovers that two or more cascaded agents have gotten out of sync after a failure and resync. This is illustrated in Figure 5.3-31 on page 5.3-38. In this diagram, conversation or session failures at TPb with TPc and TPd can lead to a heuristic decision at TPc that conflicts with the heuristic decision that is made at TPd. This can be avoided with properly defined failure recovery procedures for the LU control operator. However, if heuristic damage occurs, it is discovered when TPb resyncs with TPc and TPd. Because no failure has occurred between TPb and TPa, no resync occurs on that conversation. The Heuristic Mixed PS header is used to inform the sync point initiator, TPa, that a heuristic decision has caused damage in the distributed LUW.

SESSION FLOWS CREATED BY SYNC POINT

The following illustrates the flows that can be generated by SYNCPT:

```
                                                          SYNCPT flow              TPc
                                               <-------------------------->  (cascaded agent)
                                                      SON followed by resync
                                               <-------------------------->
     TPa          SYNCPT flow         TPb
 (sync point  <--------------->  (sync point                                         TPd
  initiator)          .              agent)               SYNCPT flow          (cascaded agent)
                      .                        <-------------------------->
                                                     SON followed by resync
                                               <-------------------------->
         PS Header(Heuristic Mixed)
       <--------------------------
```

Figure 5.3-31. Heuristic Mixed in Reply to Sync Point Flow

```
 +-------+                                                        +-------+
 | TP(1) |                                                        | TP(2) |
 |       |                                                        |       |
 +-------+                                                        +-------+
 SEND_DATA
 PREPARE_TO_RECEIVE
 SYNCPT
                                                                   RECEIVE_AND_WAIT
            data, Request Commit(Request RECEIVE), RQE1, CD          WHAT_RECEIVED=DATA
        ---------------------------------------------------------->
                                                                     WHAT_RECEIVED=DATA
                                                                   RECEIVE_AND_WAIT
                                                                     WHAT_RECEIVED=TAKE_SYNCPT
            Committed, BC, ¬EC, ¬CD                                SYNCPT
        <----------------------------------------------------------    RC=OK
                                                                     (LUW STATE is COMMITTED)

 -or-

 SEND_DATA
 SYNCPT
                                                                   RECEIVE_AND_WAIT
            data, Request Commit(Request SEND), RQE1, CD             WHAT_RECEIVED=DATA
        ---------------------------------------------------------->
                                                                   RECEIVE_AND_WAIT
                                                                     WHAT_RECEIVED=TAKE_SYNCPT
            Committed, RQE1, CD                                    SYNCPT
        <---------------------------------------------------------     RC=OK
                                                                     (LUW STATE is COMMITTED)
 -or-

 SEND_DATA
 DEALLOCATE
 SYNCPT
                                                                   RECEIVE_AND_WAIT
            data, Request Commit(Request DEALLOCATE), RQE1, CD       WHAT_RECEIVED=DATA
        ---------------------------------------------------------->
                                                                   RECEIVE_AND_WAIT
                                                                     WHAT_RECEIVED=TAKE_SYNCPT
            Committed, RQE1, CEB                                   SYNCPT
        <---------------------------------------------------------     RC=OK
                                                                     (LUW STATE is COMMITTED)
```

Figure 5.3-32. Verb Sequences and Sync Point Flows to the Last Agent, Which Has No Cascaded Resources

```
 TP(1)                                                  TP(2)

                                  .
                                  .
                                  .                     RECEIVE
SEND_DATA          ---------------------------------->    RC=OK WHAT_RECEIVED=DATA
  .                               .                         .
  .                               .                     RECEIVE
  .                               .                       RC=OK WHAT_RECEIVED=DATA
SEND_DATA          data, Prepare(Request SEND), RQE1, CD RECEIVE
SYNCPT             ---------------------------------->    RC=OK WHAT_RECEIVED=TAKE_SYNCPT
                                                        SYNCPT
                      Forget, RQE1, CD
                   <----------------------------------    RC=OK (LUW STATE is COMMITTED)
  RC=OK (LUW STATE is COMMITTED)
```

Figure 5.3-33. Sync Point with No Resources Changed: The Request SEND on the Prepare is a command to send CD on the return flow. The transaction program is not given a chance to send any data or to influence the conversation states (other than to terminate abnormally). This Request SEND is not the RH CD indicator but is in the PS header.

```
 TP(1)                                             TP(2)

                              .
                              .
                              .                    RECEIVE
SEND_DATA      ---------------------------------->   RC=OK WHAT_RECEIVED=DATA
  .                           .                        .
  .                           .                    RECEIVE
  .                           .                      RC=OK WHAT_RECEIVED=DATA
  .             data, Prepare(Request SEND), RQE1, CD  RECEIVE
SYNCPT         ---------------------------------->   RC=OK WHAT_RECEIVED=TAKE_SYNCPT
                   Request Commit, RQE1, CD        SYNCPT
               <----------------------------------
                      Committed, RQE1, CD
  .            ---------------------------------->
                        Forget, RQE1, CD
               <----------------------------------   RC=OK (LUW STATE is COMMITTED)
  RC=OK (LUW STATE is COMMITTED)
SEND_DATA
  .                           Data
  .            ---------------------------------->
                                                   RECEIVE
```

Figure 5.3-34. Sync Point with Changes to Protected Resources, Request SEND: In this flow, the Prepare requests that the CD be returned (Request SEND) on the Forget.

```
 TP(1)                                                                   TP(2)

                                        .
                                        .
                                        .                                 RECEIVE
SEND_DATA           ---------------------------------------------------->   RC=OK WHAT_RECEIVED=DATA
   .                                    .                                     .
   .                                    .                                 RECEIVE
   .                                    .                                   RC=OK WHAT_RECEIVED=DATA
PREPARE_TO_RECEIVE    Data, Prepare(Request RECEIVE), RQE1, CD            RECEIVE
SYNCPT              ---------------------------------------------------->   RC=OK
                                                                            WHAT_RECEIVED=TAKE_SYNCPT
                        Request Commit, RQE1, CD
                    <---------------------------------------------------  SYNCPT
                        Committed, RQE1, CD
                    ---------------------------------------------------->


                          Forget, BC, ¬EC
                    <---------------------------------------------------    RC=OK
  RC=OK (LUW STATE is COMMITTED)                                            (LUW STATE is COMMITTED)
                                                                          RECEIVE
                          data                                              RC=SEND
                    <---------------------------------------------------  SEND_DATA
```

Figure 5.3-35. Sync Point with Changes to Protected resources, Request RECEIVE: The Request RECEIVE in Prepare indicates that the flow is to be reversed. The Forget is flushed to let locks be released. It is not necessary to flush the Forget if the TP is sure to generate application data right away.

```
 TP(1)                                                           TP(2)

                                        .
                                        .
                                        .                        RECEIVE
SEND_DATA           -------------------------------------------->  RC=OK WHAT_RECEIVED=DATA
   .                                    .                            .
   .                                    .                        RECEIVE
   .                                    .                          RC=OK WHAT_RECEIVED=DATA
DEALLOCATE          data, Prepare(Request DEALLOCATE), RQE1, CD  RECEIVE
SYNCPT              -------------------------------------------->  RC=OK WHAT_RECEIVED=TAKE_SYNCPT
                      Request Commit, RQE1, CD                   SYNCPT
                    <--------------------------------------------
                      Committed, RQE1, CD
                    -------------------------------------------->  RC=OK (LUW STATE is COMMITTED)
                                                                 RECEIVE
                                                                   RC=DEALLOCATE_NORMAL
                        Forget, CEB, RQE1
                    <--------------------------------------------  DEALLOCATE
                                                                 local deallocation
    local deallocation
  RC=OK (LUW STATE is COMMITTED)
```

Figure 5.3-36. Sync Point with Changes to Protected Resources, Request DEALLOCATE: The Request DEALLOCATE in the PS header (Prepare) is a command to send CEB on the return flow. The transaction program is not given a chance to send any data or to influence the conversation state if the sync point completes normally. If BACKOUT or a negative response is received, the transaction program is not deallocated and the TP may issue BACKOUT.

SESSION FLOWS CREATED BY ERRORS DURING SYNC POINT

All base error flows may occur. These include application errors, local resource failures, program failures, session failures, conversation failures, and LU failures. See Chapter 2 for an explanation of the types of errors. Additionally, BACKOUT can be issued. This verb causes flows the same as SEND_ERROR (i.e., -RSP(0846) followed by FMH-7) except the FMH-7 is limited to carrying a sense code of X'0824' (Sync Point Manager Abort). BACKOUT may be issued whenever a SEND_ERROR can be issued (i.e., it is independent of the send/receive state).

BACKOUT

The BACKOUT verb results in the sequence shown in Figure 5.3-37.

```
Do until RC=OK|RESOURCE_FAILURE_*|BACKED_OUT|DEALLOCATE_*
   Issue SEND_ERROR with a sense code of X'0824'.
   Issue a CONFIRM verb (Backout flows RQD2|3).
If send control was at the other end at the last sync point
   Issue PREPARE_TO_RECEIVE (FLUSH).
```

Figure 5.3-37. BACKOUT Logic

This has the advantage of propagating the backout even if the partner transaction has issued SEND_ERROR. It also handles send and receive state variations.

The expansion shown above places responsibilities on the transaction programs: for instance, if entered while the partner has the CD bit and before the first RU of the chain arrives, it can hang in the SEND_ERROR for a long time. This is because the SEND_ERROR doesn't cause a -RSP to flow until a chain arrives. Transaction programs that issue BACKOUT must take the potential delay into account. It is the transaction program's responsibility to make sure that the delay has no undesirable results. If the BACKOUT process takes too long to complete, the session can be abnormally terminated. The LUW state will be repaired by resync processing.

Abnormal termination after a BACKOUT verb results in several flows, but this is acceptable, since it is an error case.

Transaction programs that are cooperating with each other need to obey a discipline in issuing SYNCPT. A TP must be coded to issue SYNCPT when its partner TP expects a sync point request. However, because the CD bit is, in effect, a protected variable (i.e., it flows in the Sync Point Control Modifier field of the PS header and the sync point manager is responsible for maintaining the conversation in the proper state with respect to the CD bit) the TPs do not need to obey a convention for BACKOUT. BACKOUT may be issued in SEND, DEFER, RECEIVE, CONFIRM, SYNC POINT, or BACKED-OUT state. The SEND state is restored to the transaction that owned it at the completion of the last successful SYNCPT. For BACKOUT prior to the first SYNCPT call, the CD bit is restored to the Attach sender.

This page intentionally left blank

CHAPTER 5.4. PRESENTATION SERVICES--CONTROL-OPERATOR VERBS

INTRODUCTION

This chapter presents an overview of LU services for the LU control operator, and in particular describes those services contained in the presentation services components of the LU and in LU service transaction programs.

FUNCTION SUMMARY

The control operator is represented to the LU by a control-operator transaction program which invokes operator functions by issuing LU-defined control-operator verbs. The relationship between the control-operator transaction program and the control operator is implementation-defined and is not determined by SNA. Throughout this chapter, the terms control-operator and control-operator transaction program are used synonymously.

The control-operator transaction program differs from application transaction programs in its focus on control-operator concerns and its privileged access to the control-operator verbs.

The functions available to the control operator and the control-operator verbs that invoke them are described in SNA Transaction Programmer's Reference Manual for LU Type 6.2. That book is a prerequisite to this chapter.

The control operator describes and controls the availability of certain resources. The particular functions and corresponding control-operator verbs are:

- To describe the network resources accessed by the local LU, such as transaction programs, partner LUs, and mode names. The relevant verbs are:
  - DEFINE
  - DISPLAY
- To control the number of sessions between the LU and its partners. The relevant verbs are:
  - INITIALIZE_SESSION_LIMIT
  - RESET_SESSION_LIMIT
  - CHANGE_SESSION_LIMIT
  - ACTIVATE_SESSION
  - DEACTIVATE_SESSION
- To invoke local processing on behalf of a control-operator verb issued at a remote LU. The relevant verb is:
  - PROCESS_SESSION_LIMIT (This verb is not available to the local operator, but is issued from within the LU.)

STRUCTURE SUMMARY

This chapter describes two LU components for control-operator functions: presentation services for the control operator (PS.COPR), a component of presentation services, and the CNOS service transaction program (CNOS service TP). It also describes the functional relationship of these components to the installation- or implementation-defined control-operator transaction program, to the LU resources manager (RM--see Chapter 3), to presentation services for conversations (PS.CONV--see Chapter 5.0 and Chapter 5.1), and to half-sessions (HS--see "Chapter 6.0. Half-Session").

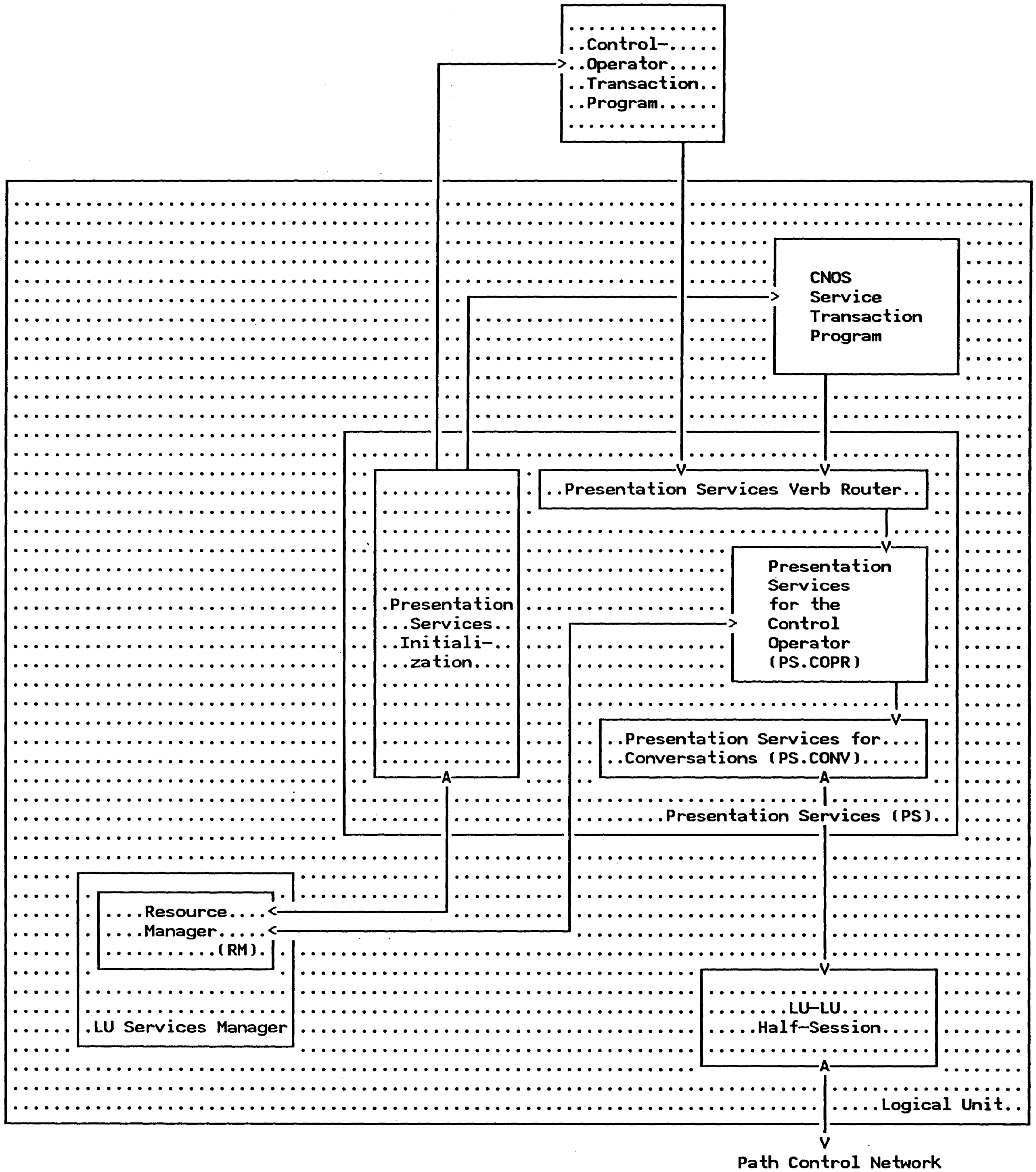
Figure 5.4-1 on page 5.4-2 shows the structural relationship of these components (see Chapter 2 for the complete structure of the LU).

CONCEPTS AND TERMS

This section describes some of the concepts and terms used throughout this chapter.

OPERATOR

The control-operator transaction program is an implementation-defined transaction program that interacts with presentation services on behalf of, or in lieu of, a human operator.



Note: Unshaded components are described in this chapter.

Figure 5.4-1. Control-Operator Components in Relation to Other Components of the LU

The control-operator transaction program interacts with presentation services by issuing control-operator verbs to control the LU or to control the interactions of the LU with a partner LU.

A control-operator verb is a privileged verb that may be issued by the control-operator transaction program to convey the operator request to the internal components of the LU. Control-operator verbs are described in SNA

Transaction Programmer's Reference Manual for LU Type 6.2.

SCOPE OF CONTROL-OPERATOR FUNCTIONS

LU control-operator-verb functions vary in scope.

Control-operator local functions affect only that LU whose control operator issues the control-operator verb, or they affect a session with another LU but take effect without the concurrent participation of a control-operator transaction program at the other LU. These functions include describing LU-accessed resources, regulating the number of sessions with single-session LUs, and activating and deactivating specific sessions.

Control-operator distributed functions affect the relationship between the LU at which the control-operator verb is issued (called the source LU) and another LU with which it shares one or more sessions (called the target LU). The functions take effect only with the cooperation of transaction programs representing the control operators at the two LUs. These functions involve primarily regulating the number of parallel sessions with other LUs, including orderly increase from no sessions and decrease to no sessions; they are called change-number-of-sessions (CNOS) functions.

A control-operator verb for distributed functions may be issued at either LU. Thus, the roles of source LU and target LU are relative to a particular verb issuance: a particular LU may be source LU for one issuance and target LU for another.

LU-ACCESSED NETWORK RESOURCES

The control operator describes to the local LU those network resources accessed from the local LU (LU-accessed network resources). The following resources are described.

- The local LU itself
- Transaction programs available for execution at this LU
- Partner LUs: The remote LUs with which this LU can have sessions
- Modes: defined sets of characteristics for sessions with particular partner LUs (One or more modes are defined for each potential partner LU.)

The control operator also controls the number and availability of the following resources:

- Sessions with particular partner LUs.

Each LU resource is identified to the operator either implicitly or by a resource key such as a transaction program name, a partner LU name, a mode name, or a session identifier.

Each LU resource is described by the LU definition that characterizes the way the LU can use it. For example, these include transaction program characteristics such as availability status and optional functions supported; LU capabilities such as parallel sessions; mode name attributes such as session limits, RU size bounds, and cryptography; and control point capabilities such as INIT (logon) formats supported.

SESSION CHARACTERISTICS

Session Identification

Most control-operator verbs do not specify a specific session, but specify only the partner LU and mode name for the session; the implementation selects the particular session. Some verbs, however, can reference a specific session by specifying an implementation-supplied unique session ID.

Single vs. Parallel Sessions

An LU can be characterized by the number of sessions it allows with other LUs. A single-session LU can have only one LU-LU session with a given partner LU. A single session LU may have more than one session concurrently, but each concurrent session is with a different partner. A parallel-session LU can have one or more concurrently active sessions with a given partner LU, subject to session limits discussed below.

The term parallel session denotes any session between a pair of parallel-session LUs, even if only one such session is currently active. This contrasts with the term single session, which denotes a session between a pair of single-session LUs or between a single-session LU and a parallel-session LU. A parallel session--even a solitary parallel session--uses protocols different from those used on a single session.

Contention Polarity

Sessions are also characterized by their contention polarity. This determines which of the two LUs has the right to control use of the session. If two LUs attempt to initiate a conversation on the same session simultaneously, the LU that is contention winner for that session will succeed and the other, the contention loser, will fail.

When used in reference to sessions, these terms are relative to the perspective of one of the LUs: a session for which an LU is the contention winner is called a contention-winner session from its perspective, but it is a contention-loser session from the perspective of the partner LU. Unless otherwise specified, the perspective used in this chapter is that of the LU at which a relevant control-operator verb is issued.

SESSION LIMITS AND COUNTS

The number of active sessions between two LUs fluctuates as a result of transaction program demand and explicit operator action. The number of sessions active at any given time is called the session count.

The maximum number of sessions allowed between LUs is set dynamically by the LU operators. This number is called a session limit. Several session limits may be specified by the operator.

The total LU-LU session limit is the maximum number of LU-LU sessions allowed by the local LU. If this limit is 1, the LU is a single-session LU; if it is greater than 1, the LU is a parallel-session LU. This limit regulates the total LU-LU session count.

The operator can regulate the number of sessions between the LU and a particular partner LU, and hence the number of transactions that can be active concurrently using that pair of LUs.

The (LU,mode) session limit specifies the currently allowed maximum number of sessions with a specific partner LU using a specific mode name. This limits the corresponding (LU,mode) session count, i.e., the number of currently active sessions with that partner LU using that mode name. One such limit and count exist for each mode name that is defined for each potential partner LU.

In this chapter, unless otherwise specified, the unqualified terms "session limit" and "session count" refer to the (LU,mode) session limit and count, respectively.

For parallel-session connections, other limits regulate the (LU,mode) session count within the (LU,mode) session limit.

The operator can assure that each LU can allocate a minimum share of the concurrent conversations by setting limits on session contention polarities.

The local-LU minimum contention-winner limit is the minimum number of sessions with a particular (LU,mode) pair for which the local LU is allowed to be the contention winner; the partner-LU minimum contention-winner limit is the minimum number of sessions with that (LU,mode) pair for which the partner LU is allowed to be the contention-winner. When activating a session, each LU selects a contention-polarity for the session that is consistent with these limits, i.e., it does not encroach on the partner's allowed contention winner sessions.

The operator can specify that a certain number of sessions be activated whenever the relevant limits allow, without waiting for explicit requests for each session.

The automatic-activation limit is the maximum number of sessions that the local LU may activate in the absence of explicit requests from transaction programs or the operator.

SESSION BRINGUP AND TAKEDOWN

Phases

The following four phases of session bringup and takedown activities exist, although some phases are omitted in some circumstances.

Session-limit initialization and reset consists of issuing control-operator verbs to specify the number of sessions the LU can have with a given partner, and to specify conditions for their activation and deactivation.

Session initiation and termination consists of control-point activity that mediates requests for session activation and deactivation, such as issuing INITIATE (INIT_SELF) and CONTROL INITIATE (CINIT) or TERMINATE (TERM_SELF) RUs.

Session shutdown consists of the LU activity to terminate conversation activity (brackets on the session by issuing BRACKET INITIATION STOPPED (BIS) RUs.

Session activation and deactivation consists of exchanging the BIND or UNBIND request and response RUs between the LUs.

Control-Operator Functions

The operator can cause an orderly deactivation of sessions between a pair of LUs by specifying that the (LU,mode) session limits be reset to 0.

The operator can also specify whether to drain (i.e., satisfy) pending allocation requests before deactivating sessions. It can specify drain separately for each of the source and target LUs. If drain is specified for an LU, that LU continues using sessions until there are no further transaction-program allocation requests for a session. If drain is not specified, the LU shuts down and deactivates the sessions as soon as the current transactions finish.

The operator can specify session-deactivation responsibility, i.e., it can request that either the source LU or the target LU take responsibility for any session deactivations required as a consequence of a particular verb issuance. Session limit decreases might leave the current session count in excess of the new limits. In this case, the LU with session-deactivation responsibility computes a termination count, which is the number of sessions it must deactivate to reach the new limits. Each LU has its own termination count, i.e., one LU could be responsible for deactivating sessions to one limit, but before it had done so, a subsequent verb could make the partner LU responsible for deactivating sessions from that limit to a newer limit.

(LU,MODE) ENTRY

The LU maintains an (LU,mode) entry for each defined combination of partner LU and mode name. This describes the dynamic relative state of the local and partner LU for that mode name. This includes the session limits, session counts, drain state, and termination count.

DISTRIBUTED OPERATOR CONTROL

Change number of sessions (CNOS) is a control-operator distributed function to regulate the number of parallel sessions between a pair of LUs and to determine when sessions will be activated or deactivated. A CNOS verb issuance causes the source LU to negotiate with the target LU to establish a mutually acceptable number of parallel sessions.

To do this, the control-operator transaction program at the source LU initiates a distributed transaction, using a conversation, with the target LU. It uses the conversation to send a copy of the operator command to the partner LU and to receive a reply from the partner.

At the target LU, the transaction program that constitutes the partner for this transaction is the CNOS service transaction program (CNOS service TP), which issues complementary control-operator verbs to receive the command and send a negotiated reply. The negotiation uses an implementation-defined algorithm that does not depend on interaction with a human operator, i.e., it can run unattended, but it may use values supplied by that operator by earlier verb issuances, e.g., from LU definition verbs. The CNOS service TP may, however, use non-interactive implementation-defined means to inform the operator of any changes.

Each program then changes its session limits and performs its local responsibility for deactivating sessions.

The CNOS transaction requires use of a session. In order to allow operator commands to be exchanged regardless of the state of session traffic between the LUs, an SNA-defined mode name, SNASVCMG, is dedicated to sessions for the control-operator transactions. Each LU supports one session of each contention polarity for this mode name with each active partner LU. Thus, an LU can always obtain a contention-winner session to send a CNOS command to its partner.

LOCAL FUNCTIONS AND SERVICES

Local control-operator verbs update definitional and operational parameters at the local LU without the participation of the operator at the remote LU.

LU DEFINITION VERBS

LU definition verbs are local control-operator verbs that define or display the locally-known characteristics of the local LU and of network resources it accesses. These resources and the principal characteristics that can be defined or displayed are:

- Local LU: the fully-qualified LU name and the optional capabilities the LU supports such as parallel sessions and map names
- Partner LUs: the various names of potential partner LUs: local LU name, fully-qualified LU name, and uninterpreted LU name; the optional capabilities of the partner LU such as parallel sessions; and the list of mode descriptions for that LU.
- Modes: the mode name and optional functions that are supported by a partner LU on a mode basis, such as sync point; and session parameters that characterize this mode, such as maximum RU size, pacing counts, and cryptography.
- Transaction programs: the transaction program name, its availability, and the optional functions that it supports such as map names and sync point.

The LU definition verbs consist of four DEFINE verbs (DEFINE_LOCAL_LU, DEFINE_REMOTE_LU, DEFINE_MODE, and DEFINE_TP), four DISPLAY verbs (DISPLAY_LOCAL_LU, DISPLAY_REMOTE_LU, DISPLAY_MODE, and DISPLAY_TP), and one DELETE verb. See SNA Transaction Programmer's Reference Manual for LU Type 6.2 for detailed descriptions of these verbs.

LOCAL SESSION-CONTROL VERBS

Local session-control verbs are local control-operator verbs that set the session limits, contention polarity, and drain specification for single-session mode names and for mode name SNASVCMG, or that activate and deactivate single or parallel sessions for any mode name.

The local session-control verbs are the following.

- INITIALIZE_SESSION_LIMIT sets the (LU,mode) session limit to allow one session, for a single-sessions mode name, or to allow one session of each contention polarity, for the parallel-session mode name SNASVCMG. This allows a session to be activated when requested by a transaction program, or to be activated immediately (automatic activation) if so specified by a previously issued LU definition verb. It also specifies the contention polarity to be selected when a session is activated by the local LU and the contention-polarity negotiation rule to be used when a session is activated by a remote LU.

- RESET_SESSION_LIMIT sets the (LU,mode) session limits to 0 to cause deactivation of any currently active sessions and to disallow any further session activations. It also specifies the drain mode, indicating whether sessions are to be deactivated immediately or only when there are no remaining requests for their use.

- ACTIVATE_SESSION requests immediate activation of a session.

- DEACTIVATE_SESSION requests deactivation of a specific session. (This is the only control-operator verb that explicitly identifies a specific session.)

DISTRIBUTED FUNCTIONS AND SERVICES

CHANGE NUMBER OF SESSIONS VERBS

Change number of sessions (CNOS) control-operator verbs specify the maximum number of parallel sessions between two LUs, and, by implication, allow or require sessions to be activated or deactivated. The verbs also specify the minimum number of sessions allowed with each contention polarity. The verbs further specify whether the sessions are to be activated or deactivated immediately or according to the needs of transaction programs, and which LU is responsible for activating or deactivating sessions to attain or maintain the number of sessions within the agreed limits.

CNOS verbs are distributed-function control-operator verbs; they take effect only with the mutual participation of both the control operator at the source LU and the CNOS service transaction program at the target LU, which enforces constraints previously specified by the control operator at that LU.

The CNOS verbs are:

- INITIALIZE_SESSION_LIMIT
- RESET_SESSION_LIMIT
- CHANGE_SESSION_LIMIT
- PROCESS_SESSION_LIMIT

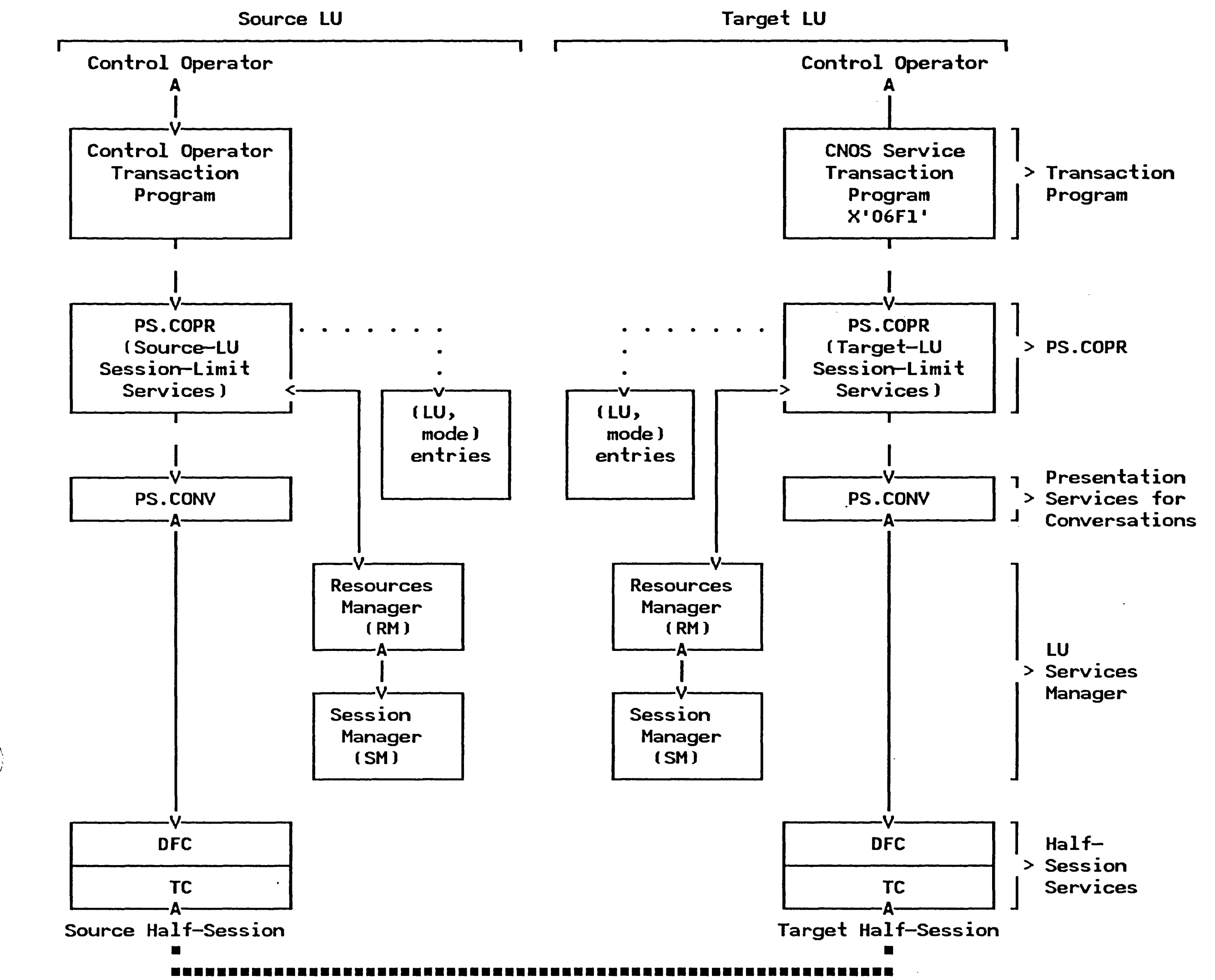
(The INITIALIZE_SESSION_LIMIT and RESET_SESSION_LIMIT verbs are included in both the local verbs and CNOS verbs. They are distinguished by the characteristics of their specified mode name.)

CNOS verbs control the number of parallel sessions by setting the (LU,mode) session limit; this limits the corresponding (LU,mode) session count.

A CNOS verb identifies the particular (LU,mode) entry that it affects, or it indicates that it affects all (LU,mode) entries for a given partner LU name. In the latter case, it affects all the (LU,mode) entries for the specified LU in the same way, e.g., it applies the same drain specification and session-deactivation responsibility to all sessions.

FUNCTIONAL RELATIONSHIPS FOR DISTRIBUTED VERB PROCESSING

The complete processing function for a CNOS verb issuance is distributed among several components at both the source and the target LUs. Figure 5.4-2 on page 5.4-7 illustrates the relationships among the major LU components involved in processing a CNOS verb. Different components are active at the source and target LUs; only the components active for the LU's role are shown for that LU.



LEGEND:

- – – –> Call/return relationship (within a process)
- <——> Send/receive relationship (between processes)
- .....> Access to shared data (within the LU)
- <■■■■> Transaction program interaction (between LUs)

Figure 5.4-2. LU Component Relationships for Distributed Session-Control Verbs

OPERATION PHASES

When the LU control operator invokes a CNOS function, the source and target LUs perform the following functions, in four phases.

1. Operator Phase--Control-Operator Transaction Program

   At the source-LU, the control-operator transaction program receives a CNOS request from the LU control operator (in an implementation-defined way) and, on behalf of the LU control operator, issues a CNOS verb. The appropriate CNOS verb invokes PS.COPR; this begins the next phase.

   Further details appear in "Control-Operator Transaction Program" on page 5.4-22.

2. Negotiation Phase--PS.COPR

   PS.COPR at the source LU initiates a conversation with PS.COPR at the target LU, via the CNOS service transaction program at the target LU. Using the conversation, the source LU sends a change number of sessions GDS variable (CNOS command) carrying an encoding of the parameters that were specified in the CNOS control-operator verb. The target LU receives the CNOS command, negotiates acceptable session limits, drain specification, and session-deactivation responsibility, and sends the acceptable values of the parameters back to the source LU in another change number of sessions GDS variable (CNOS reply).

The two LUs then terminate their conversation and make the agreed-upon changes to their respective (LU,mode) entries. Each LU then determines whether it is responsible for changing the session count, and if so, notifies its resources manager that the limits have been changed.

This phase is performed synchronously with the transaction program issuing the CNOS verb, i.e., it completes prior to return of control to the control-operator transaction program. Further details appear in "Session-Limit Services at the Source LU" on page 5.4-25 , "CNOS Service Transaction Program" on page 5.4-22, and "Session-Limit Services at the Target LU" on page 5.4-28.

3. Action Phase--Resources Manager

   The resources manager (RM) at each LU receives the session-limits-change notification (CHANGE_SESSIONS) from PS.COPR. RM determines whether any session activations or additional deactivations are required to bring the session count within the new session limits. If so, it performs the necessary session shutdown and issues requests for session deactivation to the session manager (SM). For example:

   - If the current session count is less than the minimum contention-winner limit and is also less than the automatic-activation limit, RM requests activations to reach the lower of these limits.

   - If the (LU,mode) session limit is decreased and the current session count is between the previous limit and the new limit, RM shuts down and requests deactivation of the number of sessions necessary to reduce the session count from the present value to the new limit.

   - If the (LU,mode) session limit was decreased but the current session count is above the previous limit, RM requests the additional deactivations necessary to reduce the session count from the previous limit to the new limit (the RM with session-deactivation responsibility for the previous limit continues to request the deactivations that are necessary to reach that limit).

   - If the session count for either contention polarity encroaches on the minimum contention-winner limit for the opposite polarity, RM requests deactivations sufficient to allow the minimum of each polarity, even if this would reduce the (LU,mode) session count below the (LU,mode) limit.

   When RM determines that some sessions must be deactivated, it might be that a sufficient number of sessions are not immediately free. So, each RM maintains a count, the termination count, of the number of sessions for which it has session-deactivation responsibility. It increments this count whenever a limits change requires the LU to deactivate additional sessions. It decrements this count when it requests a session deactivation.

   If the termination count is not 0, and the mutually-accepted drain specification so indicates, RM performs drain action, i.e., it continues to initiate conversations until no requests for new conversations for the specified LU name and mode name are pending from any transaction program.

   When drain action is completed, or if it was not requested, RM selects sessions of appropriate contention-polarity to be deactivated. It then shuts down all traffic on each selected session: after each partner LU ends its last bracket, it sends the BIS RU; when the partner receives this, it knows that there are no more brackets in transit from its partner. RM then issues requests to the session manager to deactivate the selected sessions.

   This phase is performed asynchronously with the transaction program issuing the CNOS verb. (Details of these functions are discussed in Chapter 3.)

4. Enforcement Phase--Session Manager

   Whenever the session manager (SM) receives a request to activate a session from RM or from the remote LU (via the PU), it checks the current session counts and session limits to determine whether another session of that contention polarity is allowed. (The resources manager also assists in limits enforcement by checking the current counts and limits before issuing session activation requests.) If another session is allowed, SM issues the appropriate BIND or response to BIND; otherwise, it rejects the request.

   Whenever the session manager receives a request to deactivate a session, it issues UNBIND or response to UNBIND.

   This phase is performed asynchronously with the transaction program issuing the CNOS verb and after the action phase. (For details of this phase, see Chapter 4.)

```
   Control-            PS.COPR at          (Information           PS.COPR at          CNOS
   Operator            Source LU            Exchanged)            Target LU           Service
   Transaction                                                                        Transaction
   Program                                                                            Program
     o-------------------o-----------------------------------------o------------------o
     .                   .                                         .                  .
 1   o *_SESSION_LIMIT   .                                         .                  .
     | (activates PS.COPR)                                         .                  .
      - - - - - - - - - ->o                                        .                  .
 2   .                   o ALLOCATE          Attach                .                  .
     .                   .   TPN(X'06F1') - - - - - - - - - - - - - - - - - - - - - -> o (CNOS service
     .                   .                                         .                  .TP invoked)
     .                   .                                         . (activates       .
     .                   .                                         .  PS.COPR)        .
 3   .                   .                                         .<- - - - - - - -  o PROCESS_SESSION
     .                   .                                         .                  .  _LIMIT
 4   .                   o GET_ATTRIBUTES                          .                  .
     .                   o GET_TP_PROPERTIES                       .                  .
 5   .                   .                                         o GET_TYPE         .
 6   .                   .                                         o GET_ATTRIBUTES.
     .                   .                                         .                  .
 7   .                   o SEND_DATA             CNOS command      o RECEIVE_AND_WAIT
     .                   .   DATA(command) - - - - - - - - - - - ->. DATA=command     .
     .                   .                                         .                  .
 8   .                   .                 Change Direction        o RECEIVE_AND_WAIT
     .                   o RECEIVE_AND_WAIT - - - - - - - - - - - ->.   WHAT_RECEIVED=SEND
     .                   .                                         .                  .
 9   .                   .                                         o (negotiates limits)
     .                   .                                         .                  .
10   .                   .                 CNOS reply              o SEND_DATA        .
     .                   .   DATA=reply  <- - - - - - - - - - - -  .   DATA(reply)    .
     .                   .                                         .                  .
11   .                   o RECEIVE_AND_WAIT   Deallocate-normal    o DEALLOCATE       .
     .                   .   RETURN_CODE=  <- - - - - - - - - - -  .   TYPE(NORMAL)
     .                   .     DEALLOCATE_NORMAL                   .                  .
     .                   .                                         .                  .
12   .                   o DEALLOCATE                              .                  .
     .                   .   TYPE(LOCAL)                           .                  .
     .                   .                                         .                  .
13   .                   o (updates limits;                        o (updates limits;
     .                      informs resources manager)             .  informs resources manager)
     .                   .                                         .                  .
14   o (informs operator)                                          .                  o (informs operator)
```

Notes:

- The figure shows the verbs issued and their most significant parameters.
- Numbers in the left column refer to the explanation in the text.
- Arrows represent information exchange resulting from verbs issued by the two transaction programs. (For an explanation of the actual message units exchanged, see Figure 5.4-4 on page 5.4-10.)

Figure 5.4-3. Sequence of Verbs and Information Exchange in CNOS Transaction Programs

CNOS TRANSACTION

The control-operator transaction program and the CNOS service transaction program, together with their corresponding PS.COPR components, process a distributed transaction to exchange the CNOS command and reply. The sequence of basic conversation verbs issued by PS.COPR at the source and target LUs is shown in Figure 5.4-3. The following comments correspond to the numbered steps in that figure.

1. The control-operator transaction program at the source LU issues one of the control-operator verbs INITIALIZE_SESSION_LIMIT, CHANGE_SESSION_LIMIT, or RESET_SESSION_LIMIT. This activates PS.COPR at the source LU (source-LU session-limit services, abbreviated SSLS). SSLS builds the CNOS command and issues a sequence of conversation verbs.

2. The source LU issues ALLOCATE to initiate a conversation with the target LU and to build an Attach FM header to invoke the CNOS service transaction program (having TP name X'06F1'). When the target LU receives the Attach, it initiates the CNOS service transaction program.

3. The CNOS service TP issues the PROCESS_SESSION_LIMIT verb. This activates PS.COPR at the target LU (target-LU session-limit services, abbreviated TSLS), which issues a sequence of conversation verbs complementary to those being issued at the source LU.

4. The source LU issues GET_ATTRIBUTES and GET_TP_PROPERTIES to get the partner's LU name and its own LU name to resolve races between contending CNOS commands.

5. TSLS issues the GET_TYPE verb to verify that this is a basic conversation.

6. TSLS issues the GET_ATTRIBUTES verb to verify that the attributes of the conversation are those expected, and to get the partner LU name. The latter is used to resolve races between contending CNOS commands.

7. SSLS issues SEND_DATA to send the CNOS command to TSLS.

   Meanwhile, TSLS issues RECEIVE_AND_WAIT to receive the command.

8. SSLS issues RECEIVE_AND_WAIT to receive the reply from SSLS. This verb has the added effect of sending a Change-Direction indication to TSLS, giving TSLS permission to send.

   Meanwhile, TSLS issues RECEIVE_AND_WAIT to receive the Change-Direction indication.

9. TSLS negotiates the proposed session limit parameters and builds the CNOS reply.

10. TSLS issues SEND_DATA to send the reply to SSLS.

    When the reply arrives at the source LU, the RECEIVE_AND_WAIT verb previously issued by SSLS completes, and SSLS receives the reply.

11. TSLS issues DEALLOCATE to end the conversation. This sends an indication to the source LU that the conversation is ended.

    Meanwhile, SSLS issues RECEIVE_AND_WAIT to receive the deallocation notification.

12. SSLS issues DEALLOCATE to complete its processing of the conversation.

13. Now both SSLS and TSLS have a copy of the negotiated reply record containing the agreed-upon limits, drain specification, and deactivation responsibility. They each update the session limits in their local data structures and inform the resources manager.

14. When SSLS and TSLS have finished processing the CNOS reply, they return control to their respective callers, the transaction programs that issued the CNOS verbs. These transaction programs then perform any further implementation-defined actions, such as notifying the LU operators of the change.

If, during the conversation, either LU detects a message unit or return code that does not conform to this protocol, it terminates the conversation by issuing DEALLOCATE TYPE(ABEND) (not shown in Figure 5.4-3), and the partner responds with DEALLOCATE TYPE(LOCAL).

For further information on verb usage, see <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>.

```
Source LU                                                  Target LU
Half-Session                                               Half-Session
o                                                                     o
.                                                                     .
.*BB, RQE1, CD, FMH-5(Attach TPN=X'06F1'), GDSID=X'1210', command data .
o-------------------------------------------------------------------->o
.                                                                     .
.                           RQE1, CEB, GDSID=X'1210', reply data      .
o<--------------------------------------------------------------------o
```

Notes:

- Each arrow represents a chain, which comprises one or more request units.
- FMH-5(Attach TPN=X'06F1') is the encoding of the Attach from the ALLOCATE verbs.
- Request-header indication CD is the encoding of Change Direction.
- GDS ID=X'1210' distinguishes the CNOS command or reply record from other GDS variables.
- Request-header indication CEB is the encoding of Deallocate-normal.
- These flows are generated by the CNOS transaction as illustrated in Figure 5.4-3 on page 5.4-9. Unless errors occur, the CNOS transaction always generates the same flow.

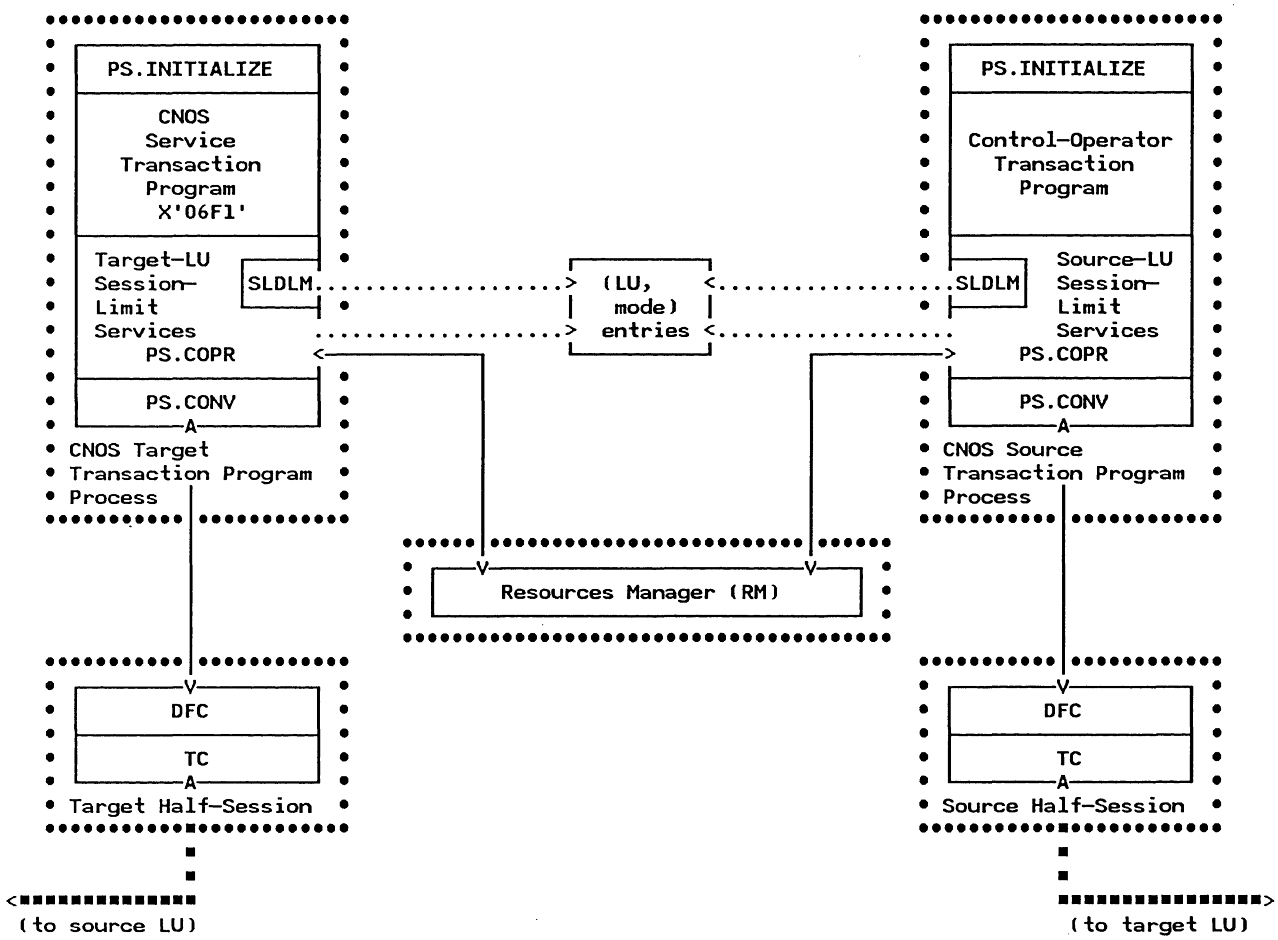
Figure 5.4-4. CNOS External Message-Unit Flows

CNOS EXTERNAL MESSAGE-UNIT FLOWS

The CNOS transaction presented in "CNOS Transaction" on page 5.4-9 causes other LU components to generate the request chains shown in Figure 5.4-4 on page 5.4-10. This is the external representation of the information exchanged by the verbs.

Exactly one bracket is initiated for each CNOS verb issued at the source LU. The bracket consists of exactly two chains, each containing exactly one Change Number Of Sessions GDS variable (CNOS command or CNOS reply).

A single CNOS verb generates only one chain in each direction, even if MODE_NAME(ALL) is specified. In that case, the verb affects all mode names the same, e.g., there is a single negotiated response, and all (new) session deactivations have the same drain status and session-deactivation responsibility.



LEGEND:
juxtaposed boxes: Call/return relationship (within a process)
<———> Send/receive relationship (between processes)
.....> Access to shared data
•••••• Process boundaries
<■■■■> Transaction program interaction (with transaction programs at other LUs)
SLDLM session-limit-data-lock manager
Note: Verb routers have been omitted.

Figure 5.4-5. CNOS Process Interactions at a Single LU

THE CNOS PROCESS RELATIONSHIPS

Processes

The LU components that support the CNOS function are distributed among several processes, as illustrated in Figure 5.4-5.

The source transaction-program process contains the control-operator transaction program; this program interacts with the internal LU components by issuing control-operator verbs, specifically, INITIALIZE_SESSION_LIMIT, CHANGE_SESSION_LIMIT, and RESET_SESSION_LIMIT.

The target transaction-program process contains the CNOS service transaction program. This program interacts with the internal LU components by issuing the PROCESS_SESSION_LIMIT verb.

(The transaction programs also interact with the LU control operators in an implementation-defined way.)

Each transaction-program process also contains within PS.COPR a session-limit services component (source or target), which processes the control-operator verbs. In processing a CNOS control-operator verb, session-limit services interacts with other LU components and, indirectly, with its peer in the partner LU, by issuing basic conversation verbs, e.g., ALLOCATE, SEND_DATA, RECEIVE_AND_WAIT, and DEALLOCATE. Session-limit services also accesses the (LU,mode) entries within the internal environment of the LU.

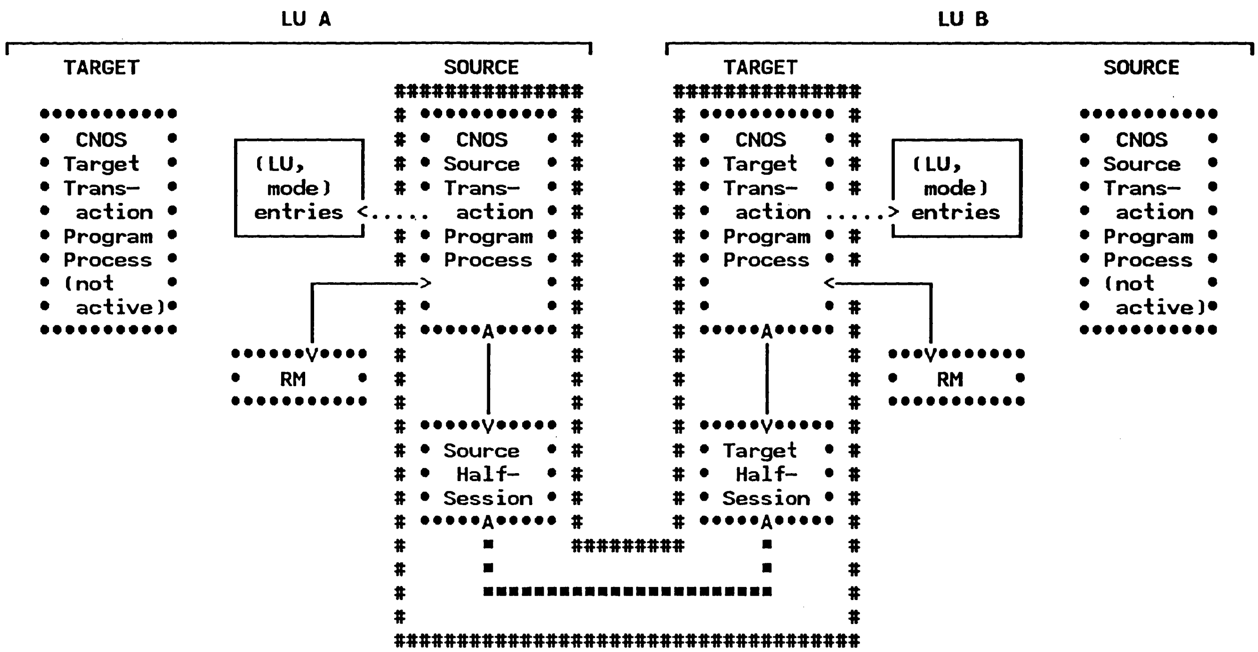
Multiple CNOS transaction-program processes, and corresponding half-session processes, can be active concurrently at any LU. For example, both the local control operator and a remote control operator might issue a CNOS verb at about the same time. Or two remote operators might both issue CNOS to the same LU. The local LU implementation might even allow two control-operator transaction programs to be active at the same time.

(Only one instance of the resources-manager process exists per LU.)

Shared Data

An (LU,mode) entry is a shared data structure owned by the LU process (not shown). An (LU,mode) entry exists for each combination of mode name and potential partner LU. Each (LU,mode) entry contains the session limits and other CNOS parameters affected by the CNOS verbs, such as the drain status. (It also contains other fields not used by CNOS.)

Each (LU,mode) entry also is associated with a session-limit-data lock field, that serves as a lock on that entry to prevent simultaneous changes to the entry by different control-operator verb issuances. The state of the session-limit-data lock is maintained by the session-limit-data-lock manager (SLDLM), a PS.COPR component that each transaction-program process invokes to obtain or release exclusive use of an (LU,mode) entry.



LEGEND:
<——> Send/receive relationship (between processes)
.....> Access to shared data (within the LU)
Transaction-handling boundaries
•••••• Process boundaries
<■■■■> Transaction program interaction (between LUs)

Figure 5.4-6. Transaction Handling Component Relationships--Case 1: Single Verb Issuance

Transaction-Handling Process Relationships

Single Verb Issuance: A single issuance of a CNOS verb uses unique instances of a control-operator transaction program process and half-session process at the source LU and of a CNOS service-transaction program processes and half-session process at the target LU. These processes have shared access to the single instances of the resources manager process and the set of (LU,mode) entries at their respective LUs. These components, with the conversation between them, process a single CNOS transaction, as illustrated in Figure 5.4-6 on page 5.4-12.

Several different cases of process and transaction relationships can occur when two CNOS verbs are issued concurrently at a local LU, at two partner LUs, or at both a local and a partner LU. If the two verb issuances are not contending for the same (LU,mode) entry, both verb issuances complete concurrently (if no errors occur). But if the two verb issuances are contending for the same (LU,mode) entry, one of the issuances will fail.

To determine whether two transactions are contending for the same (LU,mode) entry, and if so, which one wins the contention, each transaction-program process invokes its session-limit-data-lock manager. Details of this contention detection and resolution are described in "CNOS Race Resolution" on page 5.4-14.

```
                     LU A                                                   LU B
 ┌─────────────────────────────────────────────┐  ┌─────────────────────────────────────────────┐
    TARGET                           SOURCE           TARGET                           SOURCE
 ###############               ###############    ###############               ###############
 # ••••••••••• #               # ••••••••••• #    # ••••••••••• #               # ••••••••••• #
 # •  CNOS   • #  ┌─────────┐  # •  CNOS   • #    # •  CNOS   • #  ┌─────────┐  # •  CNOS   • #
 # • Target  • #  │ (LU,    │  # • Source  • #    # • Target  • #  │ (LU,    │  # • Source  • #
 # • Trans-  • #  │  mode)  │  # • Trans-  • #    # • Trans-  • #  │  mode)  │  # • Trans-  • #
 # •  action .....> entries <.....  action • #    # •  action .....> entries <.....  action • #
 # • Program • #  └─────────┘  # • Program • #    # • Program • #  └─────────┘  # • Program • #
 # • Process • #               # • Process • #    # • Process • #               # • Process • #
 # •         <────────┐  ┌────────────>     • #    # •         <────────┐  ┌────────────>     • #
 # •••••A••••• #      │  │     # •••••A••••• #    # •••••A••••• #      │  │     # •••••A••••• #
 #      │      #  •••V••V••••  #      │      #    #      │      #  •••V••V••••  #      │      #
 #      │      #  •   RM    •  #      │      #    #      │      #  •   RM    •  #      │      #
 #      │      #  •••••••••••  #      │      #    #      │      #  •••••••••••  #      │      #
 # •••••V••••• #               # •••••V••••• #    # •••••V••••• #               # •••••V••••• #
 # • Target  • #               # • Source  • #    # • Target  • #               # • Source  • #
 # •  Half-  • #               # •  Half-  • #    # •  Half-  • #               # •  Half-  • #
 # • Session • #               # • Session • #    # • Session • #               # • Session • #
 # •••••A••••• #               # •••••A••••• #    # •••••A••••• #               # •••••A••••• #
 #      ■      #               #      ■      ######      ■      #               #      ■      #
 #      ■      #               #      ■                  ■      #               #      ■      #
 #      ■      #               #      ■■■■■■■■■■■■■■■■■■■■      #               #      ■      #
 #      ■      #               #                                #               #      ■      #
 #      ■      #               ##################################               #      ■      #
 #      ■      #                                                                #      ■      #
 #      ■      #                                                                #      ■      #
 #      ■      #                                                                #      ■      #
 #      ■      ##################################################################      ■      #
 #      ■                                                                              ■      #
 #      ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■      #
 #                                                                                            #
 ##############################################################################################
```

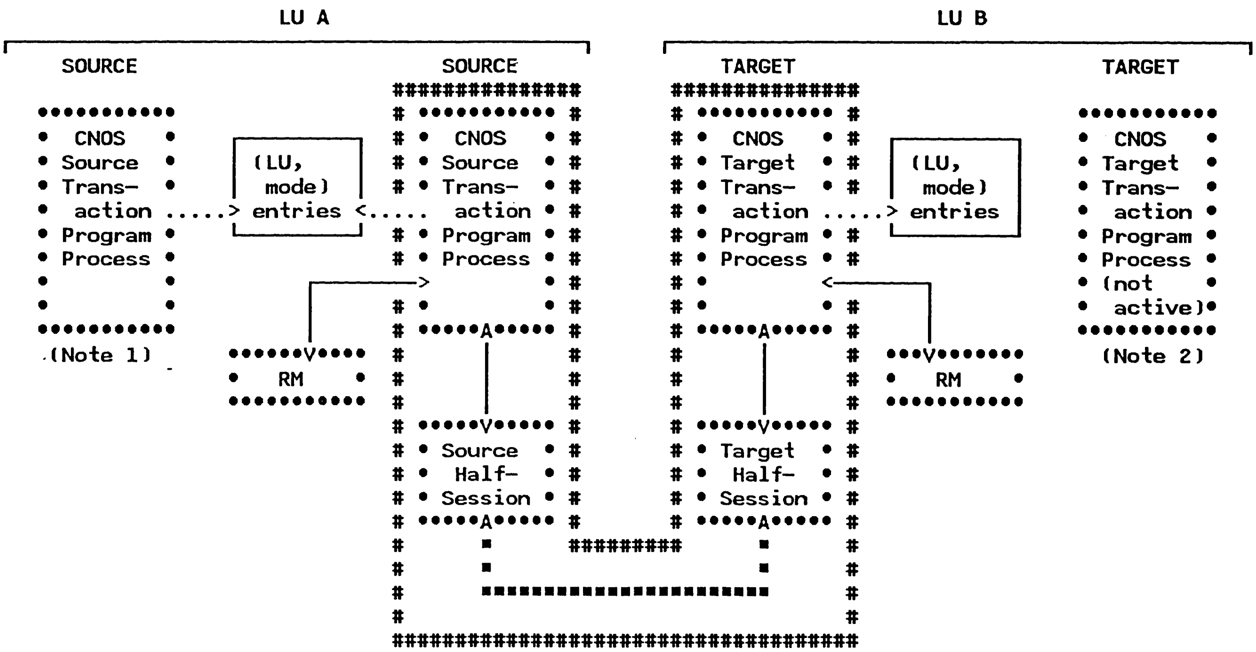
LEGEND:

```
 <────>   Send/receive relationship (between processes)
 .....>   Access to shared data (within the LU)
 ######   Transaction-handling boundaries
 ••••••   Process boundaries
 <■■■■>   Transaction program interaction (between LUs)
```

Figure 5.4-7. Transaction Handling Component Relationships--Case 2: Simultaneous Verb Issuances at Partner LUs

<u>Simultaneous Verb Issuances at Partner LUs</u>: When the LU is concurrently processing a CNOS verb from both the local LU and from the partner LU, for either the same or different (LU,mode) entries, both the source and the target processes are active at each LU, as illustrated in Figure 5.4-7 on page 5.4-13.

<u>Simultaneous Verb Issuances at the Same LU</u>: If the local LU allows two control-operator transaction programs to be concurrently active, then if two CNOS verbs are issued concurrently at that LU, two source-LU transaction-program processes become active at that LU, as illustrated in Figure 5.4-8. If contention results, the process handling the later verb issuance will terminate without initiating a conversation with its partner. If no contention results, two source processes and transactions are active at the local LU. This case is not illustrated, but is similar to Figure 5.4-7 on page 5.4-13, with the roles of source-LU and target-LU appropriately reversed.



LEGEND:

```
 <———>  Send/receive relationship (between processes)
 .....> Access to shared data (within the LU)
 ###### Transaction-handling boundaries
 •••••• Process boundaries
 <■■■■> Transaction program interaction (between LUs)
```

Notes:

1. The CNOS source transaction-program process attempts to lock an (LU,mode) entry after another source transaction-program had locked it but had not yet unlocked it. The later process is denied the lock and recognizes the contention; it goes away.
2. A target transaction-program process corresponding to the failing source process is never activated.

Figure 5.4-8. Transaction Handling Component Relationships--Case 3: Simultaneous Verb Issuances at the Same LU

CNOS RACE RESOLUTION

Command Race

Two LU control operators might simultaneously issue a CNOS verb affecting the same LU name and mode name. If such a verb is issued while another such verb at either the source or the target LU is in the negotiation phase, i.e., a prior instance of PS.COPR is active on either LU for the same (LU,mode) entry or entries, a command race has occurred, and one (but not both) of the verbs fails.

If a verb is issued when a previous verb is in the action phase, i.e., PS.COPR has already updated the (LU,mode) entry, but the resources manager and the session manager have not yet completed adjustments to the session count, an action race has occurred and neither verb fails. For details, see SNA Transaction Programmer's Reference Manual for LU Type 6.2 and Chapter 3 of this volume.

Locking the (LU,mode) Entry

When a command race occurs, PS.COPR assures that exactly one of the commands completes successfully by observing a locking protocol for the (LU,mode) entry. The session-limit services routines invoke a shared component, session-limit-data-lock manager (SLDLM), to prevent simultaneous access to an (LU,mode) entry, to detect races, and to resolve double-failure race conditions.

Source-LU session-limit services (SSLS) of PS.COPR tests and simultaneously sets the CNOS lock in the (LU,mode) entry by issuing LOCK to its SLDLM before allocating a conversation to the target LU. If another instance of session-limit services has already locked the (LU,mode) entry, SSLS returns an error code. It does not send the CNOS command to the target LU or modify the session-limit parameters in the (LU,mode) entry.

If SSLS succeeds, target-LU session-limit services (TSLS) at the partner LU issues LOCK to its SLDLM after receiving the CNOS command from the source LU. If TSLS finds the lock at its LU already set (for example, because a control-operator transaction program at its LU, acting as source LU, had simultaneously issued a CNOS verb), then TSLS sends the partner LU a CNOS reply with a reply-modifier value indicating that a command race was detected. It does not modify the session-limit parameters in the (LU,mode) entry.

In some cases, two commands issued simultaneously from each LU could both be rejected. For example, each LU might issue its command before the other arrived. Each target session-limit services would then reject the command from the partner because its source session-limit services had a command outstanding. This is called a double-failure race condition. To detect this case, SLDLM maintains another indicator, LOCK_DENIED. This is set by TSLS when it sends a command-race-detected reply modifier.

When SSLS receives the reply from TSLS, it checks the reply to determine whether the partner LU rejected the command because it detected a race. If so, it also tests the session-limit-data lock to determine if, meanwhile, its LU, acting as a target LU for another CNOS command, has rejected a command from the partner LU. SLDLM determines this from the LOCK_DENIED indicator. (LOCK_DENIED, together with the receipt of a command-race-detected reply modifier, indicates a double-failure race condition; either LOCK_DENIED or command-race-detected alone does not represent a double failure.)

Race Flows

Example flows for the types of command races that can occur are shown in Figure 5.4-10 on page 5.4-17, Figure 5.4-11 on page 5.4-18, and Figure 5.4-12 on page 5.4-19. The flows for the no-race case are shown in Figure 5.4-9 for comparison.

In the figures:

- The change number of sessions commands sent from each of the two LUs are on different conversations.
- The columns labeled "Transaction-x" show the actions performed by the CNOS transaction-program processes in processing a CNOS verb issued by the control operator at LUa.
- The columns labeled "Transaction-y" show the actions performed in processing a CNOS verb issued by the control operator at LUb.
- The column labeled "(LU,mode) entry (LUb,m)" shows the changes made by the two transactions to the (LU,mode) entry for LUb, mode name m at LUa.
- The column labeled "(LU,mode) entry (LUa,m)" shows the changes in the corresponding (LU,mode) entry for LUa, mode name m at LUb.
- MAX_SESS represents the session limit for mode name m in the (LU,mode) entry.
- SLD_LOCK represents the state (LOCKED, UNLOCKED, DENIED) of the session-limit-data lock.

The flows shown are:

- A CHANGE_SESSION_LIMIT verb (abbreviated CHANGE_SESSLIM)
- The CNOS commands and replies exchanged by the CNOS transaction-program processes,
- The internal requests (LOCK, TEST, UNLOCK) and their replies (OK, REJECT, DENIED)
- Update actions on the (LU,mode) session-limit field of the (LU,mode) entry

```
                     LUa                                              LUb
  ┌──────────────────────────────────────────────┐   ┌──────────────────────────────────────────────┐
  Transaction-x   ┌──────────┐ Transaction-y       Transaction-y  ┌──────────┐ Transaction-x
  source process  │(LU,mode) │ target process      source process │(LU,mode) │ target process
        .         │  entry   │       .                   .        │  entry   │       .
        .         │ (LUb,m)  │       .                   .        │ (LUa,m)  │       .
        .         └──────────┘       .                   .        └──────────┘       .
        .              .             .                   .              .            .
        .    (MAX_SESS is n;         .                   .    (MAX_SESS is n;        .
        .     SDL_LOCK is UNLOCKED)  .                   .     SDL_LOCK is UNLOCKED) .
        .              .             .                   .              .            .
        .              .             .        CHANGE_SESSLIM(LU(a),MODE(m),MAX_SESS(j))
1       .              .             .                   .     LOCK     .            .
        .              .             .                   o.............>             .
        .              .             .                   .    (SDL_LOCK is LOCKED)   .
2       .              .             .                   .      OK      .            .
        .              .             .                   <.............o             .
        .              .             .                   .              .            .
3       .              .             .CNOS command (MODE(m),MAX_SESS(j)).            .
        .              .             <─────────────────────             .            .
4       .              .     LOCK    .                   .              .            .
        .              <.............o                   .              .            .
        .    (SDL_LOCK is LOCKED)    .                   .              .            .
5       .              .      OK     .                   .              .            .
        .              o.............>                   .              .            .
6       .              .             .CNOS reply (MODE(m),OK)           .            .
        .              .             ─────────────────────>             .            .
7       .              .update (MAX_SESS(j))             .update (MAX_SESS(j))       .
        .              <.............o                   o.............>             .
        .    (MAX_SESS is j)         .                   .    (MAX_SESS is j)        .
8       .              .    UNLOCK   .                   .    UNLOCK    .            .
        .              <.............o                   o.............>             .
        .    (SDL_LOCK is UNLOCKED)  .                   .    (SDL_LOCK is UNLOCKED) .
        .              .             .                   .              .            .
```

Note: Numbers in the left column refer to explanations in the text.

Figure 5.4-9. No Race: Only One LU Issues a CNOS Verb

No Race: If only one LU issues a CNOS command, no race occurs, and the transaction is successful.

Figure 5.4-9 on page 5.4-16 shows the no-race case. In this example:

1. Before sending the CNOS command, the source LU (LUb) attempts to lock the affected (LU,mode) entry.

2. Since no other CNOS transaction at LUb has the (LU,mode) entry locked, the attempt is successful.

3. LUb now issues the CNOS command.

4. When the target LU (LUa) receives the CNOS command, it attempts to lock the (LU,mode) entry.

5. Since no other CNOS transaction at LUa has the (LU,mode) entry locked, the attempt is successful.

6. LUa then negotiates and sends the CNOS reply.

7. LUa then updates the (LU,mode entry).

   Similarly, when LUb receives the reply, it also updates its (LU,mode) entry.

8. Both LUs unlock the (LU,mode) entries. The (LU,mode) entries are now available for updating by subsequent CNOS verbs.

Single-Failure Races: In the single-failure cases (Figure 5.4-10 and Figure 5.4-11 on page 5.4-18), one transaction fails; it does not modify the session-limit parameters in the (LU,mode) entry. The other transaction succeeds and changes the session-limit parameters.

Figure 5.4-10 shows a single-failure race condition in which one transaction's command and reply both cross the reply of the transaction for a verb issued at the other LU. In this example,

1. LUa's command succeeds because LUb was not busy when the command arrived.

2. LUb's command fails because LUa's verb has not completed at LUa when LUb's command arrives, even though LUa's verb processing has completed at LUb.

3. When LUb receives the REJECT reply, it tests for LOCK_DENIED, which is not set, and so determines that no command from LUa (for mode name m) has been rejected

```
                         LUa                                                 LUb
 ________________________________________________    ________________________________________________
|                                                |  |                                                |
 Transaction-x   (LU,mode)   Transaction-y          Transaction-y   (LU,mode)   Transaction-x
 source process    entry     target process         source process    entry     target process
        .         (LUb,m)          .                      .        (LUa,m)           .
        .                          .                      .                          .
        .            .             .                      .            .             .
        .     (MAX_SESS is n;      .                      .     (MAX_SESS is n;      .
        .      SDL_LOCK is UNLOCKED)                      .      SDL_LOCK is UNLOCKED)
        .            .             .                      .            .             .
  CHANGE_SESSLIM(LU(LUb) MODE(m) MAX_SESS(i))             .            .             .
        .      LOCK  .             .                      .            .             .
        o............>             .                      .            .             .
        .     (SDL_LOCK is LOCKED) .                      .            .             .
        .       OK   .             .                      .            .             .
        <............o             .                      .            .             .
        .CNOS command (MODE(m) MAX_SESS(i))               .            .             .
        o------------------------------------------------------------------------->
        .            .             .                      .            .    LOCK     .
        .            .             .                      .            <............o
        .            .             .                      .     (SDL_LOCK is LOCKED) .
        .            .             .                      .            .     OK      .
        .            .             .                      .            o............>
1       .            .             .                      .      CNOS reply (MODE(m) OK).
        .            .             .      +---------------------------------------------o
        .            .             .      |               .            .update (MAX_SESS(i))
        .            .             .      |               .            <............o
        .            .             .      |               .     (MAX_SESS is i)      .
        .            .             .      |               .            .   UNLOCK    .
        .            .             .      |               .            <............o
        .            .             .      |               .     (SDL_LOCK is UNLOCKED)
        .            .             .      |               .            .             .
        .            .             .      |      CHANGE_SESSLIM(LU(LUa) MODE(m) MAX_SESS(j))
        .            .             .      |               .     LOCK   .             .
        .            .             .      |               o............>             .
        .            .             .      |               .     (SDL_LOCK is LOCKED) .
        .            .             .      |               .     OK     .             .
        .            .             .      |               <............o             .
        .            .             .CNOS command (MODE(M) MAX_SESS(j)).              .
        .            .             <----------------------o            .             .
        .            .     LOCK    .      |               .            .             .
        .            <.............o      |               .            .             .
        .     (SDL_LOCK is DENIED) .      |               .            .             .
        .            .    REJECT   .      |               .            .             .
        .            o.............>      |               .            .             .
        .     (MAX_SESS is n [no change]) |               .            .             .
2       .            .             .CNOS reply (MODE(M) RACE_REJECT)   .             .
        .            .             o--------------------->             .             .
        .            .             .      |               .     (MAX_SESS is i [no change])
3       .            .             .      |               .     TEST   .             .
        .            .             .      |               o............>             .
        .            .             .      |               .     OK     .             .
        <---------------------------------+               <............o             .
        .update (MAX_SESS(i))      .                      .     UNLOCK .             .
        o............>             .                      o............>             .
        .     (MAX_SESS is i)      .                      .     (SDL_LOCK is UNLOCKED)
        .     UNLOCK .             .                      .            .             .
        •............>             .                      .            .             .
        .     (SDL_LOCK is UNLOCKED)                      .            .             .
        .            .             .                      .            .             .
```

Note: Numbers in the left column refer to explanations in the text.

Figure 5.4-10. Single-Failure Race Condition--Case 1: Command Crosses Reply

and therefore does not attempt to retry the command.

Figure 5.4-11 shows another single-failure race condition, in which one transaction's command and reply cross the command of the transaction for a verb issued at the other LU. In this example,

1. LUb's command fails because LUa's command has not completed when LUb's command arrives.

```
                       LUa                                                LUb
  +-----------------------------------------------+   +-----------------------------------------------+
  Transaction-x  +-----------+ Transaction-y         Transaction-y  +-----------+ Transaction-x
  source process | (LU,mode) | target process        source process | (LU,mode) | target process
        .        |   entry   |      .                      .        |   entry   |      .
        .        |  (LUb,m)  |      .                      .        |  (LUa,m)  |      .
        .        +-----------+      .                      .        +-----------+      .
        .              .            .                      .              .            .
        .   (MAX_SESS is n;         .                      .   (MAX_SESS is n;         .
        .    SDL_LOCK is UNLOCKED)  .                      .    SDL_LOCK is UNLOCKED)  .
        .              .            .                      .              .            .
   CHANGE_SESSLIM(LU(LUb) MODE(m) MAX_SESS(i))             .              .            .
        .     LOCK     .            .                      .              .            .
        o.............>             .                      .              .            .
        .   (SDL_LOCK is LOCKED)    .               CHANGE_SESSLIM(LU(LUa) MODE(m) MAX_SESS(j))
        .      OK      .            .                      .     LOCK     .            .
        <.............o             .                      o.............>             .
        .CNOS command (MODE(m) MAX_SESS(i))                .   (SDL_LOCK is LOCKED)    .
        o-----------------------------------------+        .      OK      .            .
        .              .            .             |        <.............o             .
        .              .            .CNOS command (MODE(m) MAX_SESS(j)).               .
        .              .            <------------------------o            .            .
        .              .     LOCK   .             |        .              .            .
        .              <.............o            |        .              .            .
        .   (SDL_LOCK is DENIED)    .             |        .              .            .
        .              .    REJECT  .             |        .              .            .
        .              o.............>            |        .              .            .
        .   (MAX_SESS is n [no change])           |        .              .            .
1       .              .            .CNOS reply (MODE (m) RACE_REJECT) .               .
        .              .            o------------------------>            .            .
        .              .            .             |        .   (MAX_SESS is n [no change])
2       .              .            .             |        .    TEST      .            .
        .              .            .             |        o.............>             .
        .              .            .             |        .      OK      .            .
        .              .            .             |        <.............o             .
        .              .            .             |        .    UNLOCK    .            .
        .              .            .             |        o.............>             .
        .              .            .             |        .   (SDL_LOCK is UNLOCKED)  .
        .              .            .             |        .              .            .
        .              .            .             |        .              .            .
        .              .            .             +------------------------------------>
        .              .            .                      .              .     LOCK   .
        .              .            .                      .              <.............o
        .              .            .                      .   (SDL_LOCK is LOCKED)    .
        .              .            .                      .              .     OK     .
        .              .            .                      .              o.............>
3       .              .            .                      .     CNOS reply (MODE(m) OK).
        <----------------------------------------------------------------------------o
        .update (MAX_SESS(i))       .                      .              .UPDATE (MAX_SESS(i))
        o.............>             .                      .              <-------------o
        .   (MAX_SESS is i)         .                      .   (MAX_SESS is i)         .
        .   UNLOCK     .            .                      .              .   UNLOCK   .
        o.............>             .                      .              <.............o
        .   (SDL_LOCK is UNLOCKED)  .                      .   (SDL_LOCK is UNLOCKED)  .
        .              .            .                      .              .            .
```

NOTE: Numbers in left column refer to the explanations in the text.

Figure 5.4-11. Single-Failure Race Condition--Case 2: Command and Reply Cross Command

2. When LUb receives the REJECT reply, it tests LOCK_DENIED and determines that no command from LUa (for mode name m) has been rejected and therefore it does not attempt to retry the command.

3. LUa's command succeeds because LUb's unsuccessful command has already completed at LUb, and has released the lock, before LUa's command arrives at LUb.

```
                      LUa                                              LUb
   ┌─────────────────────────────────────────────┐   ┌─────────────────────────────────────────────┐
   Transaction-x    (LU,mode)   Transaction-y        Transaction-y    (LU,mode)   Transaction-x
   source process     entry     target process       source process     entry     target process
         .           (LUb,m)        .                      .           (LUa,m)        .
         .                          .                      .                          .
         .              .           .                      .              .           .
         .     (MAX_SESS is n;      .                      .     (MAX_SESS is n;      .
         .      SDL_LOCK is UNLOCKED)                      .      SDL_LOCK is UNLOCKED) .
         .              .           .                      .              .           .
1  CHANGE_SESSLIM(LU(LUb) MODE(m) MAX_SESS(i))       CHANGE_SESSLIM(LU(LUa) MODE(m) MAX_SESS(j))
         .      LOCK    .           .                      .      LOCK    .           .
         o.............>            .                      o.............>            .
         .     (SDL_LOCK is LOCKED) .                      .     (SDL_LOCK is LOCKED) .
         .       OK     .           .                      .       OK     .           .
         <.............o            .                      <.............o            .
2        .CNOS command (MODE(m) MAX_SESS(i))        CNOS command (MODE(m) MAX_SESS(j))    .
         o────────────────────────────────┐     ┌──────────o                          .
         .              .           .     │     │          .              .           .
3        .              .           .     └─────┼──┐       .              .           .
         .              .           .     ┌─────┘  │       .              .           .
         .              .           <─────┘        └──────────────────────────────────>
         .              .     LOCK  .                      .              .     LOCK  .
         .              <.............o                    .              <.............o
         .     (SDL_LOCK is DENIED)   .                    .     (SDL_LOCK is DENIED)   .
4        .              .    REJECT   .                    .              .    REJECT   .
         .              o.............>                    .              o.............>
         .     (MAX_SESS is n [no change])                 .     (MAX_SESS is n [no change])
         .              .           .CNOS reply (MODE(m) RACE_REJECT)     .           .
         .              .           o───────────┐          .CNOS reply (MODE(m) RACE_REJECT)
         .              .           .           │    ┌────────────────────────────────o
5        .              .           .     ┌─────┼────┘     .              .           .
         .              .           .     │     │          .              .           .
         <────────────────────────────────┘     └──────────>              .           .
         .      TEST    .           .                      .      TEST    .           .
         o.............>            .                      o.............>            .
6        .     DENIED   .           .                      .     DENIED   .           .
         <.............o            .                      <.............o            .
         .              .           .                      .              .           .
7   ('LUa' < 'LUb';     .           .                ('LUb' > 'LUa';      .           .
    this LU will not retry)         .                this LU will retry). .           .
         .              .           .                      .              .           .
8        .    UNLOCK    .           .                      .              .           .
         o.............>            .                      .              .           .
         .     (SDL_LOCK is UNLOCKED)                      .              .           .
         .              .           .                      .              .           .
9        .              .           .CNOS command (MODE(m) MAX_SESS(j)).  .           .
         .              .           <──────────────────────o              .           .
         .              .     LOCK  .                      .              .           .
         .              <.............o                    .              .           .
         .     (SDL_LOCK is LOCKED) .                      .              .           .
         .              .      OK   .                      .              .           .
         .              o.............>                    .              .           .
         .              .           .CNOS reply (MODE(m) OK)              .           .
         .              .           o──────────────────────>              .           .
         .              .update (MAX_SESS(j))              .update (MAX_SESS(j))      .
         .              <.............o                    o.............>            .
         .     (MAX_SESS is j)      .                      .     (MAX_SESS is j)      .
         .              .    UNLOCK .                      .     UNLOCK   .           .
         .              <.............o                    o.............>            .
         .     (SDL_LOCK is UNLOCKED)                      .     (SDL_LOCK is UNLOCKED) .
         .              .           .                      .              .           .
```

Note: Numbers in left column refer to explanations in the text.

Figure 5.4-12. Double-Failure Race Condition: Command Crosses Command, Reply Crosses Reply

Double-Failure Race: In the double-failure case (Figure 5.4-12 on page 5.4-19), both transactions initially fail. The SSLS components at each LU discover the double failure and compare their fully-qualified LU names to resolve it. (For the comparison, the fully-qualified LU names are left-justified and padded to the right with space [X'40'] characters to make the lengths equal.) The LU with LU name lower in EBCDIC collating sequence loses; the verb fails as in a single-failure race condition. The LU with LU name higher in EBCDIC collating sequence retries the CNOS command, i.e., it allocates a new conversation and sends the same CNOS command again. If no further errors occur, the verb eventually succeeds.

Figure 5.4-12 on page 5.4-19 shows a double-failure case. In this example:

1. Operators at both LUs simultaneously issue CNOS verbs.

2. The source processes successfully lock the (LU,mode) entries at their respective LUs, and issue CNOS commands.

3. The commands cross in transit.

4. When the commands arrive, the target processes attempt to lock the (LU,mode) entries but fail because they are already locked by the source processes of the other transaction, each of which has not yet received the reply to its own command. The failing attempt to lock also sets the LOCK_DENIED state of the lock. MAX_SESSIONS remains temporarily at n.

5. Each target process sends a reply indicating a race reject. These replies also cross in transit.

6. When the REJECT replies arrive, each source transaction program tests LOCK_DENIED and finds it set, indicating that a target transaction program at the same LU had attempted to set the lock but had been refused. This is a double failure: the local LU's own command has failed, and meanwhile the local LU has rejected a command from the partner LU.

7. Each source process compares LU names to determine whether it should retry.

8. The LU with low LU name (LUa) releases the lock and terminates its CNOS verb to avoid another race.

9. The LU with the high LU name (LUb) re-issues the command. Processing continues as in the no-race case (Figure 5.4-9 on page 5.4-16).

RECOVERY FROM CONVERSATION FAILURE

If conversation failure, e.g., session outage, were to occur during CNOS processing, the CNOS command would not complete successfully at the source LU. Nevertheless, it might complete at the target LU, for example, because the reply was lost after the target LU had already deallocated the conversation. In this case, the session limits could become different at the two LUs.

To prevent this discrepancy, SSLS retries any command that fails because of conversation failure. Since the original session has been lost, SSLS attempts to obtain a new session on the same or another mode name. It first tries to obtain a session with the mode name that failed, then with mode name SNASVCMG (if different), then with each mode name affected by the command, until either the command succeeds or the LU determines that no session can be allocated with any affected mode name. Session limits can be reset even if the local LU is not able to contact or complete a conversation with the partner LU. The FORCE(YES) parameter on RESET_SESSION_LIMIT instructs the control operator to set the local session limits to 0 even if the CNOS transaction is unable to complete successfully. This permits the LU to perform some clean-up that is not normally possible until session limits are 0 and no sessions are active.

BASE AND OPTIONAL SUPPORT

The basic and optional functions available at the control-operator protocol boundary are defined in SNA Transaction Programmer's Reference Manual for LU Type 6.2. This section relates those functions to the capabilities of the components in the formal description.

BASE-FUNCTION-SET SUPPORT

All implementations support an implementation-defined control-operator transaction program that is able to issue any of the required (base function set) control-operator verbs and all optional control-operator verbs and parameters that the LU supports.

The base function set, supported by all implementations, includes the functions corresponding to the LU definition verbs, i.e., the ability to specify the values of certain LU parameters that are chosen by the installation. An implementation may support issuing these verbs from the control-operator transaction program. Alternatively, instead of exposing these verbs at the control-operator protocol boundary, the implementation may provide other support in the form of installation-time, IPL-time, or run-time processing of the system-definition values, as long as the values are initialized prior to first use.

The base function set also includes local support of the functions of INITIALIZE_SESSION_LIMIT and RESET_SESSION_LIMIT that apply to single-session mode names, and includes receive support for remotely-issued ACTIVATE_SESSION and DEACTIVATE_SESSION verbs.

All LUs providing an "open" protocol boundary, i.e., one to which application transaction programs have access, also support parallel sessions, including the CNOS minimum support (see "CNOS Minimum Support Set" on page 5.4-21).

Parallel-session LUs optionally support optional function set parameters of the CNOS verbs (see "Parallel-Session Optional Functions" on page 5.4-21).

LUs with a "closed" protocol boundary, i.e., one to which application transaction programs do not have access, may optionally support parallel sessions and the corresponding CNOS minimum support.

CNOS MINIMUM SUPPORT SET

The CNOS minimum-support functions are:

- Send (source) support for INITIALIZE_SESSION_LIMIT

  This increases the session limit from 0.

- Send support for RESET_SESSION_LIMIT

  This resets the session limit to 0. This does not allow the local LU to initiate new conversations after the verb completes, but it allows the LU to accept new conversations initiated by a partner LU.

- Receive (target) support for all CNOS verbs, except that:

  - The target LU may unconditionally change RESPONSIBLE(TARGET) to RESPONSIBLE(SOURCE).

  - The target LU may unconditionally change DRAIN_TARGET(YES) to DRAIN_TARGET(NO).

The minimum-support CNOS components are:

- An implementation-supplied control-operator transaction program that can issue the CNOS minimum-support verbs

- The CNOS service transaction program (TPN=X'06F1')

- Presentation services for the control operator (PS.COPR), except for the optional functions listed in "Parallel-Session Optional Functions" on page 5.4-21

- Support for a sufficient number of reserved sessions using the SNA-defined mode name SNASVCMG

  The LU provides the capability for two such sessions for each LU with which the LU can have concurrently-active parallel sessions; these mode-name-SNASVCMG sessions are in addition to the sessions provided for user transactions. For each potential parallel-session partner LU, the operator specifies an (LU,mode) entry with mode name SNASVCMG and with limits allowing one contention-winner and one contention-loser session.

  (The SNA-defined mode name is provided so that PS.COPR will always be able to activate a session to send the CNOS command, even when all other session limits are 0, as in the initial state, or when all other active sessions are in in-brackets state or are bidder sessions on which a bid request is being refused.)

An LU that provides only the CNOS minimum-support does not expose MIN_CONWINNERS_TARGET, RESPONSIBLE, or DRAIN_TARGET at the control-operator protocol boundary. In that case, the source LU sends MIN_CONWINNERS_TARGET(implementation choice), RESPONSIBLE(SOURCE), and DRAIN_TARGET(YES) for those parameters that it does not expose.

PARALLEL-SESSION OPTIONAL FUNCTIONS

The optional parallel-session CNOS functions are:

- Receive support for DRAIN_TARGET(YES)

  This means that the LU supports local drain, i.e., it is able to start new conversations after the session limit is reset to 0 and to defer deactivating sessions until there are no more local requests for new conversations.

- Send support for any or all of the following:

  - MIN_CONWINNERS_TARGET

  - RESPONSIBLE(TARGET)

  - DRAIN_TARGET(NO)

  This means that the LU exposes these parameters at the control-operator protocol boundary.

- Receive support for RESPONSIBLE(TARGET)

  This means the LU can be responsible for decreasing the session count to a nonzero value, i.e., it maintains an exact count of sessions to be terminated.

- Send support for CHANGE_SESSION_LIMIT

  This means that the LU can increase or decrease the session limit to a nonzero value when it is currently nonzero.

This section describes the functions and interrelationships of the components for control-operator functions.

The principal components are:

- Presentation services for the control operator (PS.COPR)
- Control-operator transaction program
- CNOS service transaction program

To perform its functions, PS.COPR may invoke the following other LU components:

- Resources manager (RM), which performs session shutdown and invokes the session manager for session initiation/termination and activation/deactivation
- Presentation services for conversations, which uses an LU-LU half-session for the conversation with the partner LU

Figure 5.4-1 on page 5.4-2 illustrates the relationships among these components.

TRANSACTION PROGRAMS

Control-Operator Transaction Program

The control-operator transaction program is an implementation-defined transaction program at the source LU that represents the LU control operator. It forms part of the local-LU (source-LU) transaction-program process. It is invoked by presentation services (PS.INITIALIZE) as a result of an implementation-defined program-initiation request.

The control-operator transaction program may interact with a human operator, at the implementation- and/or installation-option, to obtain input parameters or to present results. It issues any of the supported control-operator verbs exposed at the control-operator protocol boundary.

The transaction program passes to PS.COPR a transaction-program-verb structure specifying the verb type and verb parameters. When PS.COPR processing is complete, the transaction program is returned the same structure containing the returned parameter values, e.g., a return code indicating success or a failure reason.

CNOS Service Transaction Program

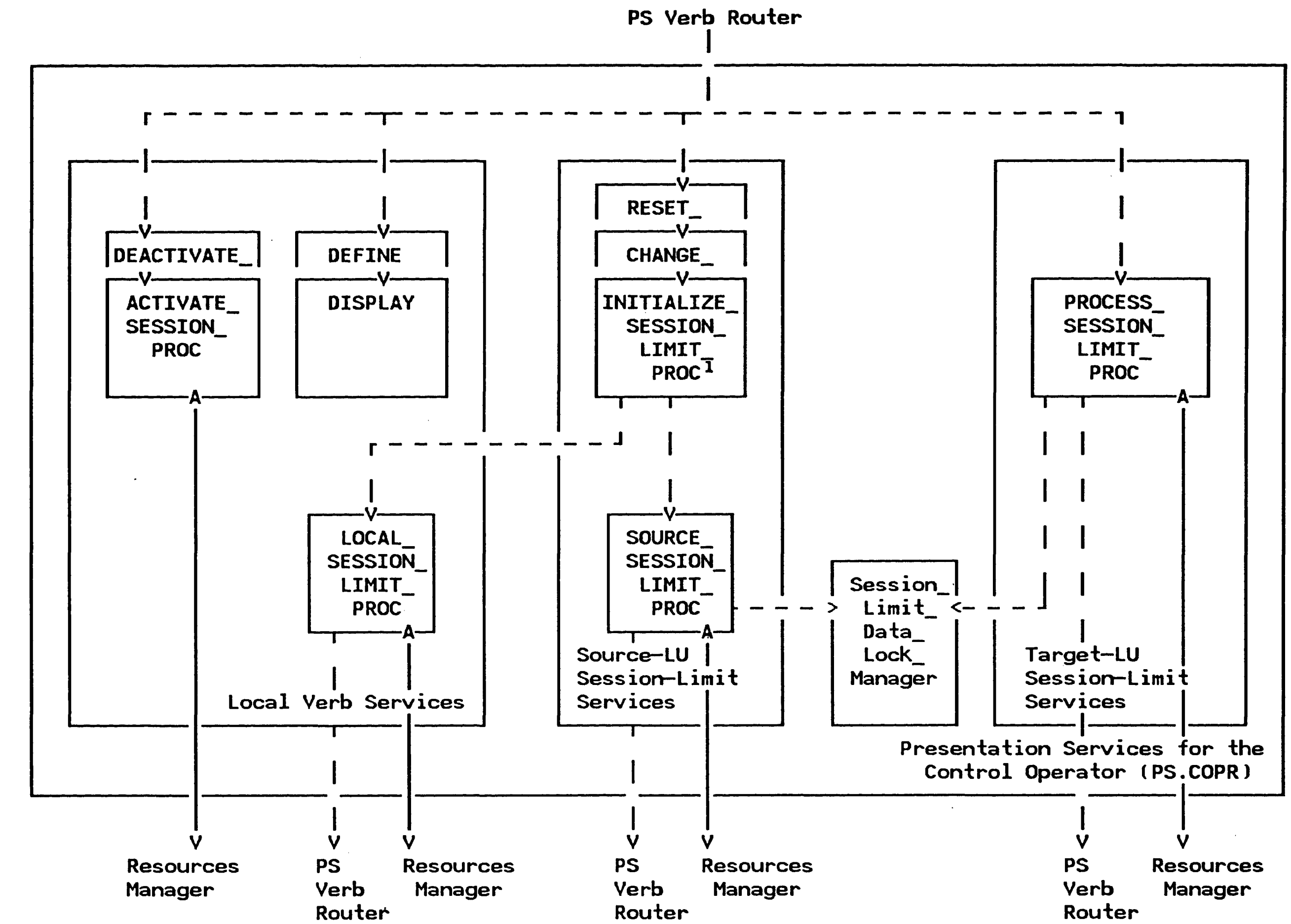
The CNOS service transaction program is that SNA-defined transaction program with transaction-program name (TPN) X'06F1'. It represents the control operator at the target LU. It is invoked by presentation services (PS.INITIALIZE) when the target LU receives the Attach FM header that resulted from the ALLOCATE verb issued by PS.COPR at the source LU.

The CNOS service transaction program performs the following functions.

- It is the target for the ALLOCATE verb issued by the source-LU control-operator transaction program. By being invoked, it completes the activation of the conversation for the CNOS transaction. (The characteristics of the conversation are discussed in section "CNOS Conversation Allocation" on page 5.4-27. The conversation parameters from the Attach FM header are verified by the resources manager and presentation services for conversations before this program is invoked.)
- It issues the PROCESS_SESSION_LIMIT verb before any other processing. Thus, the CNOS service transaction program does not induce any undue delay, e.g., it does not wait on operator input. It also does not affect the values of the negotiable parameters; these values are determined by an algorithm within PS.COPR.

  The CNOS service transaction program passes to PS.COPR a transaction-program-verb data structure specifying the verb type and identifying the return parameters for the CNOS verb. When PS.COPR processing is complete, the CNOS service transaction program is returned the same structure containing a return code indicating success or a failure reason and other parameters identifying the (LU,mode) entry or entries affected by the CNOS command. The PROCESS_SESSION_LIMIT verb does not provide the values of the session-limit parameters to the CNOS service transaction program; these values are available by issuing the DISPLAY verb.

  When control returns from the PROCESS_SESSION_LIMIT verb, the conversation with the source LU has already been deallocated and the session-limit parameters have been updated at the target LU.
- It performs an implementation-defined action to notify its control operator of the activity. For example, it could trigger an interrupt to the LU's control-operator transaction program (see section "Control-Operator Transaction Program" on page 5.4-22) to allow that program to examine the new session-limit parameters and display them for the operator.



[1] These routines are verb handlers for both local- and distributed-function session-limit verbs.

LEGEND:
- - -> Call/return relationship (within a process)
<——> Send/receive relationship (between processes)

Figure 5.4-13. Structure of Presentation Services for the Control Operator

PS.COPR COMPONENTS

Figure 5.4-13 shows the structure of PS.COPR. Its main components are:

- The control-operator-verb router (represented in the figure by the connecting arrows from the PS verb router to the various verb-handler routines)
- A verb handler for each verb (e.g., ACTIVATE_SESSION_PROC, DEFINE_PROC, DISPLAY_PROC, INITIALIZE_SESSION_LIMIT_PROC, PROCESS_SESSION_LIMIT_PROC)
- Common verb-processing routines for groups of verbs:
  - Local session-limit services for single-session mode names and for mode name SNASVCMG (LOCAL_SESSION_LIMIT_PROC)
  - Source-LU CNOS session-limit services (SOURCE_SESSION_LIMIT_PROC)
  - Target-LU CNOS session-limit services (combined with PROCESS_SESSION_LIMIT_PROC)
- The session-limit-data lock manager that controls contention between source-LU session-limit services (running on behalf of a locally-issued verb) and target-LU session-limit services (running on behalf of a remotely-issued verb).

CNOS Verb Router

The control-operator verb router component is the root procedure of PS.COPR. It is invoked by the PS verb router (see Chapter 5.0) when a transaction program issues a control-operator verb. It forms part of the transaction-program process. It is passed the transaction-program-verb structure (TRANSACTION_PGM_VERB) from the PS verb router, and passes this structure on to the corresponding verb handler. Upon regaining control from the verb handler, it returns to the PS verb router.

LOCAL CONTROL-OPERATOR VERB PROCESSING

Local-verb services comprises the verb handlers for two groups of local-function verbs: LU definition verbs and local session-control verbs.

LU DEFINITION VERB PROCESSING

The LU definition verbs include DEFINE and DISPLAY (see Figure 5.4-13). These verbs allow an implementation to define and display the parameters that are configuration dependent (i.e., the maximum number of sessions) and optional capabilities that are supported by the LU, the partner LUs, the MODES, and the transaction programs.

The verb handler checks privilege to determine that the requesting control-operator transaction program has DEFINE or DISPLAY privilege, as appropriate to the verb. It locates the relevant data structure and its containing structures using the keys provided as verb parameters. It provides a return code indicating whether the operation was performed successfully.

The verb handler copies values from control-operator transaction program variables into the LU data structures, or vice versa; the transaction program never has direct access or addressability to the LU data structures.

Verb Parameter Values			Contention Polarity to be Used	
LU_MODE_ SESSION_ LIMIT	MINIMUM_ CONWINNERS_ SOURCE	MINIMUM_ CONWINNERS_ TARGET	Polarity for Locally Activated Sessions	Polarity Negotiation for Remotely Activated Sessions
0	*	*	parameter combination not allowed	
1	0	0	contention winner	accept partner choice
1	1	0	contention winner	contention winner
1	0	1	contention loser	accept partner choice
1	1	1	parameter combination not allowed	
2 or more	*	*	parameter combination not allowed	

LEGEND:
* any value

Figure 5.4-14. Single-Session Contention Polarity Determined by Minimum-Contention-Winner-Limit Parameters

LOCAL SESSION-CONTROL VERB PROCESSING

The session-activation verb handlers (e.g., ACTIVATE_SESSION_PROC) have an interprocess (send/receive) relationship with the resources manager for exchanging the session-activation and -deactivation records.

The local session-limit services component (LOCAL_SESSION_LIMIT_PROC) provides the functions of the session-limit verbs for both single-session mode names and for the parallel-session mode name SNASVCMG, i.e., the SNA-defined mode name used by CNOS. (Even though SNASVCMG-mode-name sessions are parallel sessions, local verbs are used to initialize--to fixed session limits--and to reset the SNASVCMG mode name, because a session with this mode name must be activated before the first CNOS command and reply can be sent.) This component has an interprocess (send/receive) relationship with the resources manager to notify RM of limits changes.

INITIALIZE SESSION LIMIT: When this verb is issued for a single-session mode name or for mode name SNASVCMG, local session-limit services checks session-limit constraints and sets the (LU,mode) session limit at the local LU. The partner LU does not participate in setting the limits. Local session-limit services sends a change-sessions notification to the resources manager so that the resources manager may request activation of the allowed sessions according to its session-activation algorithm.

For single-session mode names, local session-limit services also determines the contention polarity to be used when a session is activated by the local LU and determines the contention-polarity negotiation rule to be used when a session is activated by a partner LU. It determines these settings from the minimum-contention-winner limit parameters of the verb, as specified in Figure 5.4-14 on page 5.4-24.

In the figure, the first three columns list possible combinations of verb parameter values. The next column (locally activated sessions) specifies the corresponding contention-polarity choice that will be sent in a BIND RU issued by the local LU; the partner LU may negotiate contention-winner to contention-loser (i.e., make the partner LU the contention winner), but not the reverse. The next column (remotely activated sessions) specifies the contention-polarity that will be sent in the response issued by the local LU to a BIND from a partner LU. The local LU may change a received contention-loser into a contention-winner, but not the reverse. The last two columns also indicate those combinations of verb parameter values that are invalid with single-session mode names.

For the parallel-session mode name SNASVCMG, the verb parameters have their usual interpretation, but the only accepted values are: (LU,mode) session limit = 2, minimum contention-winner limit (source) = 1, minimum contention-winner limit (target) = 1.

RESET SESSION LIMIT: When this verb is issued for a single-session mode name or for mode name SNASVCMG, local session-limit services checks session-limit constraints and sets the (LU,mode) session limit to 0 at the local LU. It also sets the drain specification for the local and remote LUs. The partner LU does not participate in setting these limits. Local session limit services sends a change-sessions notification to the resources manager so that the resources manager will deactivate the specified sessions according to its drain and session-deactivation algorithms.

For mode name SNASVCMG, local session-limit services also verifies that all other mode names for the specified partner LU are fully reset, i.e., have (LU,mode) session limit = 0 and drain state NO. If so, it sets the session limits for mode name SNASVCMG to 0 and notifies RM to deactivate the SNASVCMG-mode-name sessions; otherwise, it does not change the limits but sets the appropriate return code.

ACTIVATE SESSION: For this verb, if the TP has session control privilege, the verb handler sends a session-activation request to the resources manager, and receives a reply record indicating whether the session was successfully activated.

DEACTIVATE SESSION: For this verb, if the TP has session control privilege, PS.COPR sends a session-deactivation request to the resources manager; the resources manager sends no reply, as session deactivation is assured.

SESSION-LIMIT SERVICES AT THE SOURCE LU

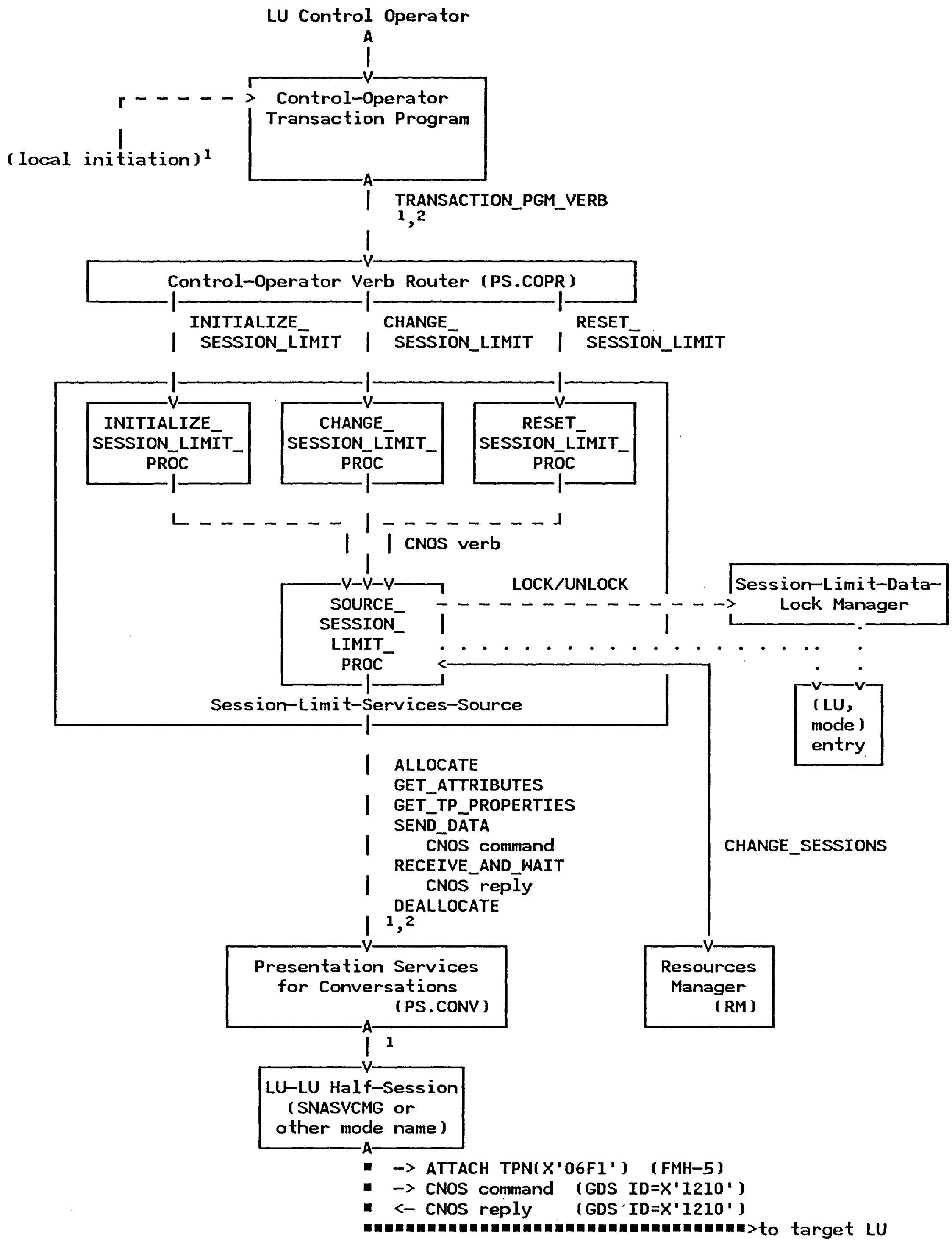
Source-LU session-limit services (SSLS) processes CNOS verbs issued at the source LU. It forms a part of the source-LU transaction-program process that includes the control-operator transaction program. It is invoked via the presentation services (PS) verb router and PS.COPR when the control-operator transaction program issues a CNOS verb, and returns to the control-operator transaction program via the routers upon completing processing.

SSLS interacts with other LU components as follows (see Figure 5.4-15).

A verb-handling routine corresponding to the specific verb (INITIALIZE_SESSION_LIMIT_PROC, CHANGE_SESSION_LIMIT_PROC, or RESET_SESSION_LIMIT_PROC), receives the verb parameters from the PS.COPR router. It then invokes the common session-limit services routine SOURCE_SESSION_LIMIT_PROC. It is returned the same structure with a return code, which it passes back to the PS.COPR router.

SOURCE_SESSION_LIMIT_PROC is passed the CNOS verb parameters which it returns updated with a return code when its processing is complete. It performs the remainder of SSLS processing, as follows.

- It verifies that the program issuing the verb is privileged to issue CNOS verbs.
- It allocates a conversation with the target.
- Using that conversation, it sends a CNOS command record and receives a CNOS reply record
- It invokes the session-limit-data-lock manager (see "Session-Limit Data Lock Manager" on page 5.4-30) to prevent simultaneous updating of the same (LU,mode) entry, or entries, and to resolve races.
- It updates the (LU,mode) entry with the accepted session-limit parameters.
- If necessary, it notifies the resources manager to increase or decrease the current number of sessions.



LEGEND:

- – – –> Call/return relationship (within a process)
- <———> Send/receive relationship (between processes)
- .....> Access to shared data (within the LU)
- <■■■■> Transaction program interaction (between LUs)

[1] See Chapter 5.1 for these interactions.
[2] PS router detail has been omitted.

Figure 5.4-15. Source-LU Component Interactions for CNOS

Privilege Checking

SSLS examines the source-LU's transaction program list to determine whether the control-operator transaction program is authorized to issue CNOS verbs, i.e., whether it has change-number-of-sessions privilege. If not, SSLS causes the verb to fail.

(Since the target transaction program has a privileged transaction-program name, i.e., TPN less than X'40', presentation services for conversations also verifies, by checking the transaction-program list at the source LU, that the transaction program issuing the ALLOCATE is allowed to invoke privileged programs.)

CNOS Conversation Allocation

SSLS allocates a conversation with the target LU to exchange the CNOS command and reply. The conversation requires only conversation verbs in the base set, but an implementation may use verbs and parameters from the locally-supported "performance" option sets that do not require remote support (see SNA Transaction Programmer's Reference Manual for LU Type 6.2).

The following subsections discuss the allocation parameters for the conversation.

LU name: SSLS uses the target LU name supplied by the CNOS verb.

Mode name: SSLS uses an implementation-defined algorithm to select a mode name for the CNOS conversation; for example, the algorithm can select a mode name for which a session is currently active and available. If no session is available on any other implementation-selected mode name, SLSS uses the SNA-defined mode name SNASVCMG. It also uses SNASVCMG for the first CNOS verb issued by the LU, i.e., when no sessions are active for other mode names and the session limits for all mode names (except SNASVCMG) are all 0.

(The operator previously initializes the session limits for mode name SNASVCMG to MAX_SESSIONS(2), MIN_CONWINNERS_SOURCE(1), and MIN_CONWINNERS_TARGET(1), so that the source LU may always succeed in activating one contention winner session to send the CNOS command and reply.)

Type: Basic Conversation

Transaction Program Name: SSLS establishes the conversation with the CNOS service transaction program, whose SNA-defined transaction program name (TPN) is X'06F1', at the target LU.

Security: The CNOS conversation uses SECURITY(NONE).

Synchronization Level: The CNOS conversation uses SYNC_LEVEL(NONE).

Recovery Level: The CNOS conversation uses RECOVERY_LEVEL(NONE).

Program Initialization Parameters: The CNOS conversation does not use program initialization parameter data, i.e., it uses PIP(NO).

GDS Variable

SSLS builds a CNOS command containing the verb and parameter information passed from the CNOS service transaction program and sends it to the target transaction program. The Change Number of Sessions GDS variable and the CNOS command and reply are described in SNA Formats. It receives from the target transaction program a similar CNOS reply containing a reply code that indicates either that the command was accepted or the reason for its rejection.

CNOS Record Flows

SSLS generates a conversation between the source-LU and the target-LU transaction programs. The sequence of conversation verbs issued by SSLS, and the complementary verbs issued by the partner program SESSION_LIMIT_SERVICES_TARGET, are shown in Figure 5.4-3 on page 5.4-9.

Errors

SSLS analyzes the CNOS verb parameters for transaction program errors, checks the return codes from conversation verbs for conversation errors such as session failure or protocol violation, and analyzes the CNOS reply for target-detected errors or changes to negotiable parameters, and determines the proper return code for the CNOS verb.

If conversation failure (session outage) occurs, the source LU retries the CNOS command as described in "Recovery from Conversation Failure" on page 5.4-20.

Update (LU,mode) Entry

If the command and reply exchange is completed without error, SSLS updates the session-limit parameters for the specified (LU,mode) entry using the new values of LU_MODE_SESSION_LIMIT, MIN_CONWINNERS_SOURCE, MIN_CONWINNERS_TARGET, RESPONSIBLE, and DRAIN_TARGET from the reply record. If the command specifies MODE_NAME(ALL), the limits for all mode names defined for the specified LU name, except the SNA-defined mode name SNASVCMG, are updated. SSLS then invokes the session-limit-data-lock manager to unlock the entries it locked (see "Session-Limit Data Lock Manager" on page 5.4-30).

The new limits are enforced by the resources manager (see "Chapter 3. LU Resources Manager") and by the session manager (see "Chapter 4. LU Session Manager").

Request Changes in Session Count

If the CNOS command action is Set, or if it designates the source LU as responsible for session deactivation, SSLS issues a CHANGE_SESSIONS request, identifying the affected LU name and mode names, to the resources manager (RM). If MODE_NAME(ALL) is specified, SLSS sends a separate CHANGE_SESSIONS request for each mode name except mode name SNASVCMG.

The CHANGE_SESSIONS request notifies RM that the session limit parameters have changed and that, as a consequence, RM may make changes to the number of sessions. RM determines the actual changes to be made to the session count and issues appropriate requests to the session manager to activate or deactivate sessions.

Return to the Transaction Program

When the above functions are completed, SSLS returns to the control-operator transaction program, passing back the appropriate return code in the transaction-program-verb structure.

SESSION-LIMIT SERVICES AT THE TARGET LU

Target-LU session-limit services (TSLS) processes the CNOS verbs issued at the target LU. It functions in a manner complementary to SSLS (see "Session-Limit Services at the Source LU" on page 5.4-25). It forms a part of the target-LU transaction-program process that includes the CNOS service transaction program. It is invoked via the presentation services (PS) verb router and the PS.COPR router when the CNOS service transaction program issues the PROCESS_SESSION_LIMIT verb; it returns to the CNOS service transaction program upon completion of processing.

TSLS interacts with other LU components as follows (see Figure 5.4-16).

- It receives the transaction-program-verb structure representing the PROCESS_SESSION_LIMIT verb

  When its processing is complete, it returns to the CNOS service transaction program, passing back the transaction-program-verb structure updated with a return code and the identity of the affected (LU,mode) entries. (The latter may be used by the implementation to inform the control operator of the changes.)

- It determines whether the issuing transaction program is the CNOS service transaction program (TPN=X'06F1') and has the change-number-of-sessions privilege. If not, TSLS abnormally terminates the transaction program, which causes the LU to issue DEALLOCATE TYPE(ABEND) on the conversation.

- It communicates with SSLS at the source LU, using the conversation with which the CNOS service transaction program was attached, by issuing conversation verbs to presentation services for conversations.

- It receives a CNOS command from the source LU, changes the source LU's requested session-limit parameters to values acceptable to the target LU, if necessary, and sends a CNOS reply, with the same format, back to the source LU.

- It invokes the session-limit-data-lock manager (see "Session-Limit Data Lock Manager" on page 5.4-30) to prevent simultaneous updating of any (LU,mode) entry.

- It updates the affected (LU,mode) entries.

- If necessary, it notifies the resources manager to increase or decrease the current number of sessions.

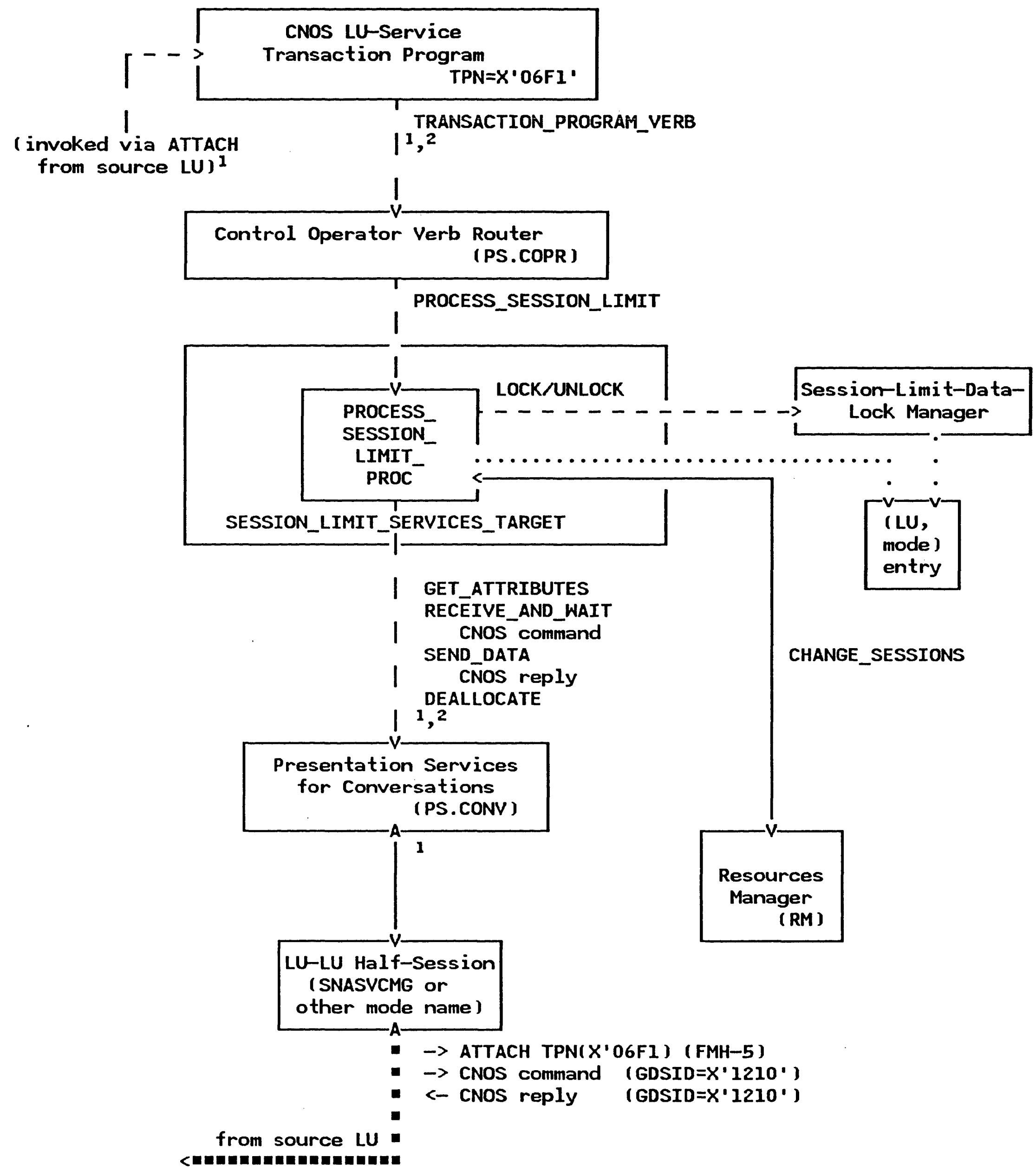
CNOS Reply

TSLS receives from the source-LU transaction program a CNOS command record containing the verb and parameter information passed from the control-operator transaction program. It builds a similar CNOS reply record containing the acceptable values of the negotiable session-limit parameters--see "Session-Limit Parameter Negotiation" on page 5.4-28--and a reply code, which either indicates that the command was accepted or gives the reason for its rejection, and sends it to the source LU.

Session-Limit Parameter Negotiation

TSLS executes an implementation-determined algorithm to accept or modify the negotiable session-limit parameters received from the source LU, subject to the negotiation rules given below. It sets the Reply Modifier field in the CNOS reply to indicate whether all the parameters were accepted as received or whether any were negotiated to new values, and sends it with the received or modified values to the source LU in the CNOS reply. (The source LU accepts any modified values that satisfy the negotiation rules.)

The negotiation rules are as follows. (In the formulas, variables prefixed with C_ refer to values of verb parameters specified by the source LU in the CNOS command record; variables prefixed with R_ refer to values of these parameters as modified by the target LU and returned in the CNOS reply record.)



LEGEND:
- - -> Call/return relationship (within a process)
<——> Send/receive relationship (between processes)
.....> Access to shared data (within the LU)
<■■■■> Transaction program interaction (between LUs)

[1] See Chapter 5.1 for these interactions.
[2] PS router detail has been omitted.

Figure 5.4-16. Target-LU Component Interactions for CNOS

If the command action is Set (INITIALIZE_ or CHANGE_SESSION_LIMIT verb issued):

- If the current (LU,mode) session count is 0, then, based on an implementation-defined decision, the LU may refuse to accept the command by returning an abnormal reply with reply-modifier value abnormal--(LU,mode) session limit is 0. The LU also has the option to issue DEALLOCATE TYPE(ABEND) for the conversation. Both LUs then ignore the session-limit parameters of the reply; they do not change the current session-limit parameters in the (LU,mode) entry.

- The target LU may decrease LU_MODE_SESSION_LIMIT to a lower number of sessions, but not to 0, i.e., the new value satisfies:

  0 < R_LU_MODE_SESSION_LIMIT ≤

  C_LU_MODE_SESSION_LIMIT.

- If the proposed source contention winners (C_MIN_CONWINNERS_SOURCE) exceeds R_LU_MODE_SESSION_LIMIT/2, the target LU may change MIN_CONWINNERS_SOURCE to any lower value not less than R_LU_MODE_SESSION_LIMIT/2 rounded downward, i.e., the new value satisfies:

  C_MIN_CONWINNERS_SOURCE ≥

  R_MIN_CONWINNERS_SOURCE ≥

  MIN(C_MIN_CONWINNERS_SOURCE, R_LU_MODE_SESSION_LIMIT/2).

- The target LU may change its own minimum contention-winner limit (R_MIN_CONWINNERS_TARGET) to any value not exceeding the difference between the total session limit and MIN_CONWINNERS_SOURCE, i.e., the new value satisfies:

  0 ≤ R_MIN_CONWINNERS_TARGET ≤

  (R_LU_MODE_SESSION_LIMIT - R_MIN_CONWINNERS_SOURCE).

- The target LU may change RESPONSIBLE to SOURCE.

If the command action is Close for only one mode name (RESET_SESSION_LIMIT (MODE_NAME(ONE,...) issued):

- If the (LU,mode) session count is 0 and the current drain state is NO, then, based on an implementation-defined decision, the target LU may refuse to accept the command by returning an abnormal reply with reply modifier abnormal--(LU,mode) session limit is 0. Both LUs then ignore the session-limit parameters of the reply; they do not change the current session-limit parameters in the (LU,mode) entry.

- The target LU may change RESPONSIBLE to SOURCE.

- The target LU may change its own drain action (DRAIN_TARGET) from YES to NO.

- The target LU does not change DRAIN_SOURCE.

If the command action is Close for all mode names (RESET_SESSION_LIMIT (MODE_NAME(ALL) issued):

- If the (LU,mode) session count is 0 and the current drain state is NO for all mode names with the partner LU, then, based on an implementation-defined decision, the target LU may refuse to accept the command by returning an abnormal reply with reply modifier abnormal--(LU,mode) session limit is 0. Both LUs then ignore the session-limit parameters of the reply; they do not change the current session-limit parameters in the (LU,mode) entry.

- The target LU may change RESPONSIBLE to SOURCE. If so, it changes all mode names not already at SESSION_LIMIT = 0 to the same (SOURCE) responsibility.

- The target LU does not send a changed value for DRAIN_TARGET in the reply, but echoes the value received. Nevertheless, if the command specifies DRAIN_TARGET(YES), and the current session limit is not zero, the target LU may set its local drain state for any mode names to either YES or NO, regardless of the previous drain state. If the current session limit is already zero and the drain state is no, the drain state for that mode is left unchanged.

- The target LU does not change DRAIN_SOURCE.

Errors

If TSLS detects a condition that precludes performing the nominal action (e.g., a race condition or unrecognized mode name), but that does not violate architectural rules, it sends an abnormal reply with the appropriate reply modifier (see SNA Formats for reply-modifier codes).

If it detects an invalid command from the source LU, e.g., undefined or disallowed parameter values, it treats this as a protocol violation. TSLS does not change the CNOS parameters or send a reply, but instead issues DEALLOCATE TYPE(ABEND). TSLS also reports any errors detected to the CNOS service transaction program via the transaction-program-verb structure.

Other Interactions

Other TSLS interactions are similar to the corresponding interactions of SSLS.

SESSION-LIMIT DATA LOCK MANAGER

Locking the (LU,mode) Entry

The session-limit services routines invoke a shared component, SESSION_LIMIT_DATA_LOCK_MANAGER (SLDLM), to prevent simultaneous access to an (LU,mode) entry, to detect races, and to resolve double-failure race conditions, as described in "CNOS Race Resolution" on page 5.4-14.

SLDLM is a shared routine, invoked from both SSLS and TSLS, that maintains the session-limit data lock. A session-limit data lock exists for each (LU,mode) entry. It is in one of the following states:

UNLOCKED: No CNOS component is currently using the (LU,mode) entry. The lock is reset to this state whenever the process that locked it completes processing.

LOCKED_BY_SOURCE: SSLS has locked the (LU,mode) entry to process a CNOS command issued at the local LU. The lock had previously been in UNLOCKED state.

LOCKED_BY_TARGET: TSLS has locked the (LU,mode) entry to process a CNOS command issued at a remote LU. The lock had previously been in UNLOCKED state.

LOCK_DENIED: While the lock was in LOCKED_BY_SOURCE state, TSLS attempted to lock it on behalf of a remotely-issued verb. TSLS was refused.

This state allows SSLS to determine whether a double-failure race occurred.

PS_COPR

FUNCTION:	This procedure receives all control-operator verbs issued by the transaction program and routes the input to the appropriate procedure for processing. It is invoked by, and returns to, the presentation-services verb router and forms part of the transaction-program process.
INPUT:	CNOS verb parameters-- received from caller, updated by called procedures
OUTPUT:	Updated return code and verb-specific returned parameters

Referenced procedures, FSMs, and data structures:

INITIALIZE_SESSION_LIMIT_PROC	page 5.4-33
CHANGE_SESSION_LIMIT_PROC	page 5.4-35
RESET_SESSION_LIMIT_PROC	page 5.4-34
PROCESS_SESSION_LIMIT_PROC	page 5.4-57
ACTIVATE_SESSION_PROC	page 5.4-36
DEACTIVATE_SESSION_PROC	page 5.4-37
DEFINE_PROC	page 5.4-38
DISPLAY_PROC	page 5.4-39
DELETE_PROC	page 5.4-40

```
Select based on type of verb parameters:
   When INITIALIZE_SESSION_LIMIT
      Call INITIALIZE_SESSION_LIMIT_PROC with the verb parameters (page 5.4-33).
   When CHANGE_SESSION_LIMIT
      Call CHANGE_SESSION_LIMIT_PROC with the verb parameters (page 5.4-35).
   When RESET_SESSION_LIMIT
      Call RESET_SESSION_LIMIT_PROC with the verb parameters (page 5.4-34).
   When PROCESS_SESSION_LIMIT
      Call PROCESS_SESSION_LIMIT_PROC with the verb parameters (page 5.4-57).
   When DEACTIVATE_SESSION
      Call DEACTIVATE_SESSION_PROC with the verb parameters  (page 5.4-37).
   When ACTIVATE_SESSION
      Call ACTIVATE_SESSION_PROC with the verb parameters (page 5.4-36).
   When DEFINE_LOCAL_LU, DEFINE_REMOTE_LU, DEFINE_MODE, or DEFINE_TP
      Call DEFINE_PROC with the verb parameters (page 5.4-38).
   When DISPLAY_LOCAL_LU, DISPLAY_REMOTE_LU, DISPLAY_MODE, or DISPLAY_TP
      Call DISPLAY_PROC with the verb parameters (page 5.4-39).
   When DELETE
      Call DELETE_PROC with the verb parameters (page 5.4-40).
```

INITIALIZE_SESSION_LIMIT_PROC

FUNCTION:	This procedure is called by PS_COPR, the control-operator verb router, when a transaction program issues an INITIALIZE_SESSION_LIMIT verb. It determines the connection type (single or parallel). If the connection is single-session or the mode name is SNASVCMG, it passes the CNOS verb parameters to LOCAL_SESSION_LIMIT_PROC; if the connection is parallel-session, it passes the CNOS verb parameters to SOURCE_SESSION_LIMIT_PROC. It passes the return code to the original caller.
INPUT:	INITIALIZE_SESSION_LIMIT verb parameters from caller; CNOS RETURN_CODE from LOCAL_ or SOURCE_SESSION_LIMIT_PROC
OUTPUT:	RETURN_CODE of INITIALIZE_SESSION_LIMIT to caller

Referenced procedures, FSMs, and data structures:

SOURCE_SESSION_LIMIT_PROC page 5.4-45
LOCAL_SESSION_LIMIT_PROC page 5.4-41

If this transaction program is authorized to issue the CNOS verb then
   Using the LUCB, determine the type of sessions possible with
    the partner LU, either single or parallel.

> For parallel session connections, an LU may elect not to expose the MIN_CONWINNERS_TARGET parameter at the control-operator protocol boundary. In this case, the implementation may choose any value that satisfies the description of this parameter in SNA Transaction Programmer's Reference Manual for LU Type 6.2.

   If the specified LU is not defined as a partner LU for this LU then
      Set the CNOS RETURN_CODE to PARAMETER_ERROR.

   Else
      If the type of connection is parallel-sessions
       and the mode name is not SNASVCMG then
         Call SOURCE_SESSION_LIMIT_PROC (page 5.4-45),
          with the verb parameters, to begin the negotiation phase of the CNOS
          process.

      Else (local control-operator verb)
         Call LOCAL_SESSION_LIMIT_PROC (page 5.4-41),
          with the verb parameters, to perform the CNOS action solely at the
          local LU.
Else
   Set CNOS RETURN_CODE to PROGRAM_PARAMETER_CHECK.

RESET_SESSION_LIMIT_PROC

FUNCTION:	This procedure is called by PS_COPR, the control-operator verb router, when a transaction program issues a RESET_SESSION_LIMIT verb. It determines the connection type (single or parallel). If the connection is single-session or the mode name is SNASVCMG, it passes the CNOS verb parameters to LOCAL_SESSION_LIMIT_PROC; if the connection is parallel-session, it passes the CNOS verb parameters to SOURCE_SESSION_LIMIT_PROC. It passes the return code to the original caller.
INPUT:	RESET_SESSION_LIMIT verb parameters from caller; CNOS RETURN_CODE from LOCAL_ or SOURCE_SESSION_LIMIT_PROC
OUTPUT:	RETURN_CODE of RESET_SESSION_LIMIT verb to caller

Referenced procedures, FSMs, and data structures:

SOURCE_SESSION_LIMIT_PROC page 5.4-45
LOCAL_SESSION_LIMIT_PROC page 5.4-41
CHANGE_ACTION page 5.4-43


```
If this transaction program is authorized to issue the CNOS verb then
   Using the LUCB, determine the type of sessions possible with
    the partner LU, either single or parallel.
```

> For parallel-session connections, an LU may elect not to expose the DRAIN_TARGET, and RESPONSIBLE parameters at the control-operator protocol boundary. In this case, the implementation provides default values for these parameters consistent with the description on page 5.4-21.
>
> For single-session connections, the RESPONSIBLE parameter on the verb is not used. It is forced to SOURCE.
>
> For the SNA-defined mode name, SNASVCMG, the DRAIN_SOURCE, DRAIN_TARGET, and RESPONSIBLE parameters on the verb are not used. They are forced to NO, NO, SOURCE, respectively.

```
   If the specified LU is not defined as a partner LU for this LU then
      Set the CNOS RETURN_CODE to PARAMETER_ERROR.

   Else
      If the type of connection is parallel-sessions
       and the mode name is not SNASVCMG then
         Call SOURCE_SESSION_LIMIT_PROC (page 5.4-45),
          with the verb parameters, to begin the negotiation phase of the CNOS process.
         If FORCE = YES is specified on the RESET_SESSION_LIMIT verb then
            If the CNOS return code indicates ALLOCATION_ERROR-ALLOCATION_FAILURE_NO_RETRY,
             LU_MODE_SESSION_LIMIT_CLOSED, RESOURCE_FAILURE_NO_RETRY or
             UNRECOGNIZED_MODE_NAME then
              Change RESPONSIBLE to SOURCE.
              Call CHANGE_ACTION (page 5.4-43) with the CNOS request to
               update the limits in the MODE structure(s) for the source LU and notify RM.
              Set the CNOS return code to OK-FORCED.

      Else (local control-operator verb)
         Call LOCAL_SESSION_LIMIT_PROC (page 5.4-41),
          with the verb parameters, to perform the CNOS action solely at the local LU.

Else
   Set CNOS RETURN_CODE to PROGRAM_PARAMETER_CHECK.
```

CHANGE_SESSION_LIMIT_PROC

FUNCTION:	This procedure is called by PS_COPR, the control-operator verb router, when a transaction program issues a CHANGE_SESSION_LIMIT verb. It passes the CNOS verb parameters to SOURCE_SESSION_LIMIT_PROC and passes the return code to the original caller.
INPUT:	CHANGE_SESSION_LIMIT parameters from caller; CNOS RETURN_CODE from SOURCE_SESSION_LIMIT_PROC
OUTPUT:	RETURN_CODE of CHANGE_SESSION_LIMIT to caller

Referenced procedures, FSMs, and data structures:
SOURCE_SESSION_LIMIT_PROC page 5.4-45

If the control-operator transaction program, at the source LU, is not authorized to issue the CNOS verb then
Set CNOS RETURN_CODE to PROGRAM_PARAMETER_CHECK.

Else
Using the LUCB, determine the type of sessions possible with the partner LU, either single or parallel.

> An LU might elect not to expose the RESPONSIBLE and MIN_CONWINNERS_TARGET parameters at the control-operator protocol boundary. In this case, the implementation provides default values for these parameters consistent with the description on page 5.4-21 and the parameter specification in SNA Transaction Programmer's Reference Manual for LU Type 6.2.

If the specified LU is not defined as a partner for this LU then
Set the CNOS RETURN_CODE to PARAMETER_ERROR.

Else
If the type of connection is parallel-sessions and the mode name is not SNASCVMG then
Call SOURCE_SESSION_LIMIT_PROC (page 5.4-45), with the verb parameters, to begin the negotiation phase of the CNOS process.

Else
Set CNOS RETURN_CODE to PROGRAM_PARAMETER_CHECK.

ACTIVATE_SESSION_PROC

FUNCTION:	This procedure is called by PS_COPR, the control-operator-verb router, when a transaction program issues an ACTIVATE_SESSION verb. It sends an RM_ACTIVATE_SESSION request to RM to activate a session, and receives the reply indicating whether the session was activated.
INPUT:	The CNOS verb (ACTIVATE_SESSION), the reply from RM (RM_ACTIVATE_SESSION)
OUTPUT:	The request to RM (RM_ACTIVATE_SESSION), RETURN_CODE of ACTIVATE_SESSION verb
NOTE:	This procedure has addressability to RM via PS_PROCESS_DATA.LU_ID.

Referenced procedures, FSMs, and data structures:

RM page 3-19
PS_PROCESS_DATA page 5.0-24
RM_ACTIVATE_SESSION page A-16
RM_SESSION_ACTIVATED page A-22

```
Verify that the verb parameters specified satisfy the
 parameter values for the ACTIVATE_SESSION verb described in
 SNA Transaction Programmer's Reference Manual for LU Type 6.2.

Select based on the result of parameter verification:
   When transaction program is not authorized to issue ACTIVATE_SESSION
      Set CNOS RETURN_CODE to PROGRAM_PARAMETER_CHECK.
   When a parameter error is identified
      Set the CNOS RETURN_CODE to PARAMETER_ERROR.
   When all parameters are correct
      Create an RM_ACTIVATE_SESSION request record.
      Set RM_ACTIVATE_SESSION.TCB_ID to PS_PROCESS_DATA.TCB_ID to identify
       the transaction control block describing this instance of PS.
      Set RM_ACTIVATE_SESSION.LU_NAME to the LU name specified in the CNOS verb.
      Set RM_ACTIVATE_SESSION.MODE_NAME to the mode name specified in the CNOS verb.
      Send RM_ACTIVATE_SESSION request to RM.
      Receive RM_SESSION_ACTIVATED reply from RM.
      Set CNOS RETURN_CODE according to the return code in the
       RM_SESSION_ACTIVATED reply received from RM.
      IF this is a single session and conwinner received (as it was requested) then
         Set the secondary code to OK.AS_SPECIFIED.
      Else
         Set the secondary code to OK.AS_NEGOTIATED.
Destroy the RM_SESSION_ACTIVATED record.
```

DEACTIVATE_SESSION_PROC

FUNCTION:	This procedure is called by PS_COPR, the control-operator-verb router, when a transaction program issues a DEACTIVATE_SESSION verb. It sends an RM_DEACTIVATE_SESSION request to RM to deactivate a session. The calling TP may be given control back before the session is actually deactivated.
INPUT:	The DEACTIVATE_SESSION verb
OUTPUT:	Request to RM (RM_DEACTIVATE_SESSION), RETURN_CODE of DEACTIVATE_SESSION verb
NOTE:	This procedure has addressability to RM via PS_PROCESS_DATA.LU_ID.

Referenced procedures, FSMs, and data structures:

RM	page 3-19
PS_PROCESS_DATA	page 5.0-24
RM_DEACTIVATE_SESSION	page A-17

```
Verify that the verb parameters specified satisfy the
 parameter values for the ACTIVATE_SESSION verb described in
 SNA Transaction Programmer's Reference Manual for LU Type 6.2.
If transaction program is authorized to issue DEACTIVATE_SESSION then
   Set the CNOS RETURN_CODE to OK.
   Create a RM_DEACTIVATE_SESSION request record.
   Set RM_DEACTIVATE_SESSION.TCB_ID to PS_PROCESS_DATA.TCB_ID to identify
    the transaction control block describing this instance of PS.
   Set RM_DEACTIVATE_SESSION.SESSION_ID to the SESSION_ID specified in the CNOS verb.
   Set RM_DEACTIVATE_SESSION.TYPE to the TYPE specified in the CNOS verb.
   Send RM_DEACTIVATE_SESSION request to RM.
Else
   Set CNOS RETURN_CODE to PROGRAM_PARAMETER_CHECK.
```

DEFINE_PROC

FUNCTION:	This procedure is called by PS_COPR, the control-operator-verb router, when a transaction program issues any of the DEFINE verbs (DEFINE_LOCAL_LU, DEFINE_REMOTE_LU, DEFINE_MODE, or DEFINE_TP). It is used to initialize or modify attributes of the LUCB, PARTNER_LU, MODE, and TRANSACTION_PROGRAM data structures.
INPUT:	The DEFINE verb parameters
OUTPUT:	The attributes of the data structure are defined with the specified values.
NOTE:	This verb may be used to define any other attributes of the LU that are meaningful for a given implementation.

Referenced procedures, FSMs, and data structures:

LUCB page A-1
PARTNER_LU page A-2
MODE page A-3
TRANSACTION_PROGRAM page A-5

Verify that the verb parameters specified satisfy the parameter values for the DEFINE verb in SNA Transaction Programmer's Reference Manual for LU Type 6.2.

If an ABEND condition is identified then
  Set CNOS RETURN_CODE to PROGRAM_PARAMETER_CHECK.

Else

The parameters specified are all valid attributes of the LUCB, PARTNER_LU, MODE, or TRANSACTION_PROGRAM data structure.

Assign values to the attributes of the data structure according to those specified on the DEFINE verb.

DISPLAY_PROC

FUNCTION:	This procedure is called by PS_COPR, the control-operator-verb router, when a transaction program issues any of the DISPLAY verbs (DISPLAY_LOCAL_LU, DISPLAY_REMOTE_LU, DISPLAY_MODE, or DISPLAY_TP). It is used to display attributes of the LUCB, PARTNER_LU, MODE, and TRANSACTION_PROGRAM data structures.
INPUT:	The DISPLAY verb parameters
OUTPUT:	The specified attributes of the data structure are displayed for the user.
NOTE:	This verb may be used to display any other attributes of the LU that are meaningful for a given implementation.

Referenced procedures, FSMs, and data structures:

LUCB page A-1
PARTNER_LU page A-2
MODE page A-3
TRANSACTION_PROGRAM page A-5

Verify that the verb parameters specified satisfy the parameter values for the DISPLAY verb in <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>.

If an ABEND condition is identified then
  Set CNOS RETURN_CODE to PROGRAM_PARAMETER_CHECK.

Else

> The parameters specified are all valid attributes of the LUCB, PARTNER_LU, MODE MODE, or TRANSACTION_PROGRAM data structure.

  Display the requested LUCB, PARTNER_LU, MODE, or TRANSACTION_PROGRAM attributes as they are currently defined.

DELETE_PROC

FUNCTION:	This procedure is called by PS_COPR, the control-operator-verb router, when a transaction program issues a DELETE verb. It is used to delete attributes of the LUCB, PARTNER_LU, MODE, and TRANSACTION_PROGRAM data structures.
INPUT:	The DELETE verb parameters
OUTPUT:	The data structure attributes are deleted.
NOTE:	This verb may be used to delete any other attributes that are meaningful for a given implementation.

Referenced procedures, FSMs, and data structures:

LUCB	page A-1
PARTNER_LU	page A-2
MODE	page A-3
TRANSACTION_PROGRAM	page A-5

Verify that the verb parameters specified satisfy the parameter values for the DELETE verb in <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>.

If an ABEND condition is identified then
   Set CNOS RETURN_CODE to PROGRAM_PARAMETER_CHECK.

Else

> The parameters specified are all valid attributes of the LUCB, PARTNER_LU, MODE, or TRANSACTION_PROGRAM data structure.

   Delete the LUCB, PARTNER_LU, MODE, or TRANSACTION_PROGRAM attributes as specified on the DELETE verb.

LOCAL_SESSION_LIMIT_PROC

FUNCTION:	This procedure is invoked by either of the following verb-specific CNOS procedures: INITIALIZE_SESSION_LIMIT, RESET_SESSION_LIMIT. It processes CNOS control-operator verbs that affect only the local LU: INITIALIZE_ and RESET_SESSION_LIMIT for single-session connections and for mode name SNASVCMG.
INPUT:	The CNOS source LU verb parameters from the calling procedure
OUTPUT:	Return code for the CNOS verb (CNOS RETURN_CODE)
NOTE:	This procedure read-locks the MODE for the entire procedure.

Referenced procedures, FSMs, and data structures:

Using the LUCB, determine the type of session possible with the partner LU, either single or parallel.

If the type of connection is single session then
  Call LOCAL_VERB_PARAMETER_CHECK (page 5.4-42), with the CNOS verb parameters, to verify the verb parameters.

Else
  Call SVCMG_VERB_PARAMETER_CHECK (page 5.4-43), with the CNOS verb parameters, to perform the appropriate parameter checks.

If the check found no errors then
  Call CHANGE_ACTION (page 5.4-43), with the CNOS verb parameters, to change the session limits at the source LU according to the parameters specified.

LOCAL_VERB_PARAMETER_CHECK

FUNCTION:	This procedure performs validity checks on a CNOS verb for single-session connections, and it returns the CNOS-verb RETURN_CODE for any error detected.
INPUT:	The CNOS source LU verb parameters, PARTNER_LU_LIST, and MODE_LIST
OUTPUT:	CNOS verb RETURN_CODE value

Referenced procedures, FSMs, and data structures:

LUCB	page A-1
MODE	page A-3

Verify that the specified verb parameters satisfy the single-session parameter values as described for this verb in <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>.

> Attributes of the mode are verified against fields in the appropriate MODE structure for the specified PARTNER_LU.

Select based on result of parameter verification:
   When all parameters are correct
      Set the CNOS RETURN_CODE to OK--AS_SPECIFIED.
   When a program parameter check condition is identified as defined in
    <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>.
      Set CNOS RETURN_CODE to PROGRAM_PARAMETER_CHECK.

   When a parameter error is identified
      Set the CNOS RETURN_CODE for this verb to PARAMETER_ERROR.
   When the MODE.SESSION_LIMIT is not 0
      Set the CNOS RETURN_CODE to LU_MODE_SESSION_LIMIT_NOT_ZERO.
   When all (LU,MODE) session limits to this single session partner LU are currently 0
    and the sum of all (LU,MODE) session limits to other partner LUs = the total
    session limit (in the LUCB)
      Set the CNOS RETURN_CODE for this verb to LU_SESSION_LIMIT_EXCEEDED.
   When the session limit specified exceeds the LOCAL_MAX_SESSION_LIMIT in the MODE
      Set the CNOS RETURN_CODE for this verb to REQUEST_EXCEEDS_MAX_ALLOWED.

SVCMG_VERB_PARAMETER_CHECK

FUNCTION:	This procedure performs validity checks on a CNOS verb for mode name SNASVCMG, and it returns the CNOS-verb RETURN_CODE for any error detected.
INPUT:	Transaction program verb parameters, PARTNER_LU_LIST, and MODE_LIST
OUTPUT:	CNOS verb RETURN_CODE value if any errors are detected; otherwise, OK is returned

Referenced procedures, FSMs, and data structures:

LUCB	page A-1
MODE	page A-3

Verify that the verb parameters specified satisfy the parameter values appropriate for parallel-session connections, as described in SNA Transaction Programmer's Reference Manual for LU Type 6.2.

> Attributes of the mode are verified against fields in the appropriate MODE structure for the specified PARTNER_LU.

```
Select, in order, based on result of parameter verification:
   When an program parameter check condition is identified as defined in
    SNA Transaction Programmer's Reference Manual for LU Type 6.2.
      Set CNOS RETURN_CODE to PROGRAM_PARAMETER_CHECK.
   When a parameter error is identified
      Set the CNOS RETURN_CODE to PARAMETER_ERROR.
   When the MODE.SESSION_LIMIT is not 0
      Set the CNOS RETURN_CODE to LU_MODE_SESSION_LIMIT_NOT_ZERO.
   When the session limit specified could not be added without exceeding
    the session limit in the LUCB for the LU (page 5.4-4)
      Set the CNOS RETURN_CODE to LU_SESSION_LIMIT_EXCEEDED.
```

CHANGE_ACTION

FUNCTION:	This procedure is called when the LU accepts a valid (and negotiated, if necessary) CNOS command. This procedure updates the (LU,mode) entries for affected mode names with the new session limit parameters. It decides whether this LU is responsible for taking any action to change the session count, and, if so, sends a CHANGE_SESSIONS request to RM.
INPUT:	The CNOS verb parameters specified, if the CNOS verb is local to this LU only; the new session limit parameters in the CNOS reply record, if the CNOS action is distributed; the role of the LU to be modified (source or target), PARTNER_LU_LIST and MODE_LIST
OUTPUT:	Session limits and drain state are updated in the MODE; CHANGE_SESSIONS to RM
NOTE:	This procedure locks the MODE for the entire procedure. See SNA Transaction Programmer's Reference Manual for LU Type 6.2 for the session-limit parameters affected by each CNOS verb. This procedure has addressability to RM via PS_PROCESS_DATA.LU_ID.

Referenced procedures, FSMs, and data structures:

RM	page 3-19
CHANGE_SESSIONS	page A-15
PARTNER_LU	page A-2
MODE	page A-3

Select based on whether one MODE or all MODEs with the PARTNER_LU are affected (see the MODE_LIST associated with the PARTNER_LU):
  When only one MODE is affected
    Update the session-limit parameters for the specified (LU, mode) entry (MODE.SESSION_LIMIT, MODE.MIN_CONWINNERS_LIMIT, MODE.MIN_CONLOSERS_LIMIT, MODE.DRAIN_SELF, MODE.DRAIN_PARTNER, MODE.RESPONSIBLE) as they are applicable:

      For single-session mode names and for mode name SNASVCMG, the session limit parameters affected are those specified on the particular CNOS verb and the changes are reflected in the source LU only.

MODE.MINCONWINNERS_LIMIT is set from MINCONWINNERS_SOURCE specified in the CNOS command. MODE.MINCONLOSERS_LIMIT is set from MINCONWINNERS_TARGET specified in the CNOS command.

      For parallel-session connections defined with the partner LU, the session limit parameters affected are those specified on the CNOS reply and the changes are reflected as appropriate in both the source and the target LU (when this procedure is called from SOURCE_SESSION_LIMIT (or LOCAL_SESSION_LIMIT) and PROCESS_SESSION_LIMIT, respectively).

At the source LU, MODE.MIN_CONWINNERS_LIMIT is set from MIN_CONWINNERS_SOURCE specified in the CNOS reply and MODE.MIN_CONLOSERS_LIMIT is set from MIN_CONWINNERS_TARGET specified in the CNOS reply. The reverse is true at the target LU.

    If the verb issued at the source LU is INITIALIZE_SESSION_LIMIT or CHANGE_SESSION_LIMIT, or, according to the responsible field of the CNOS reply (applicable only when the CNOS function is distributed), this LU is responsible for session deactivation then

      Create a CHANGE_SESSIONS request record.
      Set CHANGE_SESSIONS.LU_NAME to PARTNER_LU.LOCAL_LU_NAME.
      Set CHANGE_SESSIONS.MODE_NAME to the affected mode name as specified on the CNOS verb.
      Set CHANGE_SESSIONS.DELTA to the difference between the LU_MODE_SESSION_LIMIT specified on the CNOS command or reply and the current MODE.SESSION_LIMIT.
      If the verb issued by the source LU is CHANGE_SESSION_LIMIT and the limit in the reply is less than the current session limit, or the verb issued by the source LU is the distributed function RESET_SESSION_LIMIT verb | MODE.DRAIN_SELF = NO then
        If the responsible field value in the CNOS reply specifies the current LU (which could be source or target) then
          Set CHANGE_SESSIONS.RESPONSIBLE to YES.
        Else
          Set CHANGE_SESSIONS.RESPONSIBLE to NO.
      Else (RESPONSIBLE value will not be significant to RM)
        Set CHANGE_SESSIONS.RESPONSIBLE to NO.
      Send the CHANGE_SESSIONS request to RM.

  When all MODEs are affected (in which case the verb issued by the source LU is RESET_SESSION_LIMIT)
    Do the following for each MODE (except SNASVCMG) with the PARTNER_LU
      Set MODE.DRAIN_SELF and MODE.DRAIN_PARTNER based on the current session limit and the drain parameters of the CNOS reply.
      Set SESSION_LIMIT, MIN_CONWINNERS_LIMIT and MIN_CONLOSERS_LIMIT to 0.
      If this LU is responsible for session deactivation | MODE.DRAIN_SELF = NO then
        Create a CHANGE_SESSIONS request record as described in detail above.
        Send the CHANGE_SESSIONS request to RM.

SOURCE_SESSION_LIMIT_PROC

FUNCTION:	This procedure is invoked by any of the following verb-specific CNOS procedures: INITIALIZE_SESSION_LIMIT, CHANGE_SESSION_LIMIT, RESET_SESSION_LIMIT. It provides common overall processing of a parallel-session CNOS control-operator verb issued by a source LU control operator transaction program. It invokes other procedures to check the verb parameters for validity, detect and resolve race conditions with any other CNOS transaction, build a command record, allocate a conversation with the target LU, exchange command and reply records with the target LU, update the PARTNER_LU_LIST and MODE_LIST with the new session limit parameters, and, if necessary, request the resources manager to activate or deactivate sessions. If errors are detected at any point, it skips subsequent steps and cleans up from previous steps. It passes a RETURN_CODE to the calling procedure indicating success or a failure reason.
INPUT:	CNOS source LU verb parameters, from the calling procedure; the CNOS reply from the target LU, via SOURCE_CONVERSATION_CONTROL; the (LU,mode) entries with the session limits in the MODE, PARTNER_LU_LIST and MODE_LIST, and other CNOS parameters; the lock to control contention for the PARTNER_LU_LIST and MODE_LIST by CNOS transaction processes, and to resolve CNOS races (maintained by SESSION_LIMIT_DATA_LOCK_MANAGER)
OUTPUT:	Return code for the CNOS verb, CNOS RETURN_CODE; MODE.CNOS_NEGOTIATION_IN_PROGRESS and MODE.LIMIT_BEING_NEGOTIATED; procedure SOURCE_CONVERSATION allocates and deallocates a conversation with the target LU and issues conversation verbs; specified (LU,mode) entries updated via CHANGE_ACTION in the MODE; CHANGE_SESSIONS issued to RM—via CHANGE_ACTION

Referenced procedures, FSMs, and data structures:

SESSION_LIMIT_DATA_LOCK_MANAGER	page 5.4-66
VERB_PARAMETER_CHECK	page 5.4-47
SOURCE_CONVERSATION_CONTROL	page 5.4-48
CHECK_CNOS_REPLY	page 5.4-55
CHANGE_ACTION	page 5.4-43
PARTNER_LU	page A-2
MODE	page A-3

```
Call VERB_PARAMETER_CHECK (page 5.4-47),
 with the verb parameters, to verify the syntax of the parameters.

If all parameters are determined to be correct then

   Call SESSION_LIMIT_DATA_LOCK_MANAGER (page 5.4-66)
    to perform a source-LU lock on the affected (LU,mode) entry or entries
    and prevent simultaneous access by other CNOS transactions.

   Select based on one of the following conditions:
      When the state of the lock is changed from UNLOCKED to
       LOCKED_BY_SOURCE for each affected (LU,mode) entry

                  | MODE is now locked against any other CNOS transaction. |

         Build a CNOS command record with the parameters specified on the verb
          and consistent with the change-number-of-sessions record
          (SNA Formats).

         If the command is change or initialize session limits then
            If the MODE.SESSION_LIMIT < the new limit that is being proposed then
               Set MODE.CNOS_NEGOTIATION_IN_PROGRESS = TRUE.
               Set MODE.LIMIT_BEING_NEGOTIATED = LU_MODE_SESSION_LIMIT from verb.
               This is done so BINDs that arrive prior to the CNOS reply are not rejected.

         Do until the CHECK_CNOS_REPLY procedure does not return RETRY

                | The verb completes or a permanent error occurs. |

            Call SOURCE_CONVERSATION_CONTROL (page 5.4-48),
             with the CNOS command, to send on the conversation and to receive the CNOS reply.

            If the SOURCE_CONVERSATION_CONTROL returns OK (a CNOS reply was
             successfully received) then
               Optionally, perform syntax checking on the CNOS reply record
                according to the description in SNA Formats.

               If the CNOS reply is syntactically correct, or the syntax
                check was not performed then
                  Call CHECK_CNOS_REPLY with the CNOS reply record and the
                   network-qualified LU names for the source and target LUs
                   to determine the result of the negotiation (page 5.4-55).

               Else
                  Set the CNOS RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.

         If the session limits were successfully accepted or negotiated then
            Call CHANGE_ACTION (page 5.4-43), with the CNOS reply,
             to update the limits in the MODE structure for the source LU and notify RM.

         If the command is change or initialize session limits then
            Set MODE.CNOS_NEGOTIATION_IN_PROGRESS = FALSE.
         Call SESSION_LIMIT_DATA_LOCK_MANAGER (page 5.4-66)
          to perform the unlock operation on the affected (LU,mode) entry or entries.

      When the lock operation performed on any of the affected (LU,mode) entries
       was other than a state change from UNLOCKED to LOCKED_BY_SOURCE
       (because of a previous lock operation performed for a different CNOS command)
         Set the CNOS RETURN_CODE to COMMAND_RACE_REJECT.

      When the mode name is not found for the PARTNER_LU
         Set the CNOS RETURN_CODE to PARAMETER_ERROR.
```

VERB_PARAMETER_CHECK

FUNCTION:	This procedure performs validity checks on the CNOS verb issued by the control-operator transaction program at the source LU, and it returns the CNOS-verb RETURN_CODE for any error detected.
INPUT:	Parameters from transaction program verb, PARTNER_LU_LIST and MODE_LIST
OUTPUT:	CNOS verb RETURN_CODE value if any errors are detected; otherwise, OK is returned
NOTE:	This procedure locks the MODE for the entire procedure.

Referenced procedures, FSMs, and data structures:

LUCB	page A-1
MODE	page A-3

```
Verify that the specified verb parameters satisfy the parameter values as
 described for this verb in SNA Transaction Programmer's Reference Manual for LU Type 6.2.

     Attributes of the mode name are  verified against fields in the appro-
     priate MODE structure for the specified PARTNER_LU.

Select based on result of parameter verification:
   When all parameters are correct
      Set the CNOS RETURN_CODE for this verb to OK--AS_SPECIFIED.
   When a program parameter check condition is identified as described in
    SNA Transaction Programmer's Reference Manual for LU Type 6.2.
      Set CNOS RETURN_CODE to PROGRAM_PARAMETER_CHECK.
   When a parameter error is identified
      Set the CNOS RETURN_CODE to PARAMETER_ERROR.
   When the verb issued is INITIALIZE_SESSION_LIMIT and the MODE.SESSION_LIMIT
    is not 0 for the affected MODE at the PARTNER_LU
      Set the CNOS RETURN_CODE to LU_MODE_SESSION_LIMIT_NOT_ZERO.
   When the verb issued is CHANGE_SESSION_LIMIT and the MODE.SESSION_LIMIT
    is 0 for the affected MODE at the PARTNER_LU
      Set the CNOS RETURN_CODE to LU_MODE_SESSION_LIMIT_ZERO.
   When the specified session limit could not be added without exceeding
    the session limit in the LUCB for the LU (page 5.4-4).
      Set the CNOS RETURN_CODE to LU_SESSION_LIMIT_EXCEEDED.
   When the specified session limit could not be added without exceeding
    the LOCAL_MAX_SESSION_LIMIT in the MODE
      Set the CNOS RETURN_CODE to REQUEST_EXCEEDS_MAX_ALLOWED.
```

SOURCE_CONVERSATION_CONTROL

FUNCTION:	This procedure controls a conversation with the target LU to send the CNOS command and receive the CNOS reply. It controls the selection of mode name for the conversation. In the event of session outage, it retries the conversation either until it succeeds or until no sessions are active for any mode name affected by the CNOS verb.
INPUT:	CNOS verb parameters including the name of the target LU; CNOS command; summary of the success or failure of the CNOS exchange across the conversation (provided by the SOURCE_CONVERSATION procedure) so this routine can make a retry decision: • OK: conversation completed successfully • SON: session outage occurred; retry for the same mode name might succeed • NO_SESSION: no session is available for this mode name; retry for another mode name might succeed • FAILED: conversation or transaction failure; retry is not likely to succeed
OUTPUT:	CNOS reply; summary of outcome of conversation for caller

Referenced procedures, FSMs, and data structures:

SOURCE_CONVERSATION page 5.4-49
LUCB page A-1
PARTNER_LU page A-2
MODE page A-3

Do until the SOURCE_CONVERSATION procedure returns a value OK or FAILED, or if all possible modes are tried but no sessions are available on any of these

Choose a mode name with which to allocate a conversation. The mode name is optionally selected from an implementation-defined list (if any of these sessions is immediately available) or the SNA-defined mode name SNASVCMG.
Choose the RETURN_CONTROL value for the ALLOCATE verb (see SNA Transaction Programmer's Reference Manual for LU Type 6.2).

> Initially, choose mode names from the implementation defined list and use a RETURN_CONTROL value of IMMEDIATE. Once these have been exhausted, try the SNA-defined mode (SNASVCMG) with a RETURN_CONTROL value of WHEN_SESSION_ALLOCATED. If this is not successful, choose a mode name from those that will be affected by this CNOS command and use a RETURN_CONTROL value of WHEN_SESSION_ALLOCATED.

Call SOURCE_CONVERSATION (page 5.4-49) with the parameters chosen above and the CNOS command record. SOURCE_CONVERSATION will issue the basic conversation verbs to send the CNOS command, receive the CNOS reply over the conversation and obtain the network-qualified LU names for this and the partner LU for later comparison.

> If SON (session outage notification) is returned, the conversation is retried on another session for the same mode name.

Set the return value for this routine to the value returned from SOURCE_CONVERSATION.

SOURCE_CONVERSATION

FUNCTION:	This procedure conducts a conversation with the target LU to send the CNOS command and receive the CNOS reply. It issues the conversation verbs. It invokes other routines to analyze the return codes to determine when and how to deallocate the conversation and whether retry is necessary.
INPUT:	LU name of the partner, mode name for the conversation on which the CHANGE_NUMBER_OF_SESSIONS command and reply records are exchanged; the RETURN_CONTROL parameter for the ALLOCATE verb; CNOS command
OUTPUT:	CNOS reply; summary of the success or failure of a particular basic conversation verb, according to the particular RESULT_CHECK_* procedure called: • OK: conversation completed successfully • SON: session outage occurred; retry for the same mode name might succeed • NO_SESSION: no session is available for this mode name; retry for another mode name might succeed • FAILED: conversation or transaction failure; retry is not likely to succeed The SOURCE_CONVERSATION_CONTROL procedure will make a retry decision based on this information.
NOTE:	See SNA Transaction Programmer's Reference Manual for LU Type 6.2 for conversation verbs.

Referenced procedures, FSMs, and data structures:

RESULT_CHECK_ALLOCATE page 5.4-51
RESULT_CHECK_SEND_COMMAND page 5.4-52
RESULT_CHECK_RECEIVE_REPLY page 5.4-53
RESULT_CHECK_RECEIVE_DEALLOCATE page 5.4-54

Conduct a conversation with the partner.

Issue the ALLOCATE verb according to the mode name and RETURN_CONTROL values passed to this procedure and default values as described on page 5.4-27.
Call RESULT_CHECK_ALLOCATE to examine the RETURN_CODE value from the ALLOCATE (according to the RETURN_CONTROL value specified on the verb) and issue DEALLOCATE for the conversation if appropriate (page 5.4-51).

If the ALLOCATE verb returned OK then
   Issue a GET_ATTRIBUTES verb, with the RESOURCE parameter returned
    from the ALLOCATE, to obtain the network-qualified LU name of
    the partner LU.
   Issue a GET_TP_PROPERTIES verb to get the network-qualified LU name of
    the local LU.

These LU names are required for comparison in the CHECK_CNOS_REPLY to determine the winner for a double-failure race.

   Issue a SEND_DATA verb to send the CNOS command.
   Call RESULT_CHECK_SEND_COMMAND (page 5.4-52) to examine
    the parameters returned from the SEND_DATA verb and perform the DEALLOCATE
    if appropriate.

   If the SEND_DATA verb returned OK then
      Issue a RECEIVE_AND_WAIT verb to receive the CNOS reply.
      Call RESULT_CHECK_RECEIVE_REPLY (page 5.4-53) to examine
       the parameters returned from the RECEIVE_AND_WAIT verb and perform the DEALLOCATE
       if appropriate.

      If the RECEIVE_AND_WAIT verb returned OK then
         Issue the RECEIVE_AND_WAIT verb to receive the DEALLOCATE from the
          partner LU.
         Call RESULT_CHECK_RECEIVE_DEALLOCATE (page 5.4-54) to examine
          the parameters returned from the RECEIVE_AND_WAIT verb and perform the DEALLOCATE
          if appropriate.

Set the return code for this procedure from the value returned by the last
 RESULT_CHECK_* procedure called.

RESULT_CHECK_ALLOCATE

FUNCTION:	This procedure analyzes the RETURN_CODE and other significant returned parameters from the ALLOCATE verb that allocates the CNOS conversation, and it classifies the outcome for use in later decisions, specifically whether to retry, quit, or continue. For some error conditions, the conversation will need to be deallocated.
INPUT:	RETURN_CODE, RETURN_CONTROL
OUTPUT:	Summary of the success or failure of the ALLOCATE verb: • OK: conversation completed successfully • SON: session outage occurred; retry for the same mode name might succeed • NO_SESSION: no session is available for this mode name; retry for another mode name might succeed • FAILED: conversation or transaction failure; retry is not likely to succeed This information will be used by SOURCE_CONVERSATION_CONTROL to make a retry decision.
NOTE:	Checks are required unless designated optional.

```
Select based on the RETURN_CONTROL value specified on the ALLOCATE verb:
   When IMMEDIATE (implementation-selected mode name)

      Select based on the RETURN_CODE value from the ALLOCATE verb:
         When OK
            Return OK to the SOURCE_CONVERSATION procedure.
         When ALLOCATION_ERROR (optional check)
            Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate
             the conversation locally.
            Return FAILED to the SOURCE_CONVERSATION procedure.
         When UNSUCCESSFUL (no session is immediately available)
            Return NO_SESSION to the SOURCE_CONVERSATION procedure.
         Otherwise (optional check)
            Return FAILED to the SOURCE_CONVERSATION procedure.

   When WHEN_SESSION_ALLOCATED

      Select based on the RETURN_CODE value from the ALLOCATE verb:
         When OK
            Return OK to the SOURCE_CONVERSATION procedure.
         When ALLOCATION_ERROR--ALLOCATION_FAILURE_RETRY
            Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate the
             conversation locally.
            Return NO_SESSION to the SOURCE_CONVERSATION procedure.
         Otherwise (optional check)
            Return FAILED to the SOURCE_CONVERSATION procedure.
```

RESULT_CHECK_SEND_COMMAND

FUNCTION:	This procedure analyzes the RETURN_CODE and other significant returned parameters from the SEND_DATA verb that sends the CNOS command, and it classifies the outcome for use in later decisions, specifically whether to retry, quit, or continue. For some error conditions, the conversation may need to be deallocated.
INPUT:	RETURN_CODE, REQUEST_TO_SEND_RECEIVED
OUTPUT:	Summary of the success or failure of the SEND_DATA verb: • OK: conversation completed successfully • SON: session outage occurred; retry for the same mode name might succeed • NO_SESSION: no session is available for this mode name; retry for another mode name might succeed • FAILED: conversation or transaction failure; retry is not likely to succeed This information will later be used by SOURCE_CONVERSATION_CONTROL to make a retry decision.
NOTE:	Checks are required unless designated optional.

```
Select, in order, based on the RETURN_CODE parameter from the SEND_DATA verb:
   When OK
      If the REQUEST_TO_SEND_RECEIVED parameter returned from the SEND_DATA verb is YES then
        Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the conversation.
        Return FAILED to the SOURCE_CONVERSATION procedure.
      Else
         Return OK to the SOURCE_CONVERSATION procedure.

   When RESOURCE_FAILURE_RETRY
      Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate the conversation
       locally.
      Return SON (session outage notification) to the SOURCE_CONVERSATION procedure.

   When ALLOCATION_ERROR--SECURITY_NOT_VALID,
        ALLOCATION_ERROR--TP_NOT_AVAILABLE_NO_RETRY,
        or ALLOCATION_ERROR--TP_NOT_AVAILABLE_RETRY
      Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate the
       conversation locally.
      Return FAILED to the SOURCE_CONVERSATION procedure.
   When ALLOCATION_ERROR--* (optionally check for any other variety of ALLOCATION_ERROR)
      Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate the
       conversation locally.
      Return FAILED to the SOURCE_CONVERSATION procedure.

   When  DEALLOCATE_ABEND_PROG
      Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate the conversation
       locally.
      Return FAILED to the SOURCE_CONVERSATION procedure.
   Otherwise
      Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the
       conversation.
      Return FAILED to the SOURCE_CONVERSATION procedure.
```

RESULT_CHECK_RECEIVE_REPLY

FUNCTION:	This procedure analyzes the RETURN_CODE and other significant returned parameters from the RECEIVE_AND_WAIT verb that receives the CNOS reply, and it classifies the outcome for use in later decisions, specifically whether to retry, quit, or continue. For some error conditions, the conversation may need to be deallocated.
INPUT:	RETURN_CODE, REQUEST_TO_SENT_RECEIVED, WHAT_RECEIVED
OUTPUT:	Summary of the success or failure of the RECEIVE_AND_WAIT verb: • OK: conversation completed successfully • SON: session outage occurred; retry for the same mode name might succeed • NO_SESSION: no session is available for this mode name; retry for another mode name might succeed • FAILED: conversation or transaction failure; retry is not likely to succeed This information will later be used by SOURCE_CONVERSATION_CONTROL to make a retry decision.
NOTE:	Checks are required unless designated optional.

```
Select based on the RETURN_CODE value returned from the RECEIVE_AND_WAIT verb:
   When OK
      If the WHAT_RECEIVED parameter returned is DATA_COMPLETE then
         If the REQUEST_TO_SEND_RECEIVED parameter from the RECEIVE_AND_WAIT verb is NO then
            Return OK to the SOURCE_CONVERSATION procedure.
         Else
            Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the conversation.
            Return FAILED to the SOURCE_CONVERSATION procedure.
      Else
         Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the
          conversation.
         Return FAILED to the SOURCE_CONVERSATION procedure.
   When RESOURCE_FAILURE_RETRY
      Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate the
       conversation locally.
      Return SON (session outage notification) to the SOURCE_CONVERSATION procedure.
   When ALLOCATION_ERROR--SECURITY_NOT_VALID, ALLOCATION_ERROR--TP_NOT_AVAILABLE_NO_RETRY,
    or ALLOCATION_ERROR--TP_NOT_AVAILABLE_RETRY
       Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate the conversation locally.
       Return FAILED to the SOURCE_CONVERSATION procedure.
   When ALLOCATION_ERROR--*
      (optionally check for any other variety of ALLOCATION_ERROR)
      Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate the conversation locally.
      Return FAILED to the SOURCE_CONVERSATION procedure.
   When DEALLOCATE_NORMAL or DEALLOCATE_ABEND_PROG (optional check)
      Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate the conversation locally.
      Return FAILED to the SOURCE_CONVERSATION procedure.
   Otherwise
      Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the conversation.
            Return FAILED to the SOURCE_CONVERSATION procedure.
```

RESULT_CHECK_RECEIVE_DEALLOCATE

FUNCTION:	This procedure analyzes the RETURN_CODE and other significant returned parameters from the RECEIVE_AND_WAIT verb that receives DEALLOCATE from the target LU, and it classifies the outcome for use in later decisions, specifically whether to retry, quit, or continue. For some error conditions, the conversation may need to be deallocated.
INPUT:	RETURN_CODE, REQUEST_TO_SEND_RECEIVED, WHAT_RECEIVED (used only for error log)
OUTPUT:	Summary of the success or failure of the RECEIVE_AND_WAIT verb: • OK: conversation completed successfully • SON: session outage occurred; retry on the same mode name might succeed • NO_SESSION: no session is available for this mode name; retry on another mode name might succeed • FAILED: conversation or transaction failure; retry is not likely to succeed

```
Select based on the RETURN_CODE value returned from the DEALLOCATE verb:
   When DEALLOCATE_NORMAL
      If the REQUEST_TO_SEND_RECEIVED parameter from RECEIVE_AND_WAIT verb is YES then
         Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the conversation.
         Return FAILED to the SOURCE_CONVERSATION procedure.
      Else
         Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate the conversation locally.
         Return OK to the SOURCE_CONVERSATION procedure.

   When RESOURCE_FAILURE_RETRY
      Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate the conversation locally.
      Return SON (session outage notification) to the SOURCE_CONVERSATION procedure.

   When DEALLOCATE_ABEND_PROG
      Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate the conversation locally.
      Return FAILED to the SOURCE_CONVERSATION procedure.

   Otherwise
      Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the conversation.
      Return FAILED to the SOURCE_CONVERSATION procedure.
```

CHECK_CNOS_REPLY

FUNCTION:	This procedure is called when the conversation with the target LU completes. It determines whether the conversation must be retried due to a double-failure race condition, whether the verb must be terminated due to error, or whether to continue with the action phase of CNOS processing. It performs optional receive checks on the validity of the reply. It sets the return code for the CNOS verb.
INPUT:	Fields of the CNOS reply record, PARTNER_LU_LIST and MODE_LIST for current session limit OUTPUT.CNOS RETURN_CODE, if any errors are found; RETRY, used by caller to select subsequent processing
NOTE:	Checks are required unless designated optional.

Referenced procedures, FSMs, and data structures:

LUCB	page A-1
PARTNER_LU	page A-2
MODE	page A-3

```
Select based on the reply modifier field of the CNOS reply record:
   When the reply modifier is MODE_NAME_NOT_RECOGNIZED
      Set the CNOS RETURN_CODE to UNRECOGNIZED_MODE_NAME.
   When the reply modifier indicates an (LU,mode) session limit of 0
      Verify that, for the PARTNER_LU MODEs specified on the original CNOS verb,
       that the SESSION_LIMIT=0, and DRAIN_SELF=NO.

      If these MODE attributes are correctly verified then
         Set the CNOS RETURN_CODE to LU_MODE_SESSION_LIMIT_CLOSED.

      Else
         Set the CNOS RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
   When the reply modifier is COMMAND_RACE_DETECTED
      Check the state of the lock to determine whether the race is a single- or
       double-failure race (page 5.4-30).
      Compare the network-qualified LU names for the source and target LUs
       (returned from the GET_ATTRIBUTES and GET_TP_PROPERTIES verbs in the
       SOURCE_CONVERSATION procedure) with respect to the EBCDIC collating sequence
       (page 5.4-14).

      If the race detected is a single-failure race or the LU name of the target
       LU is greater by the above comparison then
         Set the CNOS RETURN_CODE to COMMAND_RACE_REJECT.

      Else (double-failure race condition and source LU name is greater)
         Return RETRY to SOURCE_SESSION_LIMIT_PROC.
   When the reply modifier is ACCEPTED
      Set the CNOS RETURN_CODE to OK--AS_SPECIFIED.
   When the reply modifier is NEGOTIATED
      Optionally verify that the parameters in the CNOS reply were correctly
       negotiated, according to page 5.4-28.

      If the reply parameters were successfully verified or the optional
       checks were not implemented then
         Set the CNOS RETURN_CODE to OK--AS_NEGOTIATED.

      Else
         Set the CNOS RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
```

X06F1

FUNCTION:	This procedure is the CNOS service transaction program at the target LU. It is invoked by PS_INITIALIZE as a result of an FMH-5 Attach header being received from the source LU. It issues the PROCESS_SESSION_LIMIT control operator verb to activate CNOS processing at the target LU. It informs the target-LU operator of the CNOS action.
OUTPUT:	Issues control-operator verb PROCESS_SESSION_LIMIT
NOTE:	See SNA Transaction Programmer's Reference Manual for LU Type 6.2 for control-operator verbs.

Issue the PROCESS_SESSION_LIMIT verb to be processed by PS_COPR (page 5.4-32) and inform the target-LU operator of the resulting CNOS RETURN_CODE.

> The algorithm to inform the operator is implementation dependent. This algorithm may make use of DEFINE or DISPLAY control-operator verbs to determine the current session limits, in the MODE, and then display them on the operator console.

PROCESS_SESSION_LIMIT_PROC

FUNCTION:	This procedure is invoked by PS_COPR, the control-operator-verb router, when the CNOS service transaction program at the target LU issues a PROCESS_SESSION_LIMIT control-operator verb. This procedure directs overall processing of CHANGE_NUMBER_OF_SESSIONS at the target LU. This procedure receives the CNOS command from the source LU and sends the CNOS reply. It invokes TARGET_CONVERSATION to issue the conversation verbs and process the return codes. It invokes other procedures to check the verb and the conversation attributes for validity, detect and resolve race conditions with any other CNOS transaction, negotiate CNOS parameters, update the affected MODEs with the new session limit parameters, and, if necessary, request the resources manager to activate or deactivate sessions. If errors are detected at any point, it skips subsequent steps and cleans up from previous steps. It passes a RETURN_CODE to the calling procedure in the PROCESS_SESSION_LIMIT record indicating success or a failure reason.
INPUT:	PROCESS_SESSION_LIMIT verb, CNOS command from the source LU via the conversation; PARTNER_LU_LIST and MODE_LIST
OUTPUT:	Outcome of the operation to the caller in PROCESS_SESSION_LIMIT (RETURN_CODE); CNOS reply sent to the source LU via the conversation; updated MODE entries via CHANGE_ACTION; CHANGE_SESSIONS record to RM, via CHANGE_ACTION; SESSION_LIMIT_DATA lock tested, set, and reset via SESSION_LIMIT_DATA_LOCK_MANAGER

Referenced procedures, FSMs, and data structures:

NEGOTIATE_REPLY	page 5.4-63
CHECK_CNOS_COMMAND	page 5.4-62
CHANGE_ACTION	page 5.4-43
TARGET_COMMAND_CONVERSATION	page 5.4-60
TARGET_REPLY_CONVERSATION	page 5.4-64
SESSION_LIMIT_DATA_LOCK_MANAGER	page 5.4-66
LUCB	page A-1
PARTNER_LU	page A-2
MODE	page A-3

```
Check the verb parameters to detect ABEND conditions as described in
 SNA Transaction Programmer's Reference Manual for LU Type 6.2 for this verb.

If either of the ABEND conditions exists then
   Set the CNOS RETURN_CODE to PROGRAM_PARAMETER_CHECK.

Else
   Call TARGET_COMMAND_CONVERSATION (page 5.4-60)
    with the resource ID of the conversation with the partner LU to receive
    the CNOS command from the source LU.
   If an error occurs before the CNOS command can be successfully received then
      Set the CNOS RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.

   Else
      Call SESSION_LIMIT_DATA_LOCK_MANAGER to perform a target-LU lock
       on the appropriate (LU,mode) entry or entries to prevent
       simultaneous access by other CNOS transactions (page 5.4-66).
      Optionally, perform syntax checking on the CNOS command record according
       to the description in SNA Formats.

      Select, in order, based on the values of fields in the CNOS command:
         When syntax errors are identified
            Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the
             conversation.
         When the MODEs specified on the CNOS command cannot be found
          in the list of MODEs for the PARTNER_LU
            Set the reply modifier field for the CNOS reply to MODE_NAME_NOT_RECOGNIZED.
         When the MODEs specified on the CNOS command have SESSION_LIMIT=0, and
          DRAIN_SELF=NO then
            The LU may refuse to accept the command by returning an abnormal reply
             modifier field specifying an (LU,mode) session limit of 0
             (this is implementation defined).
         Otherwise

            Select based on result of SESSION_LIMIT_DATA_LOCK_MANAGER:
               When the state of the LOCKs have changed from UNLOCKED
                to LOCKED_BY_TARGET
                  Call CHECK_CNOS_COMMAND (page 5.4-62), with the CNOS command,
                   to perform optional receive checks (if errors are found,
                   the conversation is deallocated).
                  If the checks were not performed or no errors were detected then
                     Call NEGOTIATE_REPLY (page 5.4-63), with the CNOS command
                      record, in order to generate the negotiated values of the
                      CNOS parameters.
               Otherwise (if any LOCK has been rejected)
                  Set the reply modifier field for the CNOS reply to COMMAND_RACE_DETECTED.

If the conversation has not been deallocated then
   Build the CNOS reply record consistent with the original CNOS command, the reply modifier
    field reflecting the identified errors, and the negotiated CNOS limits, as
    appropriate (see SNA Formats).

   If the command is change or initialize session limits then
      If the modifier field of the CNOS reply is accepted or negotiated then
         If the new limit > MODE.SESSION_LIMIT then
            Set MODE.CNOS_NEGOTIATION_IN_PROGRESS to TRUE.
            Set MODE.LIMIT_BEING_NEGOTIATED to LU_MODE_SESSION_LIMIT.

   Call TARGET_REPLY_CONVERSATION (page 5.4-64)
    with the CNOS reply record to be sent to the source LU.

   If the CNOS reply is successfully sent across the conversation then
      Set the CNOS RETURN_CODE for the PROCESS_SESSION_LIMIT verb according
       to the modifier field of the CNOS reply.
      If the reply modifier field indicates that the CNOS limits were either ACCEPTED
       or NEGOTIATED then
          Call CHANGE_ACTION (page 5.4-43) with the CNOS reply record
           to change the session limits at the target LU.

   Else
      Set the CNOS RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
```

PROCESS_SESSION_LIMIT_PROC

If the command is change or initialize session limits then
   Set MODE.CNOS_NEGOTIATION_IN_PROGRESS to FALSE.

Call SESSION_LIMIT_DATA_LOCK MANAGER (page 5.4-66) to UNLOCK the affected (LU,mode) entry or entries.

TARGET_COMMAND_CONVERSATION

FUNCTION:	This procedure checks the attaching conversation for validity and returns the partner LU name to the caller. If the conversation is valid, this procedure receives the CNOS command from the source LU. If an error is detected, it terminates the conversation with DEALLOCATE TYPE(ABEND_PROG).
INPUT:	Resource ID of the conversation with the partner (source) LU, conversation attributes via GET_ATTRIBUTES
OUTPUT:	Partner LU name, from conversation via GET_ATTRIBUTES; CNOS command, from the source LU via the conversation;
NOTE:	See SNA Transaction Programmer's Reference Manual for LU Type 6.2 for conversation verbs.

Referenced procedures, FSMs, and data structures:

RESULT_CHECK_RECEIVE_COMMAND page 5.4-61
RESULT_CHECK_RECEIVE_SEND page 5.4-61

Issue a GET_TYPE verb (according to the input parameters provided) to verify that the type of conversation is BASIC.
Issue a GET_ATTRIBUTES verb (according to the input parameters provided) to verify that the connection type is parallel sessions and that the SYNC_LEVEL is NONE (optional receive check).

> The GET_ATTRIBUTES verb returns the name of the source LU. The target then uses this information to determine the type of sessions possible with the source LU as a conversation partner.

If the above conversation attributes are not verified to be correct then (optional check)
  Issue a DEALLOCATE verb with TYPE=ABEND_PROG and return from this procedure. The LOG_DATA parameter of the DEALLOCATE verb, if used, is supplied by the implementation. For its format, see ERROR LOG GDS VARIABLE in SNA Formats.

Else

  Issue a RECEIVE_AND_WAIT verb to receive the CNOS command.
  Call RESULT_CHECK_RECEIVE_COMMAND to examine the parameters returned and perform the DEALLOCATE, if appropriate (page 5.4-61).

  If RESULT_CHECK_RECEIVE_COMMAND returns OK then
    Issue a RECEIVE_AND_WAIT verb to receive the SEND indicator.
    Call RESULT_CHECK_RECEIVE_SEND to examine the parameters returned and perform the DEALLOCATE, if appropriate (page 5.4-61). If RESULT_CHECK_RECEIVE_SEND returns OK, the CNOS command was successfully received.

RESULT_CHECK_RECEIVE_COMMAND

FUNCTION:	This procedure analyzes the RETURN_CODE and other significant returned parameters from the RECEIVE_AND_WAIT verb that receives the CNOS command; it determines whether to issue DEALLOCATE, and what TYPE to specify.
INPUT:	RETURN_CODE, REQUEST_TO_SEND_RECEIVED, WHAT_RECEIVED
NOTE:	Checks are required unless designated optional.

```
Select based on the RETURN_CODE parameter returned from RECEIVE_AND_WAIT:
   When OK
      If WHAT_RECEIVED = DATA_COMPLETE then
         If the REQUEST_TO_SEND_RECEIVED parameter from the RECEIVE_AND_WAIT
          verb is YES then
            Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the conversation.
         Else
            Return OK to TARGET_COMMAND_CONVERSATION.
      Else (optional check)
         Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the conversation.
   When RESOURCE_FAILURE_RETRY, DEALLOCATE_NORMAL or DEALLOCATE_ABEND_PROG (optional check)
      Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate the conversation locally.
   When RESOURCE_FAILURE_NO_RETRY
      Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the conversation.
   Otherwise (optional check)
      Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the conversation.
```

RESULT_CHECK_RECEIVE_SEND

FUNCTION:	This procedure analyzes the RETURN_CODE and other significant returned parameters from the RECEIVE_AND_WAIT verb that receives SEND; it determines whether to issue DEALLOCATE, and what TYPE to specify.
INPUT:	RETURN_CODE, REQUEST_TO_SEND_RECEIVED, WHAT_RECEIVED
NOTE:	Checks are required unless designated optional.

```
Select based on the RETURN_CODE parameter returned from the RECEIVE_AND_WAIT:
   When OK
      If WHAT_RECEIVED = SEND & REQUEST_TO_SEND_RECEIVED = NO then
         Return OK to TARGET_COMMAND_CONVERSATION.
      Else
         Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the conversation.
   When RESOURCE_FAILURE_RETRY, DEALLOCATE_NORMAL, or
    DEALLOCATE_ABEND_PROG (optional check)
      Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate the conversation locally.
   Otherwise
      Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the conversation.
```

CHECK_CNOS_COMMAND

FUNCTION:	This procedure performs receive checks at the target LU on the CNOS command received from the source LU. If errors are detected, DEALLOCATE ABEND replaces a CNOS reply.
INPUT:	CNOS command parameters
NOTE:	Checks are required unless designated optional.

Referenced procedures, FSMs, and data structures:
MODE page A-3

Optionally check the verb parameters, encoded as fields in the CNOS command, for ABEND conditions as described in SNA Transaction Programmer's Reference Manual for LU Type 6.2.

> Since the session limits of the SNA-defined mode name, SNASVCMG, may not be changed, a mode name of SNASVCMG in the CNOS command constitutes another ABEND condition.
>
> Some parameter checks may require knowledge of mode attributes that currently exist. For these, see the appropriate MODE structure for the specified PARTNER_LU.

```
If any ABEND condition is identified then
   Issue a DEALLOCATE verb with TYPE=ABEND_PROG
    to deallocate the conversation.
```

NEGOTIATE_REPLY

FUNCTION:	This procedure generates the negotiated values of the CNOS limits for the CNOS reply, including the reply modifier field. This procedure assumes that the session limit parameters in the command are valid.
INPUT:	Source-LU specified CNOS verb parameters, PARTNER_LU_LIST, and MODE_LIST
OUTPUT:	Session limit parameters for reply
NOTE:	This procedure does not change the CNOS limits in the MODE.

Referenced procedures, FSMs, and data structures:

CLOSE_ONE_REPLY	page 5.4-64
PARTNER_LU	page A-2
MODE	page A-3

```
If the CNOS verb issued at the source LU is INITIALIZE_SESSION_LIMIT
 or CHANGE_SESSION_LIMIT (when the action field of the CNOS command is SET) then
   Negotiate the LU_MODE_SESSION_LIMIT, MIN_CONWINNERS_SOURCE,
    and MIN_CONWINNERS_TARGET parameters (as described in
    SNA Transaction Programmer's Reference Manual for LU Type 6.2) according to
    an implementation-dependent algorithm.
    If any of the session limits are going to be less than the current
    limits, RESPONSIBLE may also be negotiated from TARGET to SOURCE.

Else (RESET_SESSION_LIMIT verb)
   If the command affects only one MODE at the PARTNER_LU then
      Call CLOSE_ONE_REPLY (page 5.4-64)
       with the CNOS command record to build the CNOS reply record.
   Else (all mode names affected, and only RESPONSIBLE may be negotiated)
      If RESPONSIBLE is target then
         If RESPONSIBLE parameter is negotiated for this LU then
            Negotiate the RESPONSIBLE parameter from TARGET to SOURCE.
            Set the reply modifier field of the CNOS reply to NEGOTIATED.
      Else
         Set the reply modifier field of the CNOS reply to ACCEPTED.
```

CLOSE_ONE_REPLY

FUNCTION:	This procedure builds the target-LU's reply whenever the verb issued at the source LU is RESET_SESSION_LIMIT (action field of the CNOS command is CLOSE) and only one mode name is affected. It optionally sets the reply-modifier field of the CNOS reply to MODE_NAME_CLOSED if there is an error in DRAIN_SOURCE.
INPUT:	LU_NAME of partner LU; MODE, for current state of CNOS parameters; CNOS command parameters
OUTPUT:	Updated reply modifier and negotiated parameters

Referenced procedures, FSMs, and data structures:

MODE page A-3

Create the CNOS reply according to the negotiation rules described on page 5.4-28 (when the action field in the CNOS command is CLOSE and only one mode name is affected) and the description of the DRAIN and RESPONSIBLE parameters of the RESET_SESSION_LIMITS verb in _SNA Transaction Programmer's Reference Manual for LU Type 6.2_.

> If the current session limit is 0, the drain for the source LU (MODE.DRAIN_PARTNER) is set to NO and the command specifies DRAIN_SOURCE(YES), the target LU may either issue a DEALLOCATE with TYPE=ABEND or send a CNOS reply with the MODIFIER field specifying an (LU,mode) session limit of 0.
>
> This condition occurs only when there is a design error in the source LU such that this ABEND condition is not recognized and the command is forwarded to the target LU.

TARGET_REPLY_CONVERSATION

FUNCTION:	This procedure sends the CNOS reply.
INPUT:	Resource ID of the conversation with the partner (source) LU; the change-number-of-sessions record, in this case, a CNOS reply
OUTPUT:	Outcome of conversation (reply and DEALLOCATE NORMAL sent; DEALLOCATE ABEND sent or DEALLOCATE received)
NOTE:	See _SNA Transaction Programmer's Reference Manual for LU Type 6.2_ for conversation verbs.

Referenced procedures, FSMs, and data structures:

RESULT_CHECK_SEND_REPLY page 5.4-65

Issue a SEND_DATA verb (with the resource ID of the attaching conversation) to send the CNOS reply to the source LU.

Call RESULT_CHECK_SEND_REPLY (page 5.4-65) to examine the parameters returned on the verb and perform a DEALLOCATE of the conversation, if appropriate.

RESULT_CHECK_SEND_REPLY

FUNCTION:	This procedure analyzes the RETURN_CODE and other significant returned parameters from the SEND_DATA verb that sends the CNOS reply, and it determines whether to issue DEALLOCATE, and what TYPE to specify.
INPUT:	RETURN_CODE, REQUEST_TO_SEND_RECEIVED

```
Select based on the RETURN_CODE parameter returned from the SEND_DATA verb:
   When OK
      If the REQUEST_TO_SEND_RECEIVED parameter from the SEND_DATA verb is NO then
         Issue a DEALLOCATE verb with TYPE=SYNC_LEVEL to deallocate the conversation normally.
      Else
         Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the conversation.
   When RESOURCE_FAILURE_RETRY, RESOURCE_FAILURE_NO_RETRY, DEALLOCATE_ABEND_PROG,
    DEALLOCATE_NORMAL, DEALLOCATE_ABEND_SVC, DEALLOCATE_ABEND_TIMER
      Issue a DEALLOCATE verb with TYPE=LOCAL to deallocate the conversation locally.
   Otherwise
      Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate the conversation.
```

SESSION_LIMIT_DATA_LOCK_MANAGER

FUNCTION:	This procedure determines whether the specified MODEs exist, and if so, sets or resets the session-limit-data lock in the MODE entry to prevent simultaneous access by another CNOS transaction initiated at this or the partner LU.
INPUT:	The operation to be performed, identification of whether source or target LU issued the request, partner LU name and mode name, PARTNER_LU_LIST, and MODE_LIST
OUTPUT:	The state of the lock in affected MODE entries is updated
NOTE:	This procedure locks the MODE.

Referenced procedures, FSMs, and data structures:

LUCB	page A-1
PARTNER_LU	page A-2
MODE	page A-3

Select based on the requested locking operation:

When LOCK
Change the state of the lock (or locks) as described on page 5.4-30.

The four resulting lock states depend upon their previous lock state (if applicable) and the input that caused the transition to that state. For any input operation and current lock state combination not explicitly described, the state of the lock does not change.

If the CNOS command affects all MODEs for the PARTNER_LU then the lock is to be placed on all affected (LU,mode) entries. If any of the affected (LU,mode) entries has been previously LOCKED_BY_SOURCE, LOCK_DENIED is set for that mode name, but the others are left unlocked.

When UNLOCK
The state of the (LU,mode)-entry lock can be changed to the UNLOCK state only when the UNLOCK is attempted by the transaction program at the LU that currently has the entry locked.

> Note that, in the LOCK_DENIED state, the transaction program at the source LU has the lock on the (LU,mode) entry.
>
> If the CNOS command affects ALL MODEs, the UNLOCK is performed for all affected (LU,mode) entries.

This page intentionally left blank

CHAPTER 6.0. HALF-SESSION

GENERAL DESCRIPTION

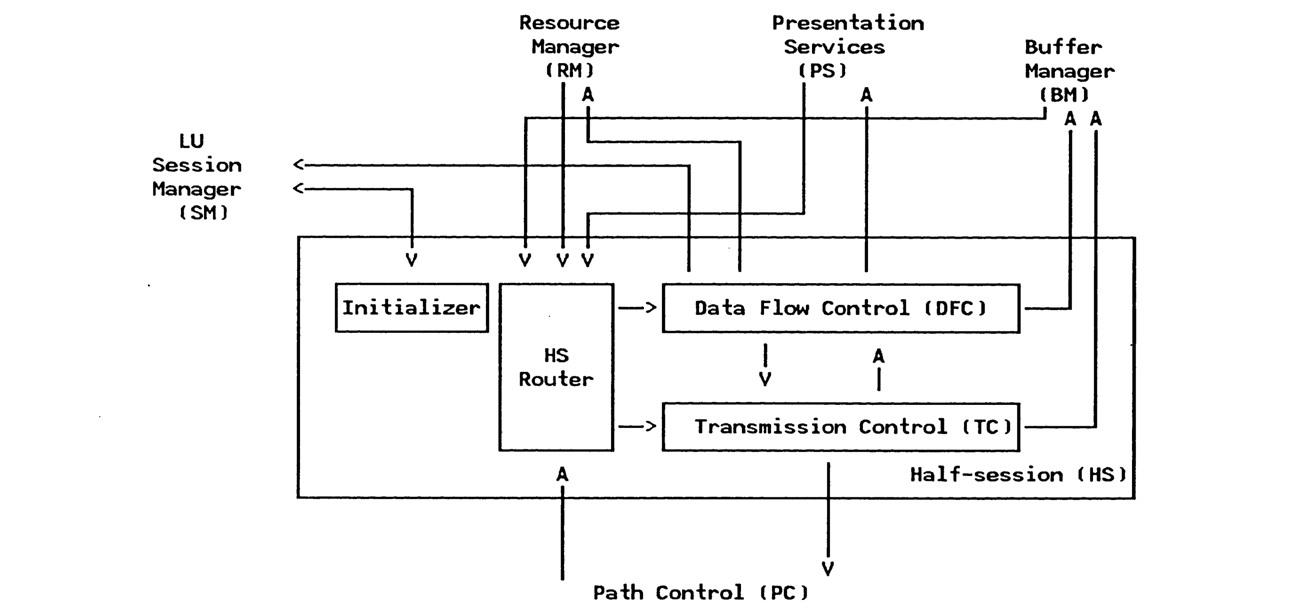


Figure 6.0-1. Overview of Half-Session

The half-session component (see Figure 6.0-1) resides in the LU and represents a session with another LU, which is an LU-LU session. The half-session's primary function is to control the data traffic flow for a session. It also performs initialization when activated and, when necessary, causes itself to be deactivated.

The components of the half-session are an initializer, a router, data flow control (DFC--see "Chapter 6.1. Data Flow Control"), and transmission control (TC--see "Chapter 6.2. Transmission Control"). The initializer records information from the session activation request (e.g., BIND) for later use by DFC and TC. The router distributes message units to DFC and TC. A record received from the LU session manager (SM--see "Chapter 4. LU Session Manager"), and message units received from the LU resources manager (RM--see "Chapter 3. LU Resources Manager") and from presentation services (PS--see "Chapter 5.0. Overview of Presentation Services") are routed to DFC. Message units received from path control (PC) are routed to TC. The primary functions of DFC are to translate between basic information units (BIUs) and records produced from transaction program verbs, and to control the flow of data between the half-session (HS) and PS, RM, and the buffer manager (BM). The primary function of TC is to control the flow of data between the half-session and path control.

The LU half-session is created by SM when a session-activation request (e.g., BIND) has been successfully processed. The half-session is destroyed by SM when (1) a session-deactivation RU(-RSP(BIND) or UNBIND) has been processed, or (2) a session route outage has occurred.

The half-session, RM, PS, SM, and PC are all separate processes. Message units are sent to HS by RM, PS, and PC. When a message unit arrives, HS may receive and process it. Another message unit cannot be received by HS until the current one is completely processed.

HS can selectively receive from these processes; e.g., when HS is waiting for a required reply or response from the partner HS, HS may elect to ignore messages from PS and process messages from only RM, SM, BM, and PC.

The protocol boundary between HS and BM is:

- GET_BUFFER (from HS to BM) - to obtain buffer
- FREE_BUFFER (from HS to BM) - to release buffer
- ADJUST_BUF_POOL (from HS to BM) - to change buffer pool size
- BUFFERS_RESERVED (from BM to HS) - to notify the buffer owner about the buffer allocation change

With each call from HS to the BM, appropriate parameters are also passed. The buffer manager (BM) sends a BUFFERS_RESERVED signal to HS, which contains the identifier of the buffer assigned. Upon receiving the signal from BM, transmission control updates the appropriate pacing counts and builds and sends the appropriate session-level pacing response (IPR or IPM). See Appendix B for more details on buffer manager function and services.

PROTOCOL BOUNDARIES BETWEEN HS AND OTHER COMPONENTS

Following is a list of the message units that flow between HS and other components.

Message units that flow from HS to RM: FM_HEADER (Attach or Security) BID BID_RSP FREE_SESSION BIS_RQ BIS_REPLY RTR_RQ RTR_RSP	Message units that flow from RM to HS: BID_WITHOUT_ATTACH RM_HS_CONNECTED BRACKET_FREED BID_RSP YIELD_SESSION BIS_RQ BIS_REPLY RTR_RQ RTR_RSP HS_PS_CONNECTED ENCIPHERED_RD2
Message units that flow from HS to PS: CONFIRMED RECEIVE_ERROR REQUEST_TO_SEND RSP_TO_REQUEST_TO_SEND	Message units that flow from PS to HS: CONFIRMED SEND_ERROR REQUEST_TO_SEND SEND_DATA_RECORD
Message units that flow from HS to SM: INIT_HS_RSP ABEND_NOTIFICATION ABORT_HS	Message units that flow from SM to HS: INIT_HS
Message units that flow from HS to PC: MU information containing session data to PC for transmission. The MU also contains the LFSID and the transmission priority.	Message units that flow from PC to HS: MU information containing session information to the appropriate HS.
Message units (modeled in this book as parameters in Call invocations) that flow from HS to BM: GET_BUFFER signal FREE_BUFFER signal ADJUST_BUF_POOL signal	Message units that flow from BM to HS: BUFFERS_RESERVED signal

Figure 6.0-2. Message Units Exchanged Between HS And Other Components.

FORMAL DESCRIPTION

HS

FUNCTION:	This procedure causes the half-session to be initialized and invokes the half-session router. On completion the HS process is destroyed.
INPUT:	At creation time, HS_CREATE_PARMS containing HS_ID (half-session identifier) and LU_ID (LU identifier); at run time, INIT_HS received from SM
OUTPUT:	INIT_HS_RSP sent to SM, HS_ID and LU_ID recorded for other procedures in the half-session. The half-session role (primary or secondary) is recorded from HS_CREATE_PARMS for use by other procedures in the half-session

Referenced procedures, FSMs, and data structures:

SM	page 4-48
RM	page 3-19
TC.INITIALIZE	page 6.2-13
DFC_INITIALIZE	page 6.1-19
PROCESS_LU_LU_SESSION	page 6.0-5
HS_CREATE_PARMS	page A-26
INIT_HS	page A-13
INIT_HS_RSP	page A-10
RM_HS_CONNECTED	page A-18
LOCAL	page 6.0-7

```
Record the HS_ID and LU_ID in the LOCAL data structure to make the
 information available to all half-session procedures.

From INIT_HS.TYPE, record an indication that this half-session is
 primary (PRI) or secondary (SEC).
Set LOCAL.SENSE_CODE to 0 (the no-error state).
From INIT_HS; record PERMANENT_BUFFER_POOL_ID in the LOCAL data structure.

Call TC.INITIALIZE(INIT_HS record) (page 6.2-13).
Call DFC_INITIALIZE(INIT_HS record) (page 6.1-19).

Initialize INIT_HS_RSP with LOCAL.SENSE_CODE and LOCAL.HS_ID.

If LOCAL.SENSE_CODE is 0 indicating that TC and DFC
 initialization were successful then
   Set the TYPE field of INIT_HS_RSP to POS and send the record to SM.
   Receive RM_HS_CONNECTED from RM.
   Call PROCESS_LU_LU_SESSION(page 6.0-5).
Else (initialization unsuccessful--LOCAL.SENSE_CODE indicates the type of error)
   Set the TYPE field of INIT_HS_RSP to NEG and send the record to SM.

Wait for the half-session to be destroyed.
```

During the processing in this chapter, a number of error conditions may be encountered. The following logic executes only if one of the detectable errors listed have been recognized. The following error condition may be detected:

- There is no buffer (permanent and demand) available to allow HS to send session data

SM	page 4-48
ABEND_NOTIFICATION	page A-25

Create and initialize an ABEND_NOTIFICATION record indicating HS abended.
Send the ABEND_NOTIFICATION record to SM.

PROCESS_LU_LU_SESSION

FUNCTION:	Does processing for half-session (FM profile 19). Message units received from RM and PS are routed to DFC. Message units received from PC are routed to TC. The half-session continues to operate until an error condition occurs or the half-session process is destroyed. If an error condition occurs, LOCAL.SENSE_CODE is set (by DFC or TC) with the sense data indicating what kind of error occurred. When this field is set, the half-session sends an ABORT message to SM. This causes SM to send an UNBIND(protocol error) for this session. HS receives BUFFERS_RESERVED signals from buffer manager (BM) and builds and sends the appropriate pacing response.
INPUT:	Message units received from PS, RM, BM, and PC; and LOCAL.SENSE_CODE may be set
OUTPUT:	ABORT_HS sent to SM if an error has been detected

Referenced procedures, FSMs, and data structures:

DFC_SEND_FROM_RM	page 6.1-21
DFC_SEND_FROM_PS	page 6.1-20
TRY_TO_RCV_SIGNAL	page 6.1-23
TC.RCV	page 6.2-23
BUFFERS_RESERVED_PROCESSING	page 6.2-31
FSM_BSM_FMP19	page 6.1-50
FSM_CHAIN_SEND_FMP19	page 6.1-53
ABORT_HS	page A-9
LOCAL	page 6.0-7


```
Do the following checks while no error has been detected.
   Wait for a record to be received on PS_TO_HS_Q, RM_TO_HS_Q, PC_TO_HS_Q or BM_TO_HS_Q.
   Check for a record on the PS_TO_HS_Q:
   If the PS_TO_HS_Q contains a record then
      If the session is not between brackets(FSM_BSM_FMP19 ≠ BETB) or
       half-session is not expecting a response(FSM_CHAIN_SEND_FMP19 ≠ PEND_RSP) or
       half-session is not expecting a reply(FSM_CHAIN_SEND_FMP19 ≠ PEND_RCV_REPLY) then
         Remove the first entry from the PS_TO_HS_Q.
         Call DFC_SEND_FROM_PS(record from PS) (page 6.1-20) to route
          the record to DFC.
      Else
         Wait for a record to be received on the RM_TO_HS_Q, PC_TO_HS_Q,
          or BM_TO_HS_Q.

   Check for a record on the RM_TO_HS_Q:
      If the RM_TO_HS_Q contains a record and no errors have been found then
         Receive the record from the RM_TO_HS_Q.
         Call DFC_SEND_FROM_RM(record from RM) (page 6.1-21)
          to route the record to DFC.

   Check for a record on the PC_TO_HS_Q:
      If the PC_TO_HS_Q contains a record and no errors have been found then
         Receive the record from the PC_TO_HS_Q.
         Call TC.RCV(record from PC) (page 6.2-23)
          to route the record to TC.
         (The input to those procedures is the received record.)

   Call TRY_TO_RCV_SIGNAL (page 6.1-23) to
    try to process a queued SIGNAL request.  Whether or not a queued SIGNAL
    request is processed depends on the state of the half-session.
    The state of the half-session may change each time a record is
    received and processed; therefore, the TRY_TO_RCV_SIGNAL
    procedure is called after each record is received so that it can check
    the current half-session state and process a SIGNAL request if necessary.

   Check for a record on the BM_TO_HS_Q:
      If the BM_TO_HS_Q contains a signal and no errors have been found then
         Receive the signal from the BM_TO_HS_Q.
         Call BUFFERS_RESERVED_PROCESSING(signal from BM, LOCAL.COMMON_CB)
          (page 6.2-31).
```

When LOCAL.SENSE_CODE is not 0 (error found) then
Send an ABORT_HS record to SM. (The ABORT_HS.SENSE_CODE comes from LOCAL.SENSE_CODE; ABORT_HS.HS_ID is the HS_ID saved during HS initialization.) (SM sends an UNBIND.)

LOCAL
This is the definition of the process data used by the half-session. This data may be accessed by any procedure in the half-session process.

```
LOCAL
   COMMON:  fields shared by all HS components
      HS_ID:  ID of this HS
      LU_ID:  the LU for this HS
      HALF_SESSION:  possible values:  PRI, SEC
      SENSE_CODE:  contains all 0's, if no error was detected;  otherwise,
       contains a nonzero sense data value
      PERMANENT_BUFFER_POOL_ID:  ID of the permanent buffer pool shared by all HSs
       within an LU
      RQ_CODE:  possible values:  CRV, BIS, LUSTAT, RTR, SIG, OTHER

   DFC:  fields used only by DFC
      LU_LU:  fields used for LU-LU sessions (FM profile 19)
         PS_ID:  ID of the PS associated with this HS
         BRACKET_ID:  ID of the current bracket
         FIRST_SPEAKER:  possible values:  YES, NO
         DIRECTION:  possible values:  HS_SEND, HS_RECEIVE
         CT_RCV:  contains correlation tables (see page 6.0-8)
         CT_SEND:  contains correlation tables (see page 6.0-8)
         SQN_SEND_CNT:  contains SNF (see page A-33)
         PHS_BB_REGISTER:  contains SNF (see page A-33)
         SHS_BB_REGISTER:  contains SNF (see page A-33)
         CURRENT_BRACKET_SQN:  contains SNF (see page A-33)
         RQD_REQUIRED_ON_CEB:  possible values:  YES, NO
         NORMAL_FLOW_RQ_COUNT:  the number of normal-flow requests sent by
          this HS
         SIG_RECEIVED:  possible values:  YES, NO
         SIG_SNF:  contains SNF (see page A-33)
         BETC:  possible values:  YES, NO
         SEND_ERROR_RSP_STATE:  indicates if a SEND ERROR negative response is owed;
          possible values:  RESET, NEG_OWED
         BB_RSP_STATE:  indicates if a BB response is owed;
          possible values:  RESET, POS_OWED, NEG_OWED
         BB_RSP_SENSE:  contains the sense data carried in a BB response,
          this field is valid only when BB_RSP_STATE=NEG_OWED
         RTR_RSP_STATE:  indicates if a RTR response is owed;
          possible values:  RESET, POS_OWED, NEG_OWED
         RTR_RSP_SENSE:  contains the sense data carried in a RTR response,
                          this field is valid only when RTR_RSP_STATE=NEG_OWED
         SIG_RQ_OUTSTANDING:  possible values:  YES, NO
         ALTERNATE_CODE:  possible values:  WILL_NOT_BE_USED, MAY_BE_USED
         SESSION_JUST_STARTED:  possible values:  YES, NO
         SAVED_MU_PTR:  pointer to the MU containing the RH and two bytes of RU data from
          the received RU
   TC:  fields used only by TC
      TCCB:  transmission control control block
         MAX_RCV_RU_SIZE:  the maximum RU size that can be received by this half-session
         SQN_RCV_CNT:  the number of normal-flow requests received by
          this half-session
         COMMON_CB:  contains the common control block for session-level
          pacing (see page 6.0-8)
         SEGMENTING_SUPPORTED:  the flag used to indicate BIU segmenting support
         CRYPTOGRAPHY:  possible values:  YES, NO
```

CT
Correlation table (CT) defines the send/receive RU operation. CT is contained in the half-session process data (LOCAL).

CT: fields used by DFC
 ENTRY_PRESENT: possible values: YES, NO
 SNF: contains the SNF of the latest RU received or sent for this chain (see page A-33)
 NEG_RSP_SENSE: 0 means no negative response received or sent; otherwise, contains the sense data from the negative response
 RH: contains request/response header (see SNA Formats).
 RQ_CODE: contains the valid RU request codes that can be received by the half-session; possible values: CRV, BIS, LUSTAT, RTR, SIG, OTHER

COMMON_CB
This is the definition of the common control block passed to the routines for session-level pacing. The calling process initializes the COMMON_CB during its initialization processing (except for the RESERVE_FLAG and UNSOLICITED_NWS).

COMMON_CB
 CALLER: the calling process
 PATH_CONTROL_ID: ID of the path control instance used by this LU
 LFSID: local-form session identifier associated with the HS (see page A-28)
 PERM_POOL_ID: ID of the permanent buffer pool to be used by HSs within this LU
 DYNAMIC_BUFFER_POOL_ID: ID of the dynamic buffer pool to be used by the half-session to receive normal-flow requests
 LIMITED_BUFFER_POOL_ID: ID of the limited buffer pool to be used by the half-session to send normal-flow requests
 TRANSMISSION_PRIORITY: possible values: LOW, MEDIUM, HIGH, NETWORK
 NUM_BUFS_PER_RU: number of buffers needed for each RU (always set to 1 by HS)
 RESERVE_FLAG: possible values: NO, ALL, MORE
 SEND_PACING: information needed for sending data from this LU to the partner LU
  TYPE: possible values: NONE, FIXED, ADAPTIVE
  RPC: number of MUs that can still be sent in this window
  NWS: size of the next window
  FIRST_WS: size of the first window when the session started (fixed window size when fixed pacing is used)
  SET_RLWI: indicates if the RLWI should be set to 1 (RLW) when the next pacing request is sent (always set to NOT_RLW for fixed pacing)
 RECEIVE_PACING: information needed for receiving data from the partner LU
  TYPE: possible values: NONE, FIXED, ADAPTIVE
  RPC: number of MUs that can still be received in this window
  NWS: size of the next window
  UNSOLICITED_IPM_OUTSTANDING: indicates if an IPM ACK is expected on from the partner LU; set to TRUE when waiting for an IPM ACK
  ADJUST_FOR_IPM_ACK_OUTSTANDING: indicates IPM ACK has been received but the limited buffer pool hasn't been adjusted yet
  UNSOLICITED_NWS: window size in an unsolicited IPM received from the partner LU

CHAPTER 6.1. DATA FLOW CONTROL

INTRODUCTION

The basic function of data flow control (DFC) component is to control the flow of data between half-sessions. DFC and FMD RUs flow through the DFC component; network control (NC) and session control (SC) RUs do not. All sessions use FM profile 19.

OVERVIEW OF DFC FUNCTIONS

DFC performs the following functions:

- Request/Response Formatting: DFC enforces correct RH parameter settings for FMD and DFC requests and responses.
- Chaining Protocol: Chaining is a means of sending or receiving a group of RUs for which there will be at most one response. DFC enforces the chaining protocol.
- Request/Response Correlation: DFC correlates responses with their associated requests.
- Request/Response Mode Protocols: Immediate request and immediate response modes are enforced by DFC.
- Send/Receive Mode Protocols: The normal-flow send/receive mode (half-duplex flip-flop) specifies a particular form of coordination between sending and receiving of normal-flow requests and responses.
- Bracket Protocols: Bracket protocols provide a means of sending or receiving a sequence of chains as a delimited transaction entity.
- Purging: When a bracket error negative response is sent for an incoming begin bracket (BB) chain, the remainder of that chain is purged.
- Buffer Management:
  - DFC specifies FREE_BUFFER in the Call to the buffer manager to release the buffer in which an MU was received or an error was detected.
  - DFC specifies ADJUST_BUF_POOL to allow the buffer manager to reduce the size of a limited buffer pool to reflect the correct send pacing count.
  - DFC specifies ADJUST_BUF_POOL to request the buffer manager to keep the same size of a dynamic buffer pool.
  - DFC specifies GET_BUFFER to request the buffer manager to allocate a permanent buffer for building an MU.

DFC STRUCTURE

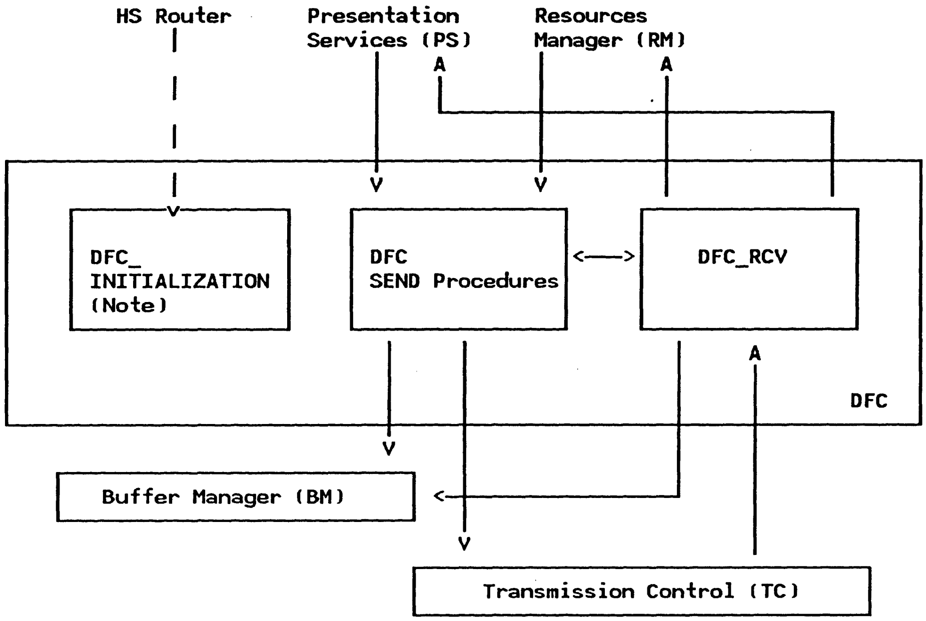
The DFC structure is shown in Figure 6.1-1 on page 6.1-2.

INITIALIZATION

The DFC initialization procedure is called by the half-session router (see "Chapter 6.0. Half-Session") at the activation of each session. It initializes FSMs and other protocol related parameters to be used during the session.

SEND

DFC procedures receive records (e.g., from presentation services and the resources manager), process the records, and send them on to transmission control (TC). The send processing consists of setting RH bits and MU header fields, and updating the states of DFC send FSMs.



Note: DFC is called by the half-session router (See "Chapter 6.0. Half-Session") at session-activation time.

Figure 6.1-1. Overview of DFC

RECEIVE

The DFC receive procedures (page 6.1-24, page 6.1-25) receive MU records from TC, process them, and send them on to PS or RM. When DFC receives records from RM or PS, it creates MUs to carry the information and sends to the DFC send procedure (page 6.1-27). DFC_RCV optionally checks the MU records for receive error conditions. These are conditions that occur only when the other half-session has violated the architecture. When DFC_RCV detects an error condition, it sets the sense code to indicate the error and returns to the half-session router. The router will then cause the half-session to be deactivated. If no receive errors are detected, the DFC_RCV processing consists mainly of updating the states of DFC receive FSMs (page 6.1-25).

TERMINATION

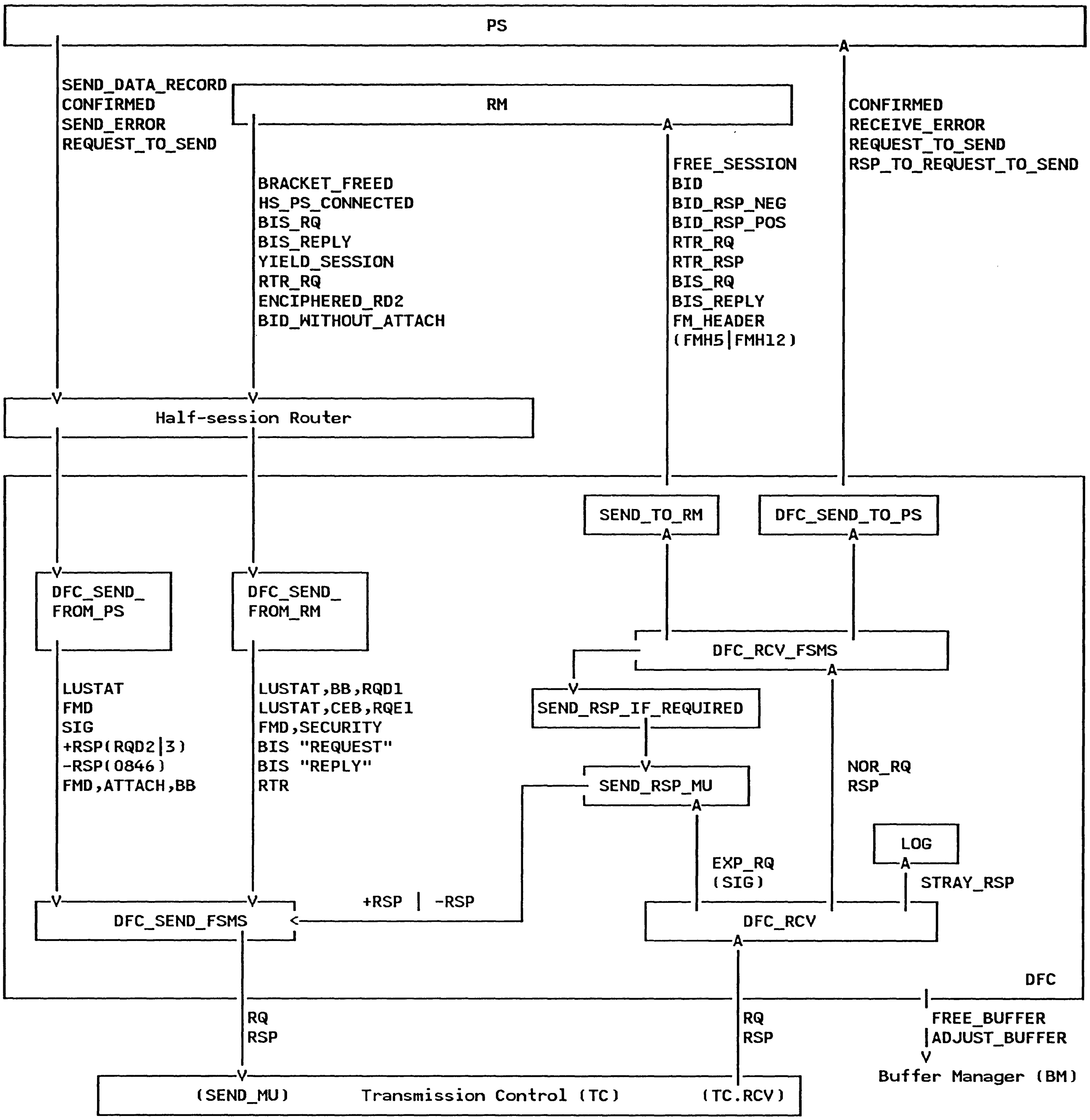
DFC and other half-session components stay active until a deactivation RU (UNBIND or -RSP(BIND)) flows. DFC causes an UNBIND to be sent when an error is detected. See Chapter 6.0.

PROTOCOL BOUNDARIES

DFC sends, receives, and processes records. The records DFC sends to, and receives from, RM and PS represent commands and replies unique to DFC's protocol boundaries with RM and PS. DFC maps the commands and replies it receives from RM and PS into MU records suitable for its processing; similarly, it maps MU records into commands and replies suitable for processing by RM and PS. The records DFC sends to, and receives from, TC are MU records that represent RU chains. See SNA Formats for details.

The protocol boundary information (records exchanged) is summarized in Figure 6.1-2 on page 6.1-3. The detailed specifications of the protocol boundaries with PS, RM, BM, and TC appear in the individual DFC procedures.

Throughout this chapter, references to request RUs and response RUs pertain to the MU records that represent the requests and responses. References to the sending or receiving of requests and responses pertain to the protocol boundary with TC, unless stated otherwise.



Note: DFC is called by the half-session router (See "Chapter 6.0. Half-Session") at session-activation time.

Figure 6.1-2. Detailed Structure and Protocol Boundaries of DFC

FUNCTION MANAGEMENT PROFILE 19

FM profiles are used to convey information about the protocols used on a session. FM profile 19 is used for half-sessions using LU 6.2 protocols. The DFC requests for this profile are BRACKET INITIATION STOPPED (BIS), LOGICAL UNIT STATUS (LUSTAT), READY TO RECEIVE (RTR), and SIGNAL (SIG). These requests are used to control the flow of data between the paired half-sessions and are described in "DFC Request and Response Descriptions" on page 6.1-16.

The FM usage settings in BIND are as follows:

- Chaining use (primary and secondary): multiple RU chains.
- Request control mode selection (primary and secondary): immediate.
- Form of response requested (primary and secondary): RQE or RQD.
- Compression indicator (primary and secondary): no compression.
- Send CEB indicator (primary and secondary): either end may send CEB.
- FM header usage: FMH-12(Security), FMH-5(Attach), and FMH-7(Error Description).
- Brackets: brackets are used and the reset state is in-brackets.
- Bracket termination rule: conditional termination.
- Alternate Code Set Allowed indicator: may or may not be used.
- Normal-flow send/receive mode: half-duplex flip-flop.
- Recovery responsibility: symmetric.
- Contention winner/loser: primary half-session (BIND sender) or secondary half-session (BIND receiver). The state is determined at BIND time (for parallel sessions, it is not negotiated). (See "Chapter 4. LU Session Manager") This determines who is bidder (contention loser) and who is first speaker (contention winner).
- Half-duplex flip-flop reset states: BIND sender is in send state after +RSP(BIND).

More detail of FM usage settings, and the formats and protocols implied by them, appear in the following pages.

USAGE ASSOCIATED WITH FM PROFILE 19

CONDITIONAL END BRACKET (CEB)

The CEB is used to indicate bracket termination. It is allowed only on an RH with EC. The bracket is terminated in all cases except when a -RSP to a (CEB,RQD2|3) chain leaves the session in-bracket (INB). The bracket terminates in all other circumstances. (See "Bracket Protocols" on page 6.1-10 for more details on bracket termination.)

FM HEADER USAGE

The Format indicator (FI), set to FMH, and RU Category (RU_CTGY) field, set to FMD, in the RH are used to indicate the presence of an FM header immediately following the RH. The FM headers for LU 6.2 are FMH-5(Attach), FMH-7(Error Description), and FMH-12(Security); See SNA Formats for details.

The FMH-5(Attach) may be carried in an RU with the Begin Chain indicator (BCI) set to BC. It may also be sent with BCI set to ¬BC when it is sent in an RU immediately following an FMH-12 that was (¬EC,¬CEB).

The FMH-7(Error Description) may appear in any RU in a chain at any time during the life of a bracket; it may be followed by data (i.e., it does not terminate the chain) or it may terminate a chain. The FMH-7 is not related to or bound by the chain state; it may be sent in a (BC,¬EC), (¬BC,¬EC), (¬BC,EC), or (BC,EC) request.

The FMH-12(Security) may flow only as the first RU after the session is initiated. If cryptography is in effect, the FMH-12 flows after the CRV exchange is complete. FMH-12 is always sent in a (BC,RQE1) request. The request may indicate either (EC,CEB) or (¬EC,¬CEB); the latter is used when the next request carries an FMH-5 with ¬BC.

USAGE OF DR1

DR1 is sent in a positive response to an RQD1 request in order to indicate that the requested function has been performed. The following are the only uses of DR1 in +RSP.

1. When the sender of Attach elects to bid for the session without sending an Attach, it may do so with an (RQD1,BB) LUSTAT(0006). The receiver sends the +DR1 when the session has been "allocated" to the sender. The only request that may follow this sequence is an FMH-5(Attach) to attach a transaction program or LUSTAT with (RQE1,CEB) to cancel the bid. (See "Chapter 3. LU Resources Manager" for more details on bidding.)

2. When RTR flows (RTR is always sent RQD1.)

3. When (RQD1,BB,CEB,Attach,data...) is received, i.e., a Bid with data

4. When (RQD1,CEB) is received as a result of the remote transaction program issuing the DEALLOCATE verb with the ABEND option

5. When (RQD1,CEB) is received at sequence numbering wrap points, as part of the stray SIGNAL and stray response logic (see "Stray SIGNALs and Responses")

SENDING RQE WITH BB FROM CONTENTION LOSER

The contention loser is allowed to send (RQE*,BB,CD,FMH-5,data) as a Bid.

USAGE OF RQE1, CEB, LUSTAT(0006)

Sessions are activated in the in-brackets (INB) state. If, for some reason, RM decides a newly activated session is not needed, it sends a YIELD_SESSION signal to DFC (see "Chapter 3. LU Resources Manager"). This results in an RQE1, CEB, LUSTAT(0006) being sent to terminate the unused bracket.

USAGE OF SIGNAL(X'00010001')

PS issues the REQUEST_TO_SEND command to DFC when the conversation is in receive state, requesting that the conversation be placed in send state (see "Send/Receive Mode Protocols" on page 6.1-11). SIGNAL always uses the REQUEST_TO_SEND code (X'00010001'). DFC then sends SIGNAL to the other half-session. When +RSP(SIG) is received, DFC passes the RSP_TO_REQUEST_TO_SEND reply up to PS. The conversation enters the send state when an RU carrying CD is received.

SEQUENCE NUMBERING OF REQUESTS AND RESPONSES

DFC assigns sequence numbers to DFC and FMD requests and responses, as follows:

- For normal-flow requests, the send sequence number count is incremented by 1 and then assigned to the request.

- A normal-flow BB response is assigned the sequence number of the corresponding BB request. The high-order bit is 0 if the bracket was started by the secondary half-session, or 1 if the bracket was started by the primary half-session.

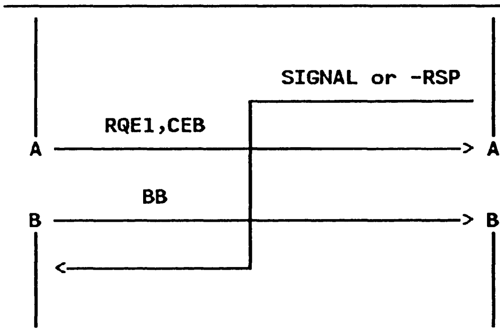
- SIGNAL (the only expedited-flow DFC request) and all responses are assigned the sequence number of the current bracket.

- A normal-flow RTR response is assigned the sequence number of the corresponding RTR request.

Figure 6.1-3 on page 6.1-6 illustrates an example of the use of sequence numbers. In this figure, some session control RUs (BIND, UNBIND, and CRV) are also illustrated.

STRAY SIGNALS AND RESPONSES

When a request is sent (RQE1,CEB) or (RQD*,CEB), a stray SIGNAL or response can occur. This happens because SIGNALs are expedited and are sent in receive state. A stray SIGNAL or response is one that is received outside the bracket it is intended for, and that could be disruptive if not eliminated or not recognized as a stray. SIGNALs received outside the intended bracket may be "early" or "late." "Early" SIGNALs are those received in a bracket that was started prior to the bracket in which the SIGNAL was generated. "Late" SIGNALs are those received in a bracket that was started after the bracket in which the SIGNAL was generated. Responses received outside the current bracket are always "late." Examples are shown in the following figures.



Bracket B gets the SIGNAL intended for A.

Figure 6.1-4. Case 1: "Late" SIGNAL or Response

```
Sending   Receiving
Sequence  Sequence
Number    Number     LUa                                             LUb
-------------------------------------------------------------------------
(Note 1)               ------------BIND------------------------------>o
          (Note 2)   o<--------------------------RSP(BIND)-------------
(Note 1)               ------------CRV------------------------------->o
          (Note 2)   o<---------------------------RSP(CRV)-------------
   1      (Note 3)     ------------Normal-flow RU ------------------->o
          (Note 4)   o<-----------------RSP(Normal-flow RU)------------
   2                   ------------Normal-flow RU ------------------->o
          (Note 4)   o<-----------------RSP(Normal-flow RU)------------
   3                   ------------Normal-flow RU ------------------->o
          (Note 4)   o<-----------------RSP(Normal-flow RU)------------
(Note 4)             o<------------SIGNAL-------------------------------
          (Note 4)     --------------------------RSP(SIGNAL)--------->o
   4                   ------------Normal-flow RU ------------------->o
          (Note 4)   o<-----------------RSP(Normal-flow RU)------------
(Note 1)               ------------UNBIND---------------------------->o
          (Note 2)   o<-----------------RSP(UNBIND)--------------------
(Note 1)               ------------BIND------------------------------>o
          (Note 2)   o<--------------------------RSP(BIND)-------------
   1                   ------------Normal-flow RU ------------------->o
          (Note 4)   o<-----------------RSP(Normal-flow RU)------------
   2                   ------------Normal-flow RU ------------------->o
          (Note 4)   o<-----------------RSP(Normal-flow RU)------------
   3                   ------------Normal-flow RU ------------------->o
          (Note 4)   o<-----------------RSP(Normal-flow RU)------------
                                        •
                                        •
                                        •
```

Notes

1. The sequence number in this case is an identifier, which can have any value 0-65535.

2. The sequence number in this case is an identifier, which has the same value as the request.

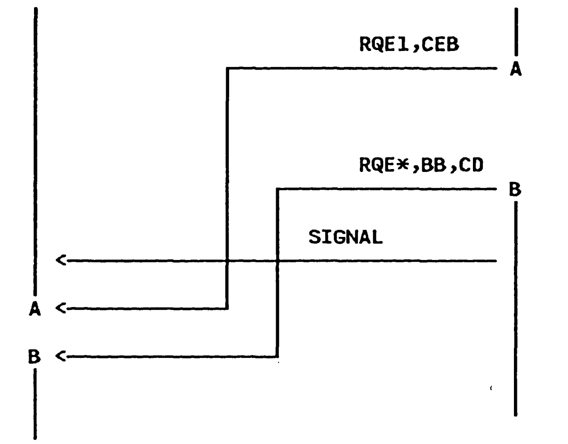
3. The first normal-flow RU following the BIND begins the first bracket. The session comes up in bracket for efficiency. The implicit bracket sequence number is 0, the sequence number of the first data RU is 1. After the first bracket is ended, subsequent brackets begin with a BB request. The bracket sequence number is the sequence number that flowed on the BB request.

4. The sequence number in this case is an identifier, which has the following properties:

   - The low-order 15 bits are the same as the low-order 15 bits of the sequence number that started the bracket.

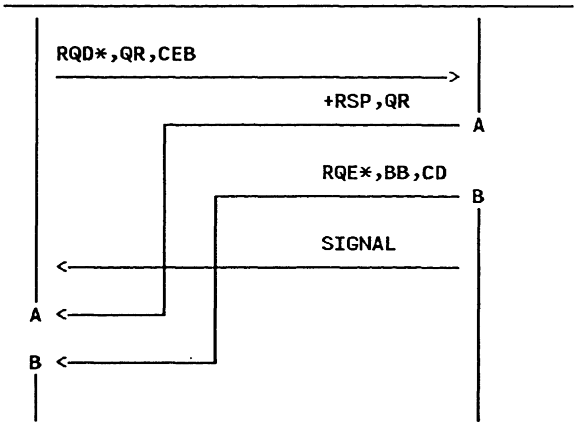
   - The high-order bit is 0 if the bracket was started by the secondary half-session, or 1 if the bracket was started by the primary half-session.

Figure 6.1-3. Use of Sequence Numbers



Bracket A gets the SIGNAL intended for B.

Figure 6.1-5. Case 2: "Early" SIGNAL



Bracket A gets SIGNAL intended for B.

Figure 6.1-6. Case 3: "Early" SIGNAL

The following subsections discuss how problems with strays are avoided.

Sending SIGNAL and Responses

Each LU eliminates problems with stray SIGNALs and stray responses by keeping three 16-bit "bracket ID" registers, a 1-bit switch, and a 15-bit normal-flow request counter:

- PHS_BB_REGISTER

  Bit 0: 1
  Bits 1-15: Low-order 15 bits of TH sequence number of last BB request sent by (or received from) primary half-session (PHS)

- SHS_BB_REGISTER

  Bit 0: 0
  Bits 1-15: Low-order 15 bits of TH sequence number of last BB request sent by (or received from) secondary half-session (SHS)

- CURRENT_BRACKET_SQN

  Bit 0: 1 = Bracket started by PHS
  0 = Bracket started by SHS
  Bits 1-15: Low-order 15 bits of TH sequence number of current bracket

- An indication that a definite response is required on the next CEB

  Bit 0: 0 = No RQD required on next CEB sent
  1 = RQD required on next CEB sent

- A count of normal-flow requests

  Bits 0-14: A count of the number of normal-flow requests sent and received since the last-sent (RQD,CEB)

When a normal-flow response (except for RSP(RTR)) or a SIGNAL is sent, DFC places the contents of the CURRENT_BRACKET_SQN register in the sequence number field (SNF) of the response or SIGNAL. The current bracket sequence is not used for RSP(RTR) because it does not flow within a bracket.

RQD required on CEB

RQD is required on some CEB requests to enable proper recognition of stray SIGNALs and stray responses. Since the CURRENT_BRACKET_SQN field is 15 bits, an identical value can occur after 2**15 RUs flow, causing the field to wrap. This can lead to confusion when recognizing stray SIGNALs and stray responses. In order to avoid this confusion, the normal flow is cleaned out periodically by the use of an (RQD,CEB) request and its response. This results in the following:

1. Whenever the count of normal-flow requests reaches 2**14, the indication that a definite response is required on the next CEB is set to YES.

2. Whenever the indication that a definite response is required on the next CEB is set to YES, the next CEB request is sent using RQD1, RQD2, or RQD3. The indication that a definite response is required on the next CEB is then reset to NO and the count of normal-flow requests is reset to 0. If DFC receives the CEB with an indication to send it RQE1 (e.g., the transaction program issued DEALLOCATE with the FLUSH option), DFC will change it to RQD1 in order to comply with this rule. When a response is received to an (RQD1,CEB) request, no information is

forwarded to PS because the transaction program is no longer communicating with the half-session.

Receiving SIGNAL Requests

When SIGNAL is received, the DFC component of the half-session does the following:

1. Validates the SIGNAL code--if it is other than request_to_send (X'00010001'), an UNBIND indicating protocol error (X'FE,10050000') is sent. The SIGNAL response is sent immediately. This creates the potential for receiving further SIGNALs before this one is processed. A 1-deep queue for SIGNAL is defined, so later SIGNALs overlay earlier ones. If overlaying occurs, the receiving transaction program gets only a single indication that a SIGNAL has been received, even though more than one SIGNAL has been sent. This is sufficient since all SIGNALs indicate request-to-send.

2. Places the SIGNAL in the correct bracket--the TH identifier field (SNF) is compared against the CURRENT_BRACKET_SQN register.

   - If they are equal, the SIGNAL is accepted and processed.

   - If the SIGNAL is early (see Figure 6.1-5 on page 6.1-7 and Figure 6.1-6 on page 6.1-7), it is pushed into the correct bracket by saving the SIGNAL value until the correct BB arrives, which can be several brackets in the future.

   - If the SIGNAL is late (see Figure 6.1-4 on page 6.1-5), it is discarded because the transaction program is no longer communicating with the half-session (i.e., the conversation has ended).

3. Reports receipt of the SIGNAL, via a REQUEST_TO_SEND record, to the PS component of the transaction's process. See "Chapter 5.1. Presentation Services--Conversation Verbs" for further discussion of the PS logic.

Receiving Responses

When a response is received, the DFC component:

1. Identifies failures--format checking and invalid sense code values are detected and a conversation failure is reported to PS and RM. An UNBIND(X'FE....') with sense data from the negative response is sent to terminate the session itself.

2. Detects stray negative responses--the TH identifier field (SNF) of the response is compared against the CURRENT_BRACKET_SQN register. If they are equal, the -RSP is intended for the current chain. If the -RSP is late (see Figure 6.1-4 on page 6.1-5), it is discarded because the transaction program the response is intended for is no longer communicating with the half-session. (If a positive response, other than +RSP(SIG), is not in the correct bracket, an UNBIND protocol error (X'FE,200E0000') is sent; +RSP(SIG) is discarded.)

3. Reports RTR responses--responses to RTR are reported to RM without regard for the bracket boundaries.

4. Reports responses to RQD1 requests--in general, responses to RQD1 requests, such as a Bid request (LUSTAT with (RQD1,BB)), are reported to RM; an exception is RSP(SIG), which is reported to PS.

5. Reports responses to RQD2 and RQD3 requests--responses to RQD2 and RQD3 requests are reported to PS.

SEND_ERROR PROCESSING

PS issues the SEND_ERROR command to DFC when PS is in HDX receive state, in order to change to send state so that it (PS) can send FMH-7(Error Description). (If already in send state, PS sends the FMH-7 without issuing the SEND_ERROR command; see "Chapter 5.0. Overview of Presentation Services" for more details.) Issuing SEND_ERROR in receive state causes DFC to send -RSP(ERP message forthcoming--X'0846') if some data has been received. If no data has been received, DFC waits until a chain is received and then responds with -RSP(X'0846').

After the EC request is received, PS can send the FMH-7(Error Description); the FMH-7 includes sense data for PS's use and is not processed by DFC. If the EC request ended the bracket, PS does not send the FMH-7.

DETAILED DESCRIPTION OF DFC FUNCTIONS

REQUEST/RESPONSE FORMATTING

DFC optionally checks that the requests and responses it receives are formatted correctly. The formatting checks involve:

- Enforcing that invalid RH bit combinations are not used, e.g., BBI=BB and BCI=¬BC, or CDI=CD and ECI=¬EC.
- Enforcing that the FM profile 19 rules are not violated, e.g., the receiving of an expedited-flow DFC request other than SIGNAL, or the receiving of a request with BB that is neither LUSTAT nor FMH-5 (Attach).

Format checks occur before the use of finite-state machines (FSMs). (State checks are checks that involve FSMs.) FSMs require the BIU record to be formatted correctly before processing it.

CHAINING PROTOCOL

Chaining provides a means to send (and receive) a sequence of requests as one entity in the context of error recovery. At most one response is sent per chain.

A chain consists of a single response RU or one or more request RUs with the following properties:

- The requests belong to the same flow (expedited or normal).
- The requests flow in the same direction.
- The first request is marked BC (Begin Chain) in the RH.
- The last request is marked EC (End Chain) in the RH.
- All requests that are neither first nor last are marked (¬BC, ¬EC) in the RH.

The checking of received requests for proper chaining is provided for each half-session.

Each response and each expedited-flow request is a single-RU chain, i.e., the RH indicates (BC,EC).

Only chains of the following types are sent:

- Exception-response (RQE) chain: Each request in the chain is marked exception-response.
- Definite-response (RQD) chain: The last request in the chain is marked definite-response; all other requests in the chain are marked exception-response.

See <u>SNA Formats</u> for details of the possible variations within each type.

The sender of the chain sets the Form of Response Requested bits properly in each request of the chain. Thus, the receiver of a chain need examine the Form of Response Requested bits only in the last request in a chain, or in a request in error.

Normal-flow DFC requests are not sent while sending a normal-flow FMD multiple-request chain.

If a chain sender receives a negative response to a chain being sent, the chain may be ended prematurely by sending the end-of-chain (EC) request.

REQUEST/RESPONSE CORRELATION

In order to remember the information on normal-flow chains that DFC sends or receives, DFC maintains two correlation entries: one for sent chains and one for received chains. There can never be more than one sent or received chain outstanding at any point in time (FM profile 19 protocol rules do not allow it), hence the need for only two entries. A correlation entry is established when the first RU in a chain is sent or received. The entry is reset when the chain has been completely processed, that is, when the end-of-chain request and its response, if any, have been processed. A correlation entry includes such information as selected RH parameters needed by DFC (e.g., RU category, BBI, and CEBI), and the DFC request code.

Some examples of how the correlation entry is used are:

- When receiving a response, the entry for the sent chain is checked to verify that the RU category in the response is the same as the RU category of the sent chain.
- When sending a response, the entry for the received chain is examined to determine whether a bracket has begun (i.e., the first RU in the chain was FMD with BBI=BB, or the single-RU chain was LUSTAT with BBI=BB).

REQUEST/RESPONSE MODE PROTOCOLS

Every half-session issues requests and responses according to the immediate request mode and the immediate response mode. Immediate request mode means that all request chains are sent under the constraint that no request may be sent by a given half-session when a previously sent request is still awaiting a response or reply. (A reply is a request sent in reaction to a received RQE request unit.) Request chains are replied or responded to in order of receipt. DFC enforces immediate request and response mode in the chaining FSMs.

Only two expedited RUs are used (SIG and CRV) and both use the immediate request mode. The two RUs flow at different times (when in use, the CRV exchange is complete before SIG is ever sent), and therefore the protocol can be enforced by the initiating components--DFC enforces the protocol for SIG, and TC enforces it for CRV.

The immediate response mode requires that responses be sent in the order the requests are received (i.e., requests are processed and responses issued first-in, first-out). When a response to a particular request is received, it means that all requests in the same flow sent before the responded-to request have been processed by the receiver, and that their responses, if any, have been sent.

BRACKET PROTOCOLS

A bracket is a sequence of normal-flow request chains and their responses, exchanged in either or both directions between two half-sessions. Bracket protocols allow contention for session resources and assist in resolving the race condition that can result from that contention.

The primary use of brackets is to carry conversations between transaction programs. A transaction program requests a conversation with another transaction program by issuing the ALLOCATE verb. ALLOCATE causes the resources manager (RM) to select a half-session (based on ALLOCATE parameters) and attempt to initiate a bracket on it. If the bracket is successful, that half-session is used to carry the conversation. (See "Chapter 3. LU Resources Manager" for more details.) A transaction program ends a conversation by issuing a DEALLOCATE verb. This causes the half-session to terminate the bracket carrying the conversation. When the bracket terminates, the half-session becomes available again for selection by RM.

A bracket is delimited by setting BBI to Begin Bracket (BB) in the first request of the first chain, and CEBI to Conditional End Bracket (CEB) in the last request of the last chain in the bracket.

BIND parameters specify one of the half-sessions as first speaker and the other as bidder. The first speaker has the freedom to begin a bracket without requesting permission from the other half-session to do so. Any request carrying BB sent by the first speaker will begin a bracket. The bidder must request and receive permission from the first speaker to begin a bracket. The bracket protocols are verified by the bracket state manager in the receiving half-session.

The bidder may attempt to initiate a bracket (i.e., Bid) by sending an FMD request chain with (RQD,BB,QR) or with (RQE,BB,CD,QR). (See "Queued Response Protocol" on page 6.1-12 for description of QR usage.) The first speaker grants the attempt via a reply to an (RQE,CD) (see "Send/Receive Mode Protocols" on page 6.1-11 for definition of reply) or a positive or negative response (other than X'0813', X'0814', or X'088B') or refuses the attempt via negative response (X'0813', X'0814', or X'088B').

A negative response with sense code X'0813', X'0814', or X'088B' indicates that the first speaker has denied permission for the bidder to begin a bracket. A Ready_To_Receive (RTR) request may be sent later by the first speaker when permission to start a bracket is granted. (The first speaker may or may not have the capability to subsequently send RTR. The X'0814' sense code is used only when the first speaker has the capability to send RTR.) If the first speaker will send RTR later, the sense code with the negative response is X'0814' (Bracket Bid Reject--RTR

Forthcoming). In this case, the bidder waits for the RTR before sending another BB. If the RTR will not be sent, the sense code is either X'0813' (Bracket Bid Reject--No RTR Forthcoming) or X'088B' (BB Not Accepted--BIS Reply Requested). In the X'0813' case, the bidder will send BB again, if it still wants to begin a bracket. In the X'088B' case, the BB is not sent again because no more conversations will be allowed to start. A BIS request will be received shortly and a BIS reply will be sent.

Expedited requests and responses are not affected by bracket indicators on normal-flow requests, nor by the states of the bracket FSMs.

BRACKET RULES

The following rules apply to the bracket indicators:

- BB may be indicated only on the first (or only) request of the first chain.
- CEB may be indicated only on the last (or only) request of the last chain. It indicates the last chain in the bracket. (If CEB is set, CD must not be indicated because CEB overrides CD.)
- BB and CEB may both be indicated within the same chain.
- BB or CEB may be indicated by either half-session.
- BB or CEB may be indicated on FMD requests.
- Neither BB nor CEB may be indicated on any normal-flow DFC request except LUSTAT.
- Neither BB nor CEB may be indicated on responses or on expedited requests.

The following bracket termination rule is used:

- Bracket Termination Rule: Bracket termination is influenced by whether the RU carrying CEB is an RQE, RQD1, or RQD2|3, request. If the request is RQD2|3 the bracket terminates only upon receipt of a positive response; a negative response to the chain causes the session to remain in bracket. If the RU is sent as RQE, the bracket terminates unconditionally upon the sending of that RU. A negative response to an (RQE,CEB) request will not find an entry in the receive correlation entity, and therefore is logged and discarded. If the RU is specified as RQD1, the bracket terminates unconditionally upon receipt of a response to the chain, whether the response is positive or negative. RQD1 is generated by DFC for the sequence number wrap case (unless the request is already RQD2|3) and for the DEALLOCATE TYPE(ABEND_*) verb. When DFC uses RQD1, PS and the transaction program consider the conversation to be terminated when the DEALLOCATE TYPE(ABEND_*) or DEALLOCATE TYPE(FLUSH) verb is issued: PS and the TP don't expect a response. DFC waits for a response before informing RM that the session is available for a new conversation.

No more than one BB can be outstanding from a half-session unless the LU is the first speaker and is not waiting for any type of response or reply.

The normal-flow DFC requests, RTR and BIS, may be sent only between brackets and do not carry bracket bits. FMD requests always carry BB when flowing between brackets. LUSTAT is treated exactly like an FMD request containing (BC,EC), and may be used with BB to bid for, or with CEB to end, a bracket.

The following types of error conditions are detected in the management of brackets:

- Bracket protocol errors detected at the receiver and caused by sender error.
- Errors detected at the receiver and caused by race conditions. The appropriate action is for the receiver to send a Bracket Bid Reject sense code (X'0813', X'0814', or X'088B') on a negative response to the other half-session. A retry of the operation may be necessary.

SEND/RECEIVE MODE PROTOCOLS

Once a bracket has started, the normal-flow send/receive mode protocol is half-duplex flip-flop (HDX-FF). One half-session is designated HDX-FF bidder, and the other, HDX-FF first speaker. Parameters in BIND specify which half-session is first speaker and which is bidder. The bidder may send a request containing BB, but its bid for the bracket is pending until it receives a response.

Once a bracket is begun, a half-duplex flip-flop state is established, and the sender issues normal-flow requests and the receiver issues responses. When the sender completes its transmission of normal-flow requests, it transfers control of sending to the other half-session by setting the Change Direction indicator to CD on the last request sent. See "Bracket Protocols" on page 6.1-10 for additional details.

The Change Direction indicator (CDI) is used in the HDX-FF protocols. Only a request on the normal flow that is marked End Chain may carry CDI=CD. When the sending half-session includes CD in a request, it indicates that it is prepared to receive and that its paired half-session may send. CD is not conveyed in a response or on a request that carries CEB.

An exception-response (RQE) chain always has CD indicated on the last RU of the chain, unless that RU carries CEB, in which case it does not indicate CD.

A "reply" is the request sent by a half-session immediately after receiving an (RQE,CD) chain. A reply is treated as implicitly containing a positive response. That is, once an (RQE,CD) chain is replied to, a negative response to that chain is not permitted. A BIS, RTR, or an RU carrying BB is not treated as a reply.

QUEUED RESPONSE PROTOCOL

DFC enforces the setting of the Queued Response Indicator (QRI) bit (in RH) on requests. The setting of the QRI bit is the same for all RUs in a chain. See SNA Formats for a discussion of this RH indicator.

QR is always indicated on a chain carrying BB that is sent by the bidder. When QR is indicated in a response, that response will not pass any other RUs flowing through the network on the same session. It is used so that a positive response to the bidder's BB chain will not interfere with a bracket sent earlier by the first speaker. The positive response will be received after the first speaker's bracket ends. QR is not indicated on any other chain.

PS SEND AND RECEIVE RECORDS

This section describes how the SEND_DATA_RECORD (sent from PS to HS) and the MU (sent from HS to PS) are mapped to and from the RH portion of a BIU containing a request RU. The SEND_DATA_RECORD is used by PS to send data in accordance with the verbs issued by a transaction program. This record (see "Chapter 5.0. Overview of Presentation Services" for details.) is mapped into a request BIU by DFC before being sent. The MU sent from the half-session to PS carries the data received on the half-session and is mapped from a received BIU containing a request. Figure 6.1-7 on page 6.1-13 summarizes the SEND_DATA_RECORD to RH mapping and Figure 6.1-8 on page 6.1-13 summarizes the RH to MU (send from HS to PS) mapping.

Parameters in SEND_DATA_RECORD (from PS to HS)	Request RH indicators
ALLOCATE=YES (see Note 1) FMH=YES (see Note 1)	BB FMH
FLUSH	¬EC,RQE1
CONFIRM PREPARE_TO_RECEIVE_CONFIRM_SHORT PREPARE_TO_RECEIVE_CONFIRM_LONG	EC,RQD3 EC,CD,RQD3 EC,CD,RQE3
PREPARE_TO_RECEIVE_FLUSH DEALLOCATE_CONFIRM DEALLOCATE_FLUSH with DEALLOCATE_ABEND_* FM header (see Note 3) DEALLOCATE_FLUSH without DEALLOCATE_ABEND_* FM header (see Note 3)	EC,CD,RQE1 EC,CEB,RQD3 EC,CEB, RQD1 EC,CEB, RQE1

Notes:

1. This parameter is used in conjunction with the rest of the parameters (e.g., if ALLOCATE is YES and FMH is YES, specified with DEALLOCATE_CONFIRM, the request RH indicators are BB,FMH,EC,CEB,RQD3).

2. RH indicators not shown (e.g., QRI) are set independently from the SEND_DATA_RECORD parameters.

3. To indicate a DEALLOCATE_ABEND_* action, FMH is set to YES and DATA (offset 2 through 4) is set to X'070864'.

Figure 6.1-7. Mapping from SEND_DATA_RECORD to request RH

Request RH indicators	Parameters set in MU (sent from HS to PS)
FMH	FMH=YES (see Note 1)
¬EC	NOT_END_OF_DATA
EC,RQD2|3 EC,CD,RQ*2|3 EC,CD,RQE1	CONFIRM PREPARE_TO_RECEIVE_CONFIRM PREPARE_TO_RECEIVE_FLUSH
EC,CEB,RQD2|3 EC,CEB,RQE1 or RQD1	DEALLOCATE_CONFIRM DEALLOCATE_FLUSH

Notes:

1. This parameter is set in conjunction with the rest of the parameters (e.g., if FMH,EC,CEB,RQD2|3 are indicated in the RH, FMH is YES and DEALLOCATE_CONFIRM is indicated in the MU sent to PS).

2. Other RH indicators (e.g., QRI) have no effect on the MU parameter settings.

Figure 6.1-8. Mapping from request RH to MU (sent to PS)

DFC REQUEST AND RESPONSE FORMATS

This section describes the DFC request and response formats; the RH formats are shown in this section; the RU formats are shown in SNA Formats. Figure 6.1-9 and Figure 6.1-10 on page 6.1-15 show the format of DFC requests and responses, respectively. The Expedited Flow indicator (EFI in the TH) shows which flow, expedited or normal, the DFC request or response flows on.

DFC Request -----> Header Indicators			BIS	RTR	LUSTAT	SIGNAL
TH		EFI	Normal	Normal	Normal	Exp
RH Byte 0	Bit 0	RRI	RQ	RQ	RQ	RQ
	Bits 1-2	RU category	DFC	DFC	DFC	DFC
	Bit 3	reserved	0	0	0	0
	Bit 4	FI	1	1	1	1
	Bit 5	SDI	*SD	*SD	*SD	*SD
	Bit 6	BCI	BC	BC	BC	BC
	Bit 7	ECI	EC	EC	EC	EC
RH Byte 1	Bit 0	DR1I	DR1	DR1	*DR1	DR1
	Bit 1	reserved	0	0	0	0
	Bit 2	DR2I	*DR2	¬DR2	*DR2	¬DR2
	Bit 3	ERI	ER	¬ER	*ER	¬ER
	Bit 4	reserved	0	0	0	0
	Bit 5	reserved	0	0	0	0
	Bit 6	QRI	¬QR	¬QR	*QR	¬QR
	Bit 7	PI	*PAC	*PAC	*PAC	¬PAC
RH Byte 2	Bit 0	BBI	¬BB	¬BB	*BB	¬BB
	Bit 1	EBI	¬EB	¬EB	¬EB	¬EB
	Bit 2	CDI	¬CD	¬CD	*CD	¬CD
	Bit 3	reserved	0	0	0	0
	Bit 4	reserved	0	0	0	0
	Bit 5	reserved	0	0	0	0
	Bit 6	reserved	0	0	0	0
	Bit 7	CEBI	¬CEB	¬CEB	*CEB	¬CEB

Notes:

1. *XX means either XX or ¬XX.
2. See SNA Formats for complete RH description.
3. For LUSTAT: If CEBI is set to CEB, CDI is set to ¬CD.
4. For LUSTAT: (DR1I,DR2I) = (0,1) | (1,0) | (1,1).
5. For LUSTAT: QRI is set to QR when BBI is set to BB.
6. The SNF and DCF TH fields are also set by DFC.
7. See SNA Formats for a complete TH description.

Figure 6.1-9. DFC Request Formats

DFC Response-----> Header Indicators			RSP(BIS)	RSP(RTR)	RSP(LUSTAT)	RSP(SIGNAL)
TH		EFI	Normal	Normal	Normal	Exp
RH Byte 0	Bit 0	RRI	RSP	RSP	RSP	RSP
	Bits 1-2	RU category	DFC	DFC	DFC	DFC
	Bit 3	reserved	0	0	0	0
	Bit 4	FI	1	1	1	1
	Bit 5	SDI	*SD	*SD	*SD	¬SD
	Bit 6	BCI	BC	BC	BC	BC
	Bit 7	ECI	EC	EC	EC	EC
RH Byte 1	Bit 0	DR1I	DR1	DR1	*DR1	DR1
	Bit 1	reserved	0	0	0	0
	Bit 2	DR2I	*DR2	¬DR2	*DR2	¬DR2
	Bit 3	RTI	±	±	±	+
	Bit 4	reserved	0	0	0	0
	Bit 5	reserved	0	0	0	0
	Bit 6	QRI	¬QR	¬QR	*QR	¬QR
	Bit 7	PI	*PAC	*PAC	*PAC	¬PAC
RH Byte 2	Bit 0-7	reserved	0...0	0...0	0...0	0...0

Notes:

1. *XX means either XX or ¬XX.

2. See SNA Formats for complete RH description.

3. For LUSTAT: DR1I, DR2I, and QRI are set the same as they were on the request.

4. The SNF and DCF TH fields are also set by DFC.

5. See SNA Formats for a complete TH description.

Figure 6.1-10. DFC Response Formats

The DFC requests for FM profile 19 are described below.

BIS (BRACKET INITIATION STOPPED)

Flow: Primary to secondary and secondary to primary (Normal)

Principal FSM:
None in DFC

BIS is sent by a half-session to indicate that it will not attempt to begin any more brackets (i.e., send any more BB requests).

The use of BIS and its principal FSMs are described in "Chapter 3. LU Resources Manager".

LUSTAT (LOGICAL UNIT STATUS)

Flow: Primary to secondary and secondary to primary (Normal)

Principal FSM:
Uses same FSMs as normal-flow data

LUSTAT is used to accompany RH bits. The status value is set to X'0006'. Specifically, LUSTAT is used in place of a null RU; that is, when it is time to send an RU to DFC, and the RU is marked (BC,EC) and has RU length = 0, an LUSTAT(0006) is sent instead. This results in the following RH encoding with LUSTAT(0006):

1. (RQD1,BB): Sending half-session bids without data.
2. (RQE2,CD): Sending half-session transfers send control to the other half-session, specifies that a Confirm be taken, and that completion of the Confirm be indicated by receipt of the next request from the other half-session. Confirm--means that the transaction program connected to the other half-session has received and processed the RU data successfully.
3. (RQD2,CD): Same as 2, except that completion of the Confirm will be indicated by receipt of +RSP.
4. (RQE1,CD): Sending half-session transfers send control to the other half-session specifying no Confirm.
5. (RQD2,CEB): Same as 3, plus the bracket will be terminated when a +RSP is received.
6. (RQE1,CEB): Same as 4, plus the bracket is terminated unconditionally.

RTR (READY TO RECEIVE)

Flow: First speaker to bidder (Normal)

Principal FSM:
None in DFC

RTR indicates to the bidder that the bidder can now initiate a bracket. An RTR request is sent only by the first speaker (see "Bracket Protocols" on page 6.1-10). The use of RTR and the RTR bit (in SCB) setting are described in "Chapter 3. LU Resources Manager".

SIG (SIGNAL)

Flow: Primary to secondary and secondary to primary (Expedited)

Principal FSM:
None in DFC

SIG is an expedited request that can be sent between half-sessions, regardless of the status of the normal flows. It is the only expedited DFC request defined for FM profile 19. It carries a four-byte value, of which the first two bytes are the signal code and the last two bytes are the signal extension value.

The only signal code defined for use with FM profile 19 is X'00010001'. This signal code is used in conjunction with the PS command REQUEST_TO_SEND. See "Chapter 5.1. Presentation Services--Conversation Verbs" for more details.

DFC_INITIALIZE

FUNCTION:	This procedure initializes fields in the half-session's local storage (for process data) that are used by DFC. This procedure is called by the half-session router ("Chapter 6.0. Half-Session") when the half-session is created.
INPUT:	INIT_HS, indicating that the half-session is first speaker or bidder
OUTPUT:	DFC local process data fields are initialized and MU information is recorded locally
NOTES:	1. LOCAL.HALF_SESSION is set to indicate that the half-session is primary or secondary 2. When a half-session is activated, it comes up in-bracket (INB). The first data BIU sent on the session uses a value of X'0001' in the TH sequence number field and does not carry BB. The first BB was implied rather than sent. Therefore, the current bracket sequence number (LOCAL.CURRENT_BRACKET_SQN) associated with the first bracket on a session is initialized to 0.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
FSM_BSM_FMP19	page 6.1-50
FSM_RCV_PURGE_FMP19	page 6.1-56
FSM_QRI_CHAIN_RCV_FMP19	page 6.1-55
FSM_CHAIN_RCV_FMP19	page 6.1-51
FSM_CHAIN_SEND_FMP19	page 6.1-53
LOCAL	page 6.0-7
INIT_HS	page A-13
MU	page A-29


```
(Record information from the input INIT_HS record that
 will be used by DFC throughout the life of this session.)
Set LOCAL.FIRST_SPEAKER to indicate if this HS is a first speaker or bidder.
Set LOCAL.ALTERNATE_CODE to indicate if an alternate code is allowed.
Reset correlation table entries.
Set LOCAL.SQN_SEND_CNT.SQN to 0.
Set LOCAL.CURRENT_BRACKET_SQN.BRACKET_STARTED_BY to PRI.
Set LOCAL.CURRENT_BRACKET_SQN.NUMBER to 0. (Note 2)
Set LOCAL.PHS_BB_REGISTER.BRACKET_STARTED_BY to PRI.
Set LOCAL.PHS_BB_REGISTER.NUMBER to 0.
Set LOCAL.SHS_BB_REGISTER.BRACKET_STARTED_BY to SEC.
Set LOCAL.SHS_BB_REGISTER.NUMBER to 0.
Set LOCAL.RQD_REQUIRED_ON_CEB to NO.
Set LOCAL.NORMAL_FLOW_RQ_COUNT to 0.
Set LOCAL.SIG_RECEIVED to NO.
Set LOCAL.SIG_SNF.SQN to 0.
Set LOCAL.PS_ID to NULL.
Reset all the FSMs in this chapter to FSM state 1.
Set LOCAL.BETC to YES.
Set LOCAL.SEND_ERROR_RSP_STATE to RESET.
Set LOCAL.BB_RSP_STATE to RESET.
Set LOCAL.RTR_RSP_STATE to RESET.
Set LOCAL.SIG_RQ_OUTSTANDING to NO.

If LOCAL.HALF_SESSION is PRI then
   Set LOCAL.SESSION_JUST_STARTED to YES (indicates that the current bracket sequence number
    has already been initialized).
Else (secondary half-session)
   Set LOCAL.SESSION_JUST_STARTED to NO.
Save addressability to the MU in LOCAL for later use, before passing it on to PS.
```

DFC_SEND_FROM_PS

FUNCTION:	Process the record received from presentation services (PS) and determine the proper response (positive or negative) or MU (data or signal) that needs to be sent to the partner HS via transmission control (TC). If an error is found while processing the PS_TO_HS record, the buffer will be freed by this procedure.
INPUT:	MU, containing a PS_TO_HS record (the record type may be SEND_DATA_RECORD, CONFIRMED, SEND_ERROR, or REQUEST_TO_SEND); LOCAL.CT_RCV, indication of the form-of-response-requested for the last chain received, if the record type is CONFIRMED; FSM_CHAIN_RCV_FMP19, indication of the state of the FSM; LOCAL.BRACKET_ID, if the record type is SEND_ERROR
OUTPUT:	LOCAL.SEND_ERROR_RSP_STATE may be set to indicate that a negative response may be sent to the next chain, a response (negative or positive), data or an MU contains REQUEST_TO_SEND signal may be sent to TC
NOTE:	PS initializes the appropriate fields in an MU before sending it to the HS

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
SEND_RSP_MU	page 6.1-45
SEND_FMD_MU	page 6.1-43
INITIALIZE_TH_RH	page 6.1-38
DFC_SEND_FSMS	page 6.1-27
FSM_CHAIN_RCV_FMP19	page 6.1-51
MU	page A-29
LOCAL	page 6.0-7
SIGNAL_RQ_RU	SNA Formats


```
If MU.PS_TO_HS.BRACKET_ID ≠ LOCAL.BRACKET_ID then
   Log information concerning the error in the system log.
   Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
    containing the erroneous MU.
Else
   Select based on PS_TO_HS record type:
      When SEND_DATA_RECORD
         Call SEND_FMD_MU(MU) (page 6.1-43) to send the MU.
      When CONFIRMED
         If LOCAL.CT_RCV shows that the last request received indicated RQD2 or RQD3
          (short lock, need to response immediately) then
            Call SEND_RSP_MU(NULL,NORMAL,POS,X'00000000') (page 6.1-45)
             to send a normal-flow positive response.
         Else (long lock, the response is delayed)
            Call buffer manager (FREE_BUFFER, buffer address) to release
             the buffer containing the MU.  (Appendix B)
      When SEND_ERROR
         If the state of FSM_CHAIN_RCV_FMP19 is BETC
          (while between chains, no response may be sent) then
            Set LOCAL.SEND_ERROR_RSP_STATE to NEG_OWED to indicate that a negative
             response should be sent to the next RU received.
            Call buffer manager (FREE_BUFFER, buffer address) to release
             the buffer containing the MU.  (Appendix B)
         Else (within the INC state, send -RSP to chain currently being processed)
            Call SEND_RSP_MU(NULL,NORMAL,NEG,X'08460000') (page 6.1-45)
             to send a normal-flow negative response.

      When REQUEST_TO_SEND
         Call INITIALIZE_TH_RH(MU) (page 6.1-38) to
          set the TH and RH fields to default values.
         Set the TH and RH fields as described in Figure 6.1-9 on page 6.1-14 for a
          SIGNAL request.
         Set RU to SIGNAL_RQ_RU (as described under SIG request in SNA Formats).
         Set MU.DCF to the length of (RH + RU).
         Set LOCAL.COMMON.RQ_CODE to SIG.
         Call DFC_SEND_FSMS(MU) (page 6.1-27) to maintain states while sending
          request or response.
```

DFC_SEND_FROM_RM

FUNCTION:	Process records received from the resources manager (RM). This procedure is called by the half-session router ("Chapter 6.0. Half-Session").
INPUT:	Record from RM (BID_WITHOUT_ATTACH, BIS_REPLY, BIS_RQ, HS_PS_CONNECTED, BRACKET_FREED, RTR_RQ, YIELD_SESSION, or ENCIPHERED_RD2); indication that session just started, LOCAL.SESSION_JUST_STARTED; primary or secondary half-session indicator, LOCAL.HALF_SESSION; LOCAL.SQN_SEND_CNT.NUMBER; possibly, an HS_PS_CONNECTED or BRACKET_FREED record from RM
OUTPUT:	The following RUs may be sent: Bid with Attach (carrying BB), Bid with LUSTAT (carrying BB), BIS, RTR, or a Yield Session with LUSTAT (carrying CEB). In addition, the PS_ID is recorded to identify the PS that is using this HS, and an indication that the session just started is also recorded.
NOTE:	The records received from RM are not MU records. Half-session builds an MU record and copies the information from the non-MU record (received from RM) and sends it to TC. Limited buffers are used to send normal-flow requests. However, half-session uses a permanent buffer for building and sending this MU instead of requesting a limited buffer because this MU cannot be held (waiting for a limited buffer to become available). However, the limited buffer pool needs to be adjusted (decremented by 1) to reflect the correct size of the send pacing count.

Referenced procedures, FSMs, and data structures:

SEND_FMD_MU	page 6.1-43
DFC_SEND_FSMS	page 6.1-27
INITIALIZE_TH_RH	page 6.1-38
FSM_BSM_FMP19	page 6.1-50
LOCAL	page 6.0-7
MU	page A-29
BID_WITHOUT_ATTACH	page A-17
BRACKET_FREED	page A-18
ENCIPHERED_RD2	page A-18
HS_PS_CONNECTED	page A-18
BIS_REPLY	page A-12
BIS_RQ	page A-12
YIELD_SESSION	page A-19
RTR_RQ	page A-12

```
Select based on type of the record from RM:
   When BID_WITHOUT_ATTACH
      Call buffer manager (GET_BUFFER, permanent buffer pool ID, no wait) to
       get a buffer for building an MU (containing the LUSTAT).  (Appendix B)
      Create an MU and Call INITIALIZE_TH_RH(MU) to set the TH and RH fields
       to default values  (page 6.1-38).
      Call buffer manager (ADJUST_BUF_POOL, limited buffer pool ID, change amount)
       to reduce the size of the limited buffer pool (Appendix B).
       The change amount is a negative value of 1 (see note).
      Set the RH fields as described in Figure 6.1-9 on page 6.1-14 for
       an LUSTAT request.
      Set the RU to LUSTAT_RQ_RU as described in SNA Formats.
      Call DFC_SEND_FSMS(MU) (page 6.1-27).

   When BIS_REPLY
      Call buffer manager (GET_BUFFER, permanent buffer pool ID, no wait) to
       get a buffer for building an MU (containing the BIS).  (Appendix B).
      Create an MU and Call INITIALIZE_TH_RH(MU) to set the TH and RH fields
       to default values.  (page 6.1-38).
      Call buffer manager (ADJUST_BUF_POOL, limited buffer pool ID, change amount)
       to reduce the size of the limited buffer pool  (Appendix B).
       The change amount is a negative value of 1.
      Set the RH fields as described in Figure 6.1-10 on page 6.1-15 for
       a BIS reply.
      Set the RU to BIS_RQ_RU as described in SNA Formats.
      Call DFC_SEND_FSMS(MU) (page 6.1-27).
```

```
When BIS_RQ
   Call buffer manager (GET_BUFFER, permanent buffer pool ID, no wait) to
    get a buffer for building an MU (containing the BIS).  (Appendix B)
   Create an MU and Call INITIALIZE_TH_RH(MU) to set the TH and RH fields
    to default values.  (page 6.1-38).
   Call buffer manager (ADJUST_BUF_POOL, limited buffer pool ID, change amount)
    to reduce the size of the limited buffer pool  (Appendix B).
    The change amount is a negative value of 1.
   Set the RH fields as described in Figure 6.1-9 on page 6.1-14 for
    a BIS request.
   Set the RU to BIS_RQ_RU as described in SNA Formats.
   Call DFC_SEND_FSMS(MU) (page 6.1-27).

When BRACKET_FREED
   For each MU on the PS_TO_HS_Q with a BRACKET_ID equal to BRACKET_FREED.BRACKET_ID
      Remove the MU from the PS_TO_HS queue.
      Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
       that containing the MU.  (Appendix B)

When HS_PS_CONNECTED
   Record PS_ID and BRACKET_ID (from HS_PS_CONNECTED record)
    in the corresponding LOCAL fields.
   Call FSM_BSM_FMP19 (page 6.1-50) with an INB signal to indicate that
    this half-session is connected to a PS.
   If LOCAL.SESSION_JUST_STARTED is YES then
    (current bracket sequence number was initialized at session start-up)
      Set LOCAL.SESSION_JUST_STARTED to NO.
   Else (set the current bracket sequence number and BB_REGISTER)
      The following calculates the value for the current bracket sequence number and
       the BB_REGISTER before the BB request (to be sent) is received by DFC.
      Set LOCAL.CURRENT_BRACKET_SQN.NUMBER to LOCAL.SQN_SEND_CNT.NUMBER + 1 (taking
       into account that the number wraps at 32767).
      Set LOCAL.CURRENT_BRACKET_SQN.BRACKET_STARTED_BY to the value of
       LOCAL.HALF_SESSION (PRI|SEC).
      Based on the value of LOCAL.HALF_SESSION (PRI|SEC) set
       LOCAL.PHS_BB_REGISTER.NUMBER or LOCAL.SHS_BB_REGISTER.NUMBER to
       LOCAL.CURRENT_BRACKET_SQN.NUMBER.

When RTR_RQ
   Call buffer manager (GET_BUFFER, permanent buffer pool ID, no wait) to
    get a buffer for building an MU (containing the RTR).  (Appendix B)
   Create an MU and Call INITIALIZE_TH_RH(MU) to set the TH and RH fields
    to default values.  (page 6.1-38).
   Call buffer manager (ADJUST_BUF_POOL, limited buffer pool ID, change amount)
    to reduce the size of the limited buffer pool  (Appendix B).
    The change amount is a negative value of 1.
   Set the RH fields as described in Figure 6.1-9 on page 6.1-14 for
    a RTR request.
   Set the RU to RTR_RQ_RU as described in SNA Formats.
   Set LOCAL.COMMON.RQ_CODE to RTR.
   Call DFC_SEND_FSMS(MU) (page 6.1-27).
```

When YIELD_SESSION
   If LOCAL.SESSION_JUST_STARTED is YES then (session has just started)
      Set LOCAL.SESSION_JUST_STARTED to NO (reset so current bracket SQN will be updated on future brackets)
   Call buffer manager (GET_BUFFER, permanent buffer pool ID, no wait) to get a buffer for building an MU (containing the LUSTAT). (Appendix B)
   Create an MU and Call INITIALIZE_TH_RH(MU) to set the TH and RH fields to default values. (page 6.1-38).
   Call buffer manager (ADJUST_BUF_POOL, limited buffer pool ID, change amount) to reduce the size of the limited buffer pool (Appendix B). The change amount is a negative value of 1.
   Set the RH fields as described in Figure 6.1-9 on page 6.1-14 for a LUSTAT request.
   Set the RU to LUSTAT_RQ_RU as described in SNA Formats.
   Set LOCAL.COMMON.RQ_CODE to LUSTAT.
   Call DFC_SEND_FSMS(MU) (page 6.1-27).

When ENCIPHERED_RD2
   If ENCIPHERED_RD2.SEND_PARM.TYPE is DEALLOCATE_FLUSH then
      Set LOCAL.SESSION_JUST_STARTED to NO.
   Call buffer manager (GET_BUFFER, permanent buffer pool ID, no wait) to get a buffer for building an MU (containing the ENCIPHERED_RD2). (Appendix B)
   Create an MU and set the MU.HEADER to indicate PS_TO_HS, SEND_DATA_RECORD, ENCIPHERED_RD2.SEND_PARM.ALLOCATE, ENCIPHERED_RD2.SEND_PARM.FMH, ENCIPHERED_RD2.SEND_PARM.TYPE.
   Set RU to ENCIPHERED_RD2.SEND_PARM.DATA as described in SNA Formats.
   Call SEND_FMD_MU(MU) (page 6.1-43).

TRY_TO_RCV_SIGNAL

FUNCTION:	This procedure determines if a REQUEST_TO_SEND record should be sent to PS to indicate a SIGNAL has been received. This procedure is called by the half-session router ("Chapter 6.0. Half-Session").
INPUT:	None
OUTPUT:	REQUEST_TO_SEND sent to PS if required, indication that a SIGNAL has been received (LOCAL.SIG_RECEIVED) altered if stray or current SIGNAL detected
NOTE:	LOCAL.SIG_RECEIVED is set to indicate that a SIGNAL has been received before this procedure is called.

Referenced procedures, FSMs, and data structures:

DFC_SEND_TO_PS	page 6.1-30
SIGNAL_STATUS	page 6.1-47
FSM_BSM_FMP19	page 6.1-50
REQUEST_TO_SEND	page A-10
LOCAL	page 6.0-7

If the state of FSM_BSM_FMP19 is INB and LOCAL.SIG_RECEIVED = YES then
   Call SIGNAL_STATUS (page 6.1-47) to determine the type of signal received.
   Select, based on the result returned from SIGNAL_STATUS:
      When CURRENT signal
         Call DFC_SEND_TO_PS(NULL, REQUEST_TO_SEND) (page 6.1-30).
         Set LOCAL.SIG_RECEIVED to NO.

      When STRAY signal
         Log error or informational message in the system log.
         Set LOCAL.SIG_RECEIVED to NO.

      When FUTURE signal
         Do nothing. (The signal will remain in LOCAL until the bracket in which it was sent becomes the current bracket.)

DFC_RCV

FUNCTION:	Process MUs received from TC. This procedure is called by TC ("Chapter 6.2. Transmission Control").
INPUT:	MU, containing either a request (normal or expedited) or a response; LOCAL.COMMON_RQ_CODE; indication whether the alternate code may be used, LOCAL.ALTERNATE_CODE
OUTPUT:	LOCAL.SIG_RECEIVED is set if SIGNAL is received; the SNF of the SIGNAL is saved in LOCAL.SIG_SNF; LOCAL.DIRECTION is set.

Referenced procedures, FSMs, and data structures:

DFC_RCV_FSMS	page 6.1-25
FORMAT_ERROR	page 6.1-31
SEND_RSP_MU	page 6.1-45
STRAY_RSP	page 6.1-48
TRANSLATE	page 6.1-49
LOCAL	page 6.0-7
MU	page A-29


```
Set LOCAL.DIRECTION to RECEIVE state.
If an alternate code may be used then
   Call TRANSLATE(MU) (page 6.1-49).
If LOCAL.SENSE_CODE is set to X'00000000' then
   If MU.RH.RU_CTGY is DFC then
      Set the LOCAL.COMMON.RQ_CODE to the MU request code.
      IF LOCAL.COMMON.RQ_CODE is ¬(CRV, BIS, LUSTAT, RTR, SIG) then
         Set LOCAL.COMMON.RQ_CODE to OTHER.
   Else
      Set LOCAL.COMMON.RQ_CODE to OTHER.
Call FORMAT_ERROR(MU) to perform optional format error checks (page 6.1-31).
If a format error was found in the MU then
   Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
    containing the erroneous MU  (Appendix B).
   Return to the HS router (Chapter 6.0) with LOCAL.SENSE_CODE set
    to a nonzero value.  This will cause the session to be deactivated and the
    half-session to be destroyed.
Else (no format error)
   If request then
      If MU.TH.EFI indicates normal-flow then
         Call DFC_RCV_FSMS(MU) (page 6.1-25).
      Else (expedited-flow SIGNAL request)
         Set LOCAL.SIG_RECEIVED to YES.
         Record the MU sequence number in LOCAL.SIG_SNF (used in determining the
          bracket the SIGNAL was intended for).
         Call SEND_RSP_MU(MU,EXP,POS,X'00000000') (page 6.1-45) to
          send an expedited positive response to the SIGNAL request immediately.
         Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
          containing the SIGNAL request MU.  (Appendix B)
   Else (response)
      Call STRAY_RSP(MU) to determine if the response is stray (page 6.1-48)
      If the response is not stray then
         Call DFC_RCV_FSMS(MU) (page 6.1-25).
      Else (it is a stray response)
         Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
          containing the stray response MU.  (Appendix B)
```

DFC_RCV_FSMS

FUNCTION:	Enforce data flow control protocols for received requests and responses.
INPUT:	MU, containing either a response or a normal-flow request
OUTPUT:	The request or response is sent to RM or PS. Data is recorded (from MU to LOCAL) for later use before passing the MU to RM or PS.

Referenced procedures, FSMs, and data structures:


```
Call RCV_STATE_ERROR(MU) (page 6.1-41).  These checks are optional.
If a state error is found then
   Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
    containing the erroneous MU  (Appendix B).
Else (no state error)
   Record MU header, header type, MU.DCF, MU.RH in LOCAL fields.
   If RU is present in the MU then
      Save the addressability to the MU.RU in LOCAL fields.
   Select anyorder:
      When normal-flow request
         If LOCAL.RQD_REQUIRED_ON_CEB = NO then
            Increment LOCAL.NORMAL_FLOW_RQ_COUNT by 1.
            If LOCAL.NORMAL_FLOW_RQ_COUNT exceeds 2**14 then
               set LOCAL.RQD_REQUIRED_ON_CEB to YES.
         Call CT_UPDATE(MU, LOCAL.CT_RCV) (page 6.1-29).
         If BBI = BB then
            If half-session is in send state then
               If primary half-session then
                  Set LOCAL.PHS_BB_REGISTER.NUMBER to MU.TH.SNF.NUMBER.
                  Set LOCAL.CT_SEND.SNF.BRACKET_STARTED_BY to PRI.
               Else
                  Set LOCAL.SHS_BB_REGISTER.NUMBER to MU.TH.SNF.NUMBER.
                  Set LOCAL.CT_SEND.SNF.BRACKET_STARTED_BY to SEC.
            Else (in receive state)
               If primary half-session then
                  Set LOCAL.SHS_BB_REGISTER.NUMBER to MU.TH.SNF.NUMBER.
                  Set LOCAL.CT_RCV.SNF.BRACKET_STARTED_BY to SEC.
               Else
                  Set LOCAL.PHS_BB_REGISTER.NUMBER to MU.TH.SNF.NUMBER.
                  Set LOCAL.CT_RCV.SNF.BRACKET_STARTED_BY to PRI.
         If the state of FSM_RCV_PURGE_FMP19 ≠ PURGE then
            Call GENERATE_RM_PS_INPUTS(MU) (page 6.1-36).
         Else
            Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
             containing the MU. (Appendix B)

         Call FSM_RCV_PURGE_FMP19(MU) to maintain a purging state for received
          BB chains (page 6.1-56).
         If the state of FSM_CHAIN_SEND_FMP19 = PEND_RCV_REPLY then
            Call FSM_CHAIN_SEND_FMP19(MU, NOT_SPECIFIED) (page 6.1-53).
         If BCI = BC then
            Call FSM_CHAIN_RCV_FMP19(MU, BEGIN_CHAIN) (page 6.1-51).
         If ECI = EC then
            Call FSM_CHAIN_RCV_FMP19(MU, END_CHAIN) (page 6.1-51).
         Call FSM_QRI_CHAIN_RCV_FMP19(MU) (page 6.1-55).
         Call SEND_RSP_IF_REQUIRED(MU) (page 6.1-44).
```

```
When normal-flow response
   Call CT_UPDATE(MU, LOCAL.CT_SEND) (page 6.1-29).
   Call SEND_RSP_TO_RM_OR_PS(MU) (page 6.1-46).
   Call FSM_CHAIN_SEND_FMP19(MU, NOT_SPECIFIED) (page 6.1-53).

When expedited-flow response (i.e., a positive RSP[SIG])
   Set LOCAL.SIG_RQ_OUTSTANDING to NO.
   Call SEND_RSP_TO_RM_OR_PS(MU) (page 6.1-46).
```

FUNCTION:	Maintain states by invoking the appropriate FSM while sending requests and responses.
INPUT:	MU containing request or response; alternate code allowed indicator, LOCAL.ALTERNATE_CODE
OUTPUT:	Request/response to TC; possible update of the following fields: LOCAL.DIRECTION; LOCAL.SIG_RQ_OUTSTANDING; sequence number for request or response, LOCAL.SQN_SEND_CNT

Referenced procedures, FSMs, and data structures:

SEND_MU	page 6.2-20
CT_UPDATE	page 6.1-29
TRANSLATE	page 6.1-49
FSM_CHAIN_RCV_FMP19	page 6.1-51
FSM_CHAIN_SEND_FMP19	page 6.1-53
MU	page A-29
LOCAL	page 6.0-7

```
Set LOCAL.DIRECTION to SEND.
Calculate the proper sequence number to use in the request or response and
 place the value in MU.TH.SNF.
Select anyorder:
   When normal-flow request
      If LOCAL.RQD_REQUIRED_ON_CEB = NO then
         Increment LOCAL.NORMAL_FLOW_RQ_COUNT by 1.
         If LOCAL.NORMAL_FLOW_RQ_COUNT exceeds 2**14 then
            set LOCAL.RQD_REQUIRED_ON_CEB to YES.
      Call CT_UPDATE(MU, LOCAL.CT_SEND) (page 6.1-29).
      If CEBI is CEB then
         If a definite response is required on a RQE1 request then
            Change RQE1 to RQD1.
            (This allows stray SIGNALs and responses to be accurately recognized.
         If RQD request then
            Reset LOCAL.RQD_REQUIRED_ON_CEB to No and LOCAL.NORMAL_FLOW_RQ_COUNT to 0.
      If LOCAL.FSM_CHAIN_RCV_FMP19 is in PEND_SEND_REPLY state then
         Call FSM_CHAIN_RCV_FMP19(MU, NOT_SPECIFIED) (page 6.1-51).
         (This request is an implicit response).
      If BBI is BB then
         If half-session is in send state then
            If primary half-session then
               Set LOCAL.PHS_BB_REGISTER.NUMBER to MU.TH.SNF.NUMBER.
               Set LOCAL.CT_SEND.SNF.BRACKET_STARTED_BY to PRI.
            Else
               Set LOCAL.SHS_BB_REGISTER.NUMBER to MU.TH.SNF.NUMBER.
               Set LOCAL.CT_SEND.SNF.BRACKET_STARTED_BY to SEC.
         Else (in receive state)
            If primary half-session then
               Set LOCAL.SHS_BB_REGISTER.NUMBER to MU.TH.SNF.NUMBER.
               Set LOCAL.CT_RCV.SNF.BRACKET_STARTED_BY to SEC.
            Else
               Set LOCAL.PHS_BB_REGISTER.NUMBER to MU.TH.SNF.NUMBER.
               Set LOCAL.CT_RCV.SNF.BRACKET_STARTED_BY to PRI.
      If BCI is BC then
         Call FSM_CHAIN_SEND_FMP19(MU, BEGIN_CHAIN) (page 6.1-53).
      If ECI is EC then
         Call FSM_CHAIN_SEND_FMP19(MU, END_CHAIN) (page 6.1-53).
  When normal-flow response
     Call CT_UPDATE(MU,LOCAL.CT_RCV) (page 6.1-29).
     Call FSM_CHAIN_RCV_FMP19(MU, NOT_SPECIFIED) (page 6.1-51).

  When expedited-flow request (i.e., a SIGNAL request)
     Set the LOCAL.SIG_RQ_OUTSTANDING to YES.

If an alternate code may be used then
   Call TRANSLATE(MU) (page 6.1-49).

Call SEND_MU(MU, LOCAL.COMMON_CB) (page 6.2-20).
```

BUILD_HS_TO_PS_HEADER

FUNCTION:	Fill in MU.HS_TO_PS_HEADER based on the contents of MU.RH.
INPUT:	MU containing data that needs to be passed to PS and information about the type of HS_TO_PS header that needs to be built
OUTPUT:	MU fields may be set properly to reflect the contents of MU.RH

Referenced procedures, FSMs, and data structures:
MU page A-29

```
Set MU.HEADER_TYPE to HS_TO_PS.
Set MU.HS_TO_PS.FMH to NO.

If FI = FMH then
   If RU_CTGY = FMD (FMD request) then
      Set MU.HS_TO_PS.FMH to YES.
   Else (LUSTAT request)
      Set MU.DCF to the length of MU.RH.
If ECI = EC then
   Select in any order
      When (RQE1 and CDI=CD)
         Set MU.HS_TO_PS.TYPE to PREPARE_TO_RCV_FLUSH.
      When (RQ*1 and CEBI=CEB)
         Set MU.HS_TO_PS.TYPE to DEALLOCATE_FLUSH.
      When (RQD2|3 and CDI=¬CD and CEBI=¬CEB)
         Set MU.HS_TO_PS.TYPE to CONFIRM.
      When (RQ*2|3 and CDI=CD)
         Set MU.HS_TO_PS.TYPE to PREPARE_TO_RCV_CONFIRM.
      When (RQD2|3 and CEBI=CEB)
         Set MU.HS_TO_PS.TYPE to DEALLOCATE_CONFIRM.
Else
   Set MU.HS_TO_PS.TYPE to NOT_END_OF_DATA.
```

CT_UPDATE

FUNCTION:	Record information about the last chain sent or received. This is done by updating the correlation table entry (CT).
INPUT:	MU, containing the information to save about a sent or received chain; CT may be updated, either CT_SEND or CT_RCV;
OUTPUT:	Information from the input MU added to the information that was saved from the earlier RUs of the chain, this information is saved in the input correlation table (CT_RCV or CT_SEND)
NOTE:	LOCAL.COMMON.RQ_CODE is set properly before this procedure is called.

Referenced procedures, FSMs, and data structures:

```
CT                                                        page 6.0-8
MU                                                        page A-29
```

```
If MU contains a request then
   If BCI = BC then
      Record the following information relating to the chain in the input
       correlation table (CT):
       Set the ENTRY_PRESENT to YES.
       Set SNF to MU.SNF;
       Set NEG_RSP_SENSE to X'00000000'.
       Set RU_CTGY to MU.RU_CTGY.
       Save the value of DR1I, DR2I, ERI, QRI, BBI, ¬CEB, ¬CD in the
        CT.RH fields.
       If the RU_CTGY = DFC then
          Record the MU request code in CT.RQ_CODE.
       Else
          Set CT.RQ_CODE to OTHER.
   If MU.ECI = EC then
      Save the value of DR1I, DR2I, ERI, CEBI and CDI in the CT.RH fields.
Else (MU contains a response)
   If MU.SDI = SD then
      Record that an error was found for this chain by saving the sense data from
       the response in CT.NEG_RSP_SENSE.
```

DFC_SEND_TO_PS

FUNCTION:	Send a record to PS. This procedure may locally create the record to send, depending on record type.
INPUT:	MU address (may be NULL if record type ¬= MU); record type to specify the type of record to send; the bracket ID that identifies this conversation, LOCAL.BRACKET_ID
OUTPUT:	Appropriate record or an MU is sent to PS.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
PS	page 5.0-8
MU	page A-29
LOCAL	page 6.0-7
CONFIRMED	page A-10
RECEIVE_ERROR	page A-10
REQUEST_TO_SEND	page A-10
RSP_TO_REQUEST_TO_SEND	page A-11

```
If the input record type is MU then
   Set the MU.HS_TO_PS.BRACKET_ID to LOCAL.BRACKET_ID.
Else (record type ¬= MU)
   Create the requested record type (CONFIRMED, REQUEST_TO_SEND,
    RSP_TO_REQUEST_TO_SEND, or RECEIVE_ERROR) with BRACKET_ID
    set to LOCAL.BRACKET_ID.

Send the record to the PS that is using this session.
If the PS is no longer receiving then
   If the record is an MU then
      Call buffer manager (FREE_BUFFER, buffer address) to release the
       buffer containing the MU. (Appendix B)
   Else
      Destroy the record.
```

FORMAT_ERROR

FUNCTION:	Perform format checks on all requests and responses for LU-LU session. These checks are optional. If an error is detected, the LOCAL.SENSE_CODE is set to the appropriate sense data. None, some, or all of these checks may be done.
INPUT:	MU, containing a request or response
OUTPUT:	TRUE for format error detected; otherwise, FALSE

Referenced procedures, FSMs, and data structures:

FORMAT_ERROR_RQ_FMD	page 6.1-34
FORMAT_ERROR_RQ_DFC	page 6.1-33
FORMAT_ERROR_NORM_RSP	page 6.1-32
FORMAT_ERROR_EXP_RSP	page 6.1-32
MU	page A-29
LOCAL	page 6.0-7

```
Set return code to FALSE.
Select based on one of the following conditions:
   When request
      If RU_CTGY = FMD
         Call FORMAT_ERROR_RQ_FMD(MU) (page 6.1-34).
      Else (When request with RU category of DFC)
         Call FORMAT_ERROR_RQ_DFC(MU) (page 6.1-33).

   When normal-flow response
      Call FORMAT_ERROR_NORM_RSP(MU) (page 6.1-32).

   When expedited-flow response
      Call FORMAT_ERROR_EXP_RSP(MU) (page 6.1-32).

(LOCAL.SENSE_CODE is set with the sense data indicating the type of error
 if an error is found by any of the above called procedures.)
If LOCAL.SENSE_CODE ≠ X'0000 0000' then
   Set the return code to TRUE. (Format error found.)

Pass the return code to the calling procedure.
```

FORMAT_ERROR_EXP_RSP

FUNCTION:	Perform format checks on expedited-flow responses. These checks are optional.
INPUT:	MU, containing an expedited-flow response
OUTPUT:	For an error, LOCAL.SENSE_CODE is set to the appropriate sense data.

Referenced procedures, FSMs, and data structures:

MU	page A-29
LOCAL	page 6.0-7

```
Select, in order, based on fields in the MU:
   When RU_CTGY ≠ DFC
      Set LOCAL.SENSE_CODE to X'40110000'.
   When FI = ¬FMH
      Set LOCAL.SENSE_CODE to X'400F0000'.
   When (SDI = SD and RTI = POS) or (SDI = ¬SD and RTI = NEG)
      Set LOCAL.SENSE_CODE to X'40130000'.
   When BCI = ¬BC or ECI = ¬EC
      Set LOCAL.SENSE_CODE to X'400B0000'.
   When QRI = QR
      Set LOCAL.SENSE_CODE to X'40150000'.
   When RQ_CODE ≠ SIG
      Set LOCAL.SENSE_CODE to X'40120000'.
   When RTI = NEG (-RSP to expedited request)
      Set LOCAL.SENSE_CODE to the sense data in the MU (first 4 bytes).
```

FORMAT_ERROR_NORM_RSP

FUNCTION:	Perform format checks on normal-flow responses. These checks are optional.
INPUT:	MU, containing a normal-flow response
OUTPUT:	For an error, LOCAL.SENSE_CODE is set to the appropriate sense data.

Referenced procedures, FSMs, and data structures:

MU	page A-29
LOCAL	page 6.0-7

```
Select, in order, based on fields in the MU:
   When BCI = ¬BC or ECI =  ¬EC
      Set LOCAL.SENSE_CODE to X'400B0000'.
   When (SDI =  SD and RTI = POS) or (SDI = ¬SD and RTI = NEG)
      Set LOCAL.SENSE_CODE to X'40130000'.
   When RU_CTGY = DFC and FI = ¬FMH
      Set LOCAL.SENSE_CODE to X'400F0000'.
   When RU_CTGY = FMD, RTI = POS, and FI = FMH
      Set LOCAL.SENSE_CODE to X'400F0000'.
   When RTI = NEG (negative response) and the sense data in the MU (first 4 bytes)
    is not (X'08130000', X'08140000', X'08190000', X'08460000', or X'088B0000')
      Set LOCAL.SENSE_CODE to the response sense data.
```

FORMAT_ERROR_RQ_DFC

FUNCTION:	Perform format checks for data flow control (DFC) requests. These checks are optional.
INPUT:	MU, containing DFC request
OUTPUT:	If error, LOCAL.SENSE_CODE is set to the appropriate sense data

Referenced procedures, FSMs, and data structures:

FORMAT_ERROR_RQ_FMD	page 6.1-34
MU	page A-29
LOCAL	page 6.0-7
LUSTAT_RQ_RU	SNA Formats
SIGNAL_RQ_RU	SNA Formats

```
Select, in the following order, based on one of the following conditions:
   When normal-flow and the request code is not (BIS, LUSTAT, or RTR)
      Set LOCAL.SENSE_CODE to X'10030000'.
   When expedited-flow and request code is not SIGNAL
      Set LOCAL.SENSE_CODE to X'10030000'.
   When expedited-flow and the request code is SIGNAL and
      (MU.DCF is too short for SIGNAL or
       SIGNAL_CODE in the SIGNAL_RQ_RU does not match the LU 6.2 defined format)
      Set LOCAL.SENSE_CODE to X'10050000'.
   When FI ≠ FMH
      Set LOCAL.SENSE_CODE to X'400F0000'.
   When BCI = ¬BC or ECI = ¬EC
      Set LOCAL.SENSE_CODE to X'400B0000'.
   When CSI = CODE1
      Set LOCAL.SENSE_CODE to X'40100000'.
   When EDI = ED
      Set LOCAL.SENSE_CODE to X'40160000'.
   When PDI = PD
      Set LOCAL.SENSE_CODE to X'40170000'.
   Otherwise
      If LUSTAT request then
         If MU.DCF is too long for a LUSTAT request MU and
          MU.RU contains LUSTAT_RQ_RU.STATUS_VALUE  then
            Call FORMAT_ERROR_RQ_FMD(MU) (page 6.1-34).
             (LOCAL.SENSE_CODE set by called procedure if an error is detected.)
         Else (too short for LUSTAT)
            Set LOCAL.SENSE_CODE to X'10050000'.
      Else (¬LUSTAT request)
         Select, in order, based on one of the following:
            When (BIS request with RQD2|RQD3 set) or
             (¬BIS request with ¬RQD1 set)
               Set LOCAL.SENSE_CODE to X'40140000'.
            When QRI = QR
               Set LOCAL.SENSE_CODE to X'40150000'.
            When BBI = BB or EBI = EB or CEBI = CEB
               Set LOCAL.SENSE_CODE to X'400C0000'.
            When CDI = CD
               Set LOCAL.SENSE_CODE to X'40090000'.
```

FORMAT_ERROR_RQ_FMD

FUNCTION:		Perform format checks on FM data (FMD) requests. These checks are optional.
INPUT:		MU, containing FMD request
OUTPUT:		For an error, LOCAL.SENSE_CODE is set to the appropriate sense data.
NOTES:	1.	FMH field is only 7 bits and the concatenation bit is a reserved bit (set to 0).
	2.	LOCAL.ALTERNATE_CODE is set before this procedure is called.

Referenced procedures, FSMs, and data structures:

```
         MU                                                          page A-29
         LOCAL                                                       page 6.0-7
```

```
Record FMH type (from MU.RU) for later use (see note 1).
Select, in order, based on the TH or RH settings in the MU:
   When expedited-flow
      Set LOCAL.SENSE_CODE to X'40110000'.

   When the form-of-response-requested is not RQE or RQD
      Set LOCAL.SENSE_CODE to X'40140000'.

   When the form-of-response-requested is RQD and ECI = ¬EC
      Set LOCAL.SENSE_CODE to X'40070000'.

   When BBI = BB and BCI = ¬BC
      Set LOCAL.SENSE_CODE to X'40030000'.

   When BBI = BB, RU CTGY = FMD, and ¬(FI = FMH and FM header type = ATTACH_FMH)
      Set LOCAL.SENSE_CODE to X'40030000'.

   When CSI = CODE1 and alternate code will not be used
      Set LOCAL.SENSE_CODE to X'40100000'.

   When EBI = EB (EB not allowed with FM profile 19.)
      Set LOCAL.SENSE_CODE to X'40040000'.

   When CDI = CD and ECI = ¬EC (CD allowed only with EC)
      Set LOCAL.SENSE_CODE to X'40090000'.

   When CDI = CD and form-of-response-requested is RQD1 (CD may not be sent with RQD1)
      Set LOCAL.SENSE_CODE to X'40090000'.

   When CEBI = CEB and ECI = ¬EC
      Set LOCAL.SENSE_CODE to X'40040000'.

   When BCI = BC and ((the request is received from the bidder with BBI=BB and QRI=¬QR)
    or (the request is received from the first speaker with BBI = BB or QRI = QR))
      Set LOCAL.SENSE_CODE to X'40180000'.
   When CEBI = CEB and CDI = CD (Transaction program verbs cannot generate this combination.)
      Set LOCAL.SENSE_CODE to X'40090000'.

   When CEBI = CEB and form-of-response-requested is RQE2 or RQE3
    (DEALLOCATE-CONFIRM (CEB,RQD2|3) and DEALLOCATE-FLUSH (CEB,RQE1) are valid)
      Set LOCAL.SENSE_CODE to X'40040000'.

   When CEBI = ¬CEB, CDI = ¬CD, ECI = EC, and form-of-response-requested is RQE
      Set LOCAL.SENSE_CODE to X'40190000'.

   When FI = FMH, CEBI = ¬CEB, and form-of-response-requested is RQD1
      Set LOCAL.SENSE_CODE to X'40190000'.

   When BBI = BB, CEBI = CEB, form-of-response-requested is RQE1, and this
    half-session is the first speaker (BB, CEB, RQE1 not allowed from the bidder)
      Set LOCAL.SENSE_CODE to X'40040000'.
```

```
When FI = FMH, CEBI = CEB, FM header type = ERROR_FMH, and ERI = ER
  Set LOCAL.SENSE_CODE to X'40060000'.

When FI = FMH, RU_CTGY = FMD, and FM header type is not (ATTACH_FMH or ERROR_FMH)
   If FM header type is SECURITY_FMH then
      If (ECI = EC and CEBI = ¬CEB) or BCI = BC then
         Set LOCAL.SENSE_CODE to X'080F6051'.
   Else
      Set LOCAL.SENSE_CODE to X'10084001'.
```

FUNCTION:	Generate the appropriate records for RM and PS based on the passed MU's content.
INPUT:	MU containing normal-flow request; information about the last request sent, LOCAL.CT_SEND; possibly in addition, a BID_RSP or an RTR_RSP record from RM
OUTPUT:	Appropriate records sent to RM and PS, LOCAL.CURRENT_BRACKET_SQN, ID of the PS connected to this HS

Referenced procedures, FSMs, and data structures:

DFC_SEND_TO_PS	page 6.1-30
RM	page 3-19
PROCESS_RU_DATA	page 6.1-40
OK_TO_REPLY	page 6.1-39
FSM_RCV_PURGE_FMP19	page 6.1-56
BID	page A-11
BID_RSP	page A-11
BIS_RQ	page A-12
BIS_REPLY	page A-12
RTR_RQ	page A-12
BRACKET_FREED	page A-18
RTR_RSP	page A-13
MU	page A-29
LOCAL	page 6.0-7


```
Select, in order, based on one of the following conditions:
   When BBI = BB
      Create a BID record with HS_ID set to LOCAL.HS_ID and send it to RM.
      Receive the BID_RSP from RM.
      Check the RM_TO_HS_Q to see if a BIS, RTR, or BRACKET_FREED record has
       been received from RM.  If so, send the record now (a BID race may
       have occurred).

      If a positive Bid response is received then
         If RU category is FMD then
            Call PROCESS_RU_DATA(MU) (page 6.1-40).
         Else
            Set LOCAL.BB_RSP_STATE to POS_OWED to record that a positive response is owed.
            Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
             containing the received MU  (Appendix B).
      Else (negative response to Bid)
         Call FSM_RCV_PURGE_FMP19 SIGNAL(PURGE) (page 6.1-56) to cause the
          remainder of this BB chain to be purged.
         Set LOCAL.BB_RSP_STATE to NEG_OWED to record that a negative response is owed.
         Set LOCAL.BB_RSP_SENSE to BID_RSP.SENSE_CODE to record the sense data.
         Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
          containing received MU  (Appendix B).
      Destroy the BID_RSP record from RM.

   When RU_CTGY is DFC and request code is BIS
      If the form-of-response-requested is RQE1 then
         Create a BIS_RQ record with HS_ID set to LOCAL.HS_ID and send it to RM.
      Else (RQE2|3)
         Create a BIS_REPLY record with HS_ID set to LOCAL.HS_ID and send it to RM.
      Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
       containing received MU  (Appendix B).
   When RU_CTGY is DFC and request code is RTR
      Create an RTR_RQ record with HS_ID set to LOCAL.HS_ID and send it to RM.
      Receive RTR_RSP from RM.
      If a positive RTR response is received then
         Set LOCAL.RTR_RSP_STATE to POS_OWED to record that a positive response is owed.
      Else (negative response to RTR)
         Set LOCAL.RTR_RSP_STATE to NEG_OWED to record that a negative response is owed.
         Set LOCAL.RTR_RSP_SENSE to RTR_RSP.SENSE_CODE to record the sense data.
      Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
       containing received MU  (Appendix B).
      Destroy the RTR_RSP record.
```

Otherwise
   Call OK_TO_REPLY(MU) (page 6.1-39) to determine if MU is a reply.
   If MU is a reply and the last chain sent was RQE2 or RQE3 then
      Call DFC_SEND_TO_PS(MU pointer, CONFIRMED) (page 6.1-30).

   Call PROCESS_RU_DATA(MU) (page 6.1-40).

INITIALIZE_TH_RH

FUNCTION:	Initialize the TH and RH fields of an MU record.
INPUT:	A newly created MU
OUTPUT:	Initialized RH and selected TH bits, LOCAL.COMMON.RQ_CODE

Referenced procedures, FSMs, and data structures:

MU	page A-29

Set default values for the TH fields in the MU as follows:
EFI to normal-flow, SNF.SQN to 0, BBIUI to BBIU, and EBIUI to EBIU.

Set default values for the RH fields in the MU as follows:
RRI to RQ, RU_CTGY to FMD, FI to ¬FMH, SDI to ¬SD, RTI to POS,
BCI to ¬BC, ECI to ¬EC, RQE1, RLWI to ¬RLW, QRI to ¬QR,
PI to ¬PAC, BBI to ¬BB, EBI to ¬EB, CDI to ¬CD,
CSI to CODE0, EDI to ¬ED, PDI to ¬PD, CEBI to ¬CEB

Set LOCAL.COMMON.RQ_CODE to OTHER.

INVALID_SENSE_CODE

FUNCTION:	Determine if sense data on negative response is valid.
INPUT:	MU containing negative response; information about the last chain sent, LOCAL.CT_SEND; first-speaker indicator, LOCAL.FIRST_SPEAKER
OUTPUT:	TRUE for invalid sense data; otherwise, FALSE

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
MU	page A-29
LOCAL	page 6.0-7

```
If this is a response to a BB chain then
   If this half-session is first speaker then
      If the sense data in the response is not X'08460000' or X'088B0000' then
         Return with a value of TRUE (invalid sense data).
   Else (bidder)
      If the sense data in the response is not X'08130000', X'08140000',
       or X'088B0000' then
         Return with a value of TRUE (invalid sense data).

Else (response to ¬BB chain)
   If response to RTR then
      If the sense data in the response is not X'08190000' then
         Return with a value of TRUE (invalid sense data).
   Else (not response to RTR)
      If response to BIS then (negative response to BIS not allowed)
         Return with a value of TRUE (invalid sense data).
      Else
         If the sense data in the response is not X'08460000' then
            Return with a value of TRUE (invalid sense data).

Return with a value of FALSE (valid sense data).
```

OK_TO_REPLY

FUNCTION:	Determine whether or not a request is a valid reply. A reply is a request sent (or received) after receiving (or sending) an (RQE,CD) request.
INPUT:	MU, containing a normal-flow request; LOCAL.DIRECTION; LOCAL.CURRENT_BRACKET_SQN; information about the last chain sent, LOCAL.CT_SEND
OUTPUT:	TRUE if valid reply; otherwise, FALSE

Referenced procedures, FSMs, and data structures:

LOCAL page 6.0-7
MU page A-29
FSM_CHAIN_RCV_FMP19 page 6.1-51
FSM_CHAIN_SEND_FMP19 page 6.1-53

Select, in order, based on one of the following conditions:
When the request is BIS or RTR
Return with a value of FALSE (not a valid reply).

When the request indicates BBI = BB or BCI = ¬BC
Return with a value of FALSE (not a valid reply).

When (sending and the state of FSM_CHAIN_RCV_FMP19 is not PEND_SEND_REPLY) or (receiving and the state of FSM_CHAIN_SEND_FMP19 is not PEND_RCV_REPLY) (page 6.1-51 and page 6.1-53)
Return with a value of FALSE (not a valid reply).

When receiving and the state of FSM_BSM_FMP19 (page 6.1-50) is INB and the last chain sent carried BB and LOCAL.CURRENT_BRACKET_SQN ≠ the SNF of that chain
Return with a value of FALSE (not a valid reply).

Otherwise
Return with a value of TRUE (valid reply).

PROCESS_RU_DATA

FUNCTION:	Process an RU and, based on the content of the RU, send the appropriate records to RM and PS.
INPUT:	MU containing a normal-flow request; LOCAL.SHS_BB_REGISTER; LOCAL.PHS_BB_REGISTER; LOCAL.HALF_SESSION (indication that half-session is primary or secondary); possibly in addition, an HS_PS_CONNECTED record received from RM.
OUTPUT:	Appropriate records sent to RM or PS; if an FMH-5(Attach) is present, LOCAL.CURRENT_BRACKET_SQN is set.

Referenced procedures, FSMs, and data structures:

DFC_SEND_TO_PS	page 6.1-30
FSM_BSM_FMP19	page 6.1-50
BID_RSP	page A-11
HS_PS_CONNECTED	page A-18
BUILD_HS_TO_PS_HEADER	page 6.1-28
MU	page A-29
LOCAL	page 6.0-7


```
IF FI = FMH and RU_CTGY = FMD then
   Select, based on FMH type in RU:
      When X'05' (Attach)
         Call BUILD_HS_TO_PS_HEADER(MU) (page 6.1-28).
         Set MU.HS_TO_RM.HS_ID to LOCAL.HS_ID.
         Send the request MU to RM.
         Receive the HS_PS_CONNECTED record from RM.
         Save the PS_ID and the BRACKET_ID from the HS_TO_PS_CONNECTED record
          in the corresponding LOCAL fields.
         Call FSM_BSM_FMP19 with an INB signal (page 6.1-50)
          to indicate that the HS is connected to a PS.
         Destroy the HS_PS_CONNECTED record from RM.
         If primary half-session then
            Set LOCAL.CURRENT_BRACKET_SQN to LOCAL.SHS_BB_REGISTER.
         Else
            Set LOCAL.CURRENT_BRACKET_SQN to LOCAL.PHS_BB_REGISTER.
      When X'07' (Error Description)
         Call BUILD_HS_TO_PS_HEADER(MU) (page 6.1-28).
         Call DFC_SEND_TO_PS(MU pointer, MU)  (page 6.1-30).
      When X'0C' (Security)
         Build an HS_TO_PS MU header.
         Set MU.HS_TO_RM.HS_ID to LOCAL.HS_ID.
         Send the record to RM.
Else
   If ECI = EC or MU.RU is present then
      Call BUILD_HS_TO_PS_HEADER(MU) (page 6.1-28).
      Call DFC_SEND_TO_PS(MU pointer, MU)  (page 6.1-30).
   Else
      Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
       containing the MU. (Appendix B)
```

RCV_STATE_ERROR

FUNCTION:	Perform state error checking on received requests and responses. The types of errors found here are protocol violations by the sender of the request or response. These checks are optional. None, some, or all of the checks may be made.
INPUT:	MU containing request or response; indication of whether a response to a SIGNAL is expected, LOCAL.SIG_RQ_OUTSTANDING
OUTPUT:	TRUE if a state error was encountered; otherwise, FALSE. If TRUE, LOCAL.SENSE_CODE is set to the appropriate sense data

Referenced procedures, FSMs, and data structures:

INVALID_SENSE_CODE	page 6.1-38
FSM_BSM_FMP19	page 6.1-50
FSM_QRI_CHAIN_RCV_FMP19	page 6.1-55
FSM_CHAIN_RCV_FMP19	page 6.1-51
FSM_CHAIN_SEND_FMP19	page 6.1-53
MU	page A-29
LOCAL	page 6.0-7

```
Select based on EFI and RRI:
   When normal-flow request
      Select, in order, based on the following conditions:
         When a (RQE,BB,CEB) chain is received from the bidder
            Set LOCAL.SENSE_CODE to X'40040000' ((RQE,BB,CEB) not allowed from bidder).
            Return with a value of TRUE.

         When executing FSM_BSM_FMP19(MU) (page 6.1-50),
          FSM_CHAIN_RCV_FMP19(MU) (page 6.1-51), or
          FSM_QRI_CHAIN_RCV_FMP19(MU) (page 6.1-55) would cause a state
          check (>) condition
            Execute the corresponding output code in the first FSM that encountered
             a state-check condition (to set LOCAL.SENSE_CODE).
            Return with a value of TRUE.

   When normal-flow response
      Select based on the following conditions:
         When RU category of the response ≠ RU category of the request
            Set LOCAL.SENSE_CODE to X'40110000'.
            Return with a value of TRUE.
         When RU category of the response = DFC and
          the request code of the response ≠ the request code of the request
            Set LOCAL.SENSE_CODE to X'40120000'.
            Return with a value of TRUE.
         When the QRI field of the response ≠ the QRI of the request
            Set LOCAL.SENSE_CODE to X'40210000'.
            Return with a value of TRUE.
         When response is negative and contains an invalid sense data
          (call INVALID_SENSE_CODE(MU) [page 6.1-38])
            Set LOCAL.SENSE_CODE to X'20120000'.
            Return with a value of TRUE.
         When executing FSM_CHAIN_SEND_FMP19(MU) (page 6.1-53).
          would cause a state-check (>) condition
            Execute the corresponding output code (to set LOCAL.SENSE_CODE).
            Return with a value of TRUE.

   When expedited-flow response (i.e., a positive response to SIGNAL)
      If a SIGNAL request is not outstanding (not waiting for response to SIGNAL) then
         Set LOCAL.SENSE_CODE to X'200E0000' (response correlation error).
         Return with a value of TRUE.

Return with a value of FALSE.
```

REPLY_TO_BID

FUNCTION:	Determine if a normal-flow request is a reply to a BID request. A reply is a request sent (or received) immediately after receiving (or sending) a request carrying (RQE,CD). A reply implies a positive response to the (RQE,CD) request.
INPUT:	MU containing a normal-flow request; information about the last chain sent, LOCAL.CT_SEND
OUTPUT:	TRUE if MU is reply to a bid; FALSE, otherwise

Referenced procedures, FSMs, and data structures:

OK_TO_REPLY	page 6.1-39
MU	page A-29
LOCAL	page 6.0-7
FSM_BSM_FMP19	page 6.1-50

```
Call OK_TO_REPLY(MU) (page 6.1-39).
If it is OK to reply and
 the state of FSM_BSM_FMP19 = BETB (page 6.1-50) and
 the last chain sent was a BB chain then
   Return with a value of TRUE.
Else
   Return with a value of FALSE.
```

SEND_BID_POS_RSP

FUNCTION:	Send RM a positive response to a Bid, and receive the HS_PS_CONNECTED record that will result in this half-session being connected to a PS.
INPUT:	MU; information about the last chain sent, LOCAL.CT_SEND
OUTPUT:	BID_POS_RSP sent to RM, LOCAL.CURRENT_BRACKET_SQN

Referenced procedures, FSMs, and data structures:

FSM_BSM_FMP19	page 6.1-50
MU	page A-29
HS_PS_CONNECTED	page A-18
LOCAL	page 6.0-7

```
Create a positive BID_RSP record with HS_ID set to LOCAL.HS_IS and SENSE_CODE set to 0.
Send the record to RM.

Receive the HS_PS_CONNECTED record associated with the BID_POS_RSP from RM.
Save the PS_ID and the BRACKET_ID from the HS_TO_PS_CONNECTED in the
 corresponding LOCAL fields.
Call FSM_BSM_FMP19 with an INB signal (page 6.1-50)
 to indicate that the HS is connected to a PS.
Destroy the HS_PS_CONNECTED record from RM.

Set LOCAL.CURRENT_BRACKET_SQN to the SNF of the last chain sent.
```

SEND_FMD_MU

FUNCTION:	Send an MU according to passed instructions.
INPUT:	MU, containing a PS_TO_HS record (it informs this procedure how to set the BBI, FI, BETC, BCI, ECI, ERI, DR1I, and DR2I bits in the RH)
OUTPUT:	The MU is created and initialized; MU.RH bits, LOCAL.COMMON.RQ_CODE, MU.RU are set (according to the PS_TO_HS record); and the MU is sent to PS, BETC

Referenced procedures, FSMs, and data structures:

INITIALIZE_TH_RH	page 6.1-38
DFC_SEND_FSMS	page 6.1-27
LOCAL	page 6.0-7
LUSTAT_RQ_RU	SNA Formats
MU	page A-29

```
Call INITIALIZE_TH_RH(MU) to set the TH and RH fields of the input MU
 to default values.
If input MU contains an FMH header then
   Set MU.RH.FI to FMH.
If starting a new chain (the last RU sent indicated EC) then
   Set MU.RH.BCI to BC and LOCAL.BETC to NO.
   If PS_TO_HS.ALLOCATE = YES then
      Set MU.RH.BBI to BB to indicate that this is a BB chain.
If MU.PS_TO_HS.TYPE is not FLUSH then
   Set the RH indicators as described in Figure 6.1-7 on page 6.1-13 based on the value
    of MU.PS_TO_HS.TYPE.
   Set LOCAL.BETC to YES to indicate between-chain state.
If this MU indicates (BC, EC) and there is no data in the RU then
   Convert the RU to an LUSTAT request (set RH bits to indicate FMH and DFC).
   Set RU to LUSTAT_RQ_RU (see SNA Formats).

Call DFC_SEND_FSMS(MU) (page 6.1-27).
```

SEND_RSP_IF_REQUIRED

FUNCTION:	Send a response to the passed MU if required.
INPUT:	MU containing a normal-flow request; information about the last received request; indication that a response is owed; the type (positive or negative) response to a BB request or RTR request or negative response to the next chain; when a negative response is owed, the sense data (included in the response).
OUTPUT:	Response sent if required, indication that a response is owed

Referenced procedures, FSMs, and data structures:

SEND_RSP_MU page 6.1-45
FSM_CHAIN_RCV_FMP19 page 6.1-51
MU page A-29
LOCAL page 6.0-7


```
Select in order, based on the following conditions:
   When a response is owed to a BB request
      If a positive response is owed then (it can be only an [LUSTAT,BB])
         Call SEND_RSP_MU(MU, NORMAL, POS, X'00000000') (page 6.1-45).
      Else (-RSP owed to [BB,FMD|LUSTAT])
         Call SEND_RSP_MU(MU, NORMAL, NEG, LOCAL.BB_RSP_SENSE) (page 6.1-45).

      Reset LOCAL.BB_RSP_STATE to indicate that a response is no longer owed
       to the BB request.

   When a response is owed to an RTR request
      If a positive response is owed then
         Call SEND_RSP_MU(MU, NORMAL, POS, X'00000000') (page 6.1-45).
      Else (-RSP owed to RTR)
         Call SEND_RSP_MU(MU, NORMAL, NEG, LOCAL.RTR_RSP_SENSE) (page 6.1-45).

      Reset LOCAL.RTR_RSP_STATE to indicate that a response is no longer owed
       to the RTR request.

   When a negative response is owed to the next RU received
      If MU.BCI = BC and (MU.RU_CTGY = FMD or [RU_CTGY
       = DFC and a LUSTAT request]) and the MU.BBI ≠ BB then
         Call SEND_RSP_MU(MU, NORMAL, NEG, X'08460000') (page 6.1-45).
         Reset LOCAL.SEND_ERROR_RSP_STATE to indicate that a negative response is no
          longer owed to the next RU.

   When the state of FSM_CHAIN_RCV_FMP19 = PEND_RSP (a response is owed) and
    the last chain received was CEB,RQD1
      Call SEND_RSP_MU(MU, NORMAL, POS, X'00000000') (page 6.1-45).
```

SEND_RSP_MU

FUNCTION:	Create and send a response. The response is based on the request MU (if passed by the caller) or on information about the last chain received (if a null MU is passed).
INPUT:	Request MU (if any), flow type (expedited or normal), response type (positive or negative), sense data. (Information about the last chain received is used, (LOCAL.CT_RCV), when the input request MU has a null value.)
OUTPUT:	A RSP_MU is built and sent to TC.
NOTE:	When PS sends an MU that indicates a -RSP is to be sent, the RU must be at least 4 bytes (for the sense data).

Referenced procedures, FSMs, and data structures:

DFC_SEND_FSMS	page 6.1-27
INITIALIZE_TH_RH	page 6.1-38
FSM_CHAIN_RCV_FMP19	page 6.1-51
RSP_MU, see MU	page A-29
LOCAL	page 6.0-7
MU	page A-29


```
If no MU is passed by the caller then
   Call buffer manager (GET_BUFFER, permanent buffer pool ID, no wait) to get
    a buffer for building a response MU (RSP_MU). (Appendix B)
   Create a response MU and Call INITIALIZE_TH_RH  (page 6.1-38).
Else (reuse the buffer to build a response MU)
   Call INITIALIZE_TH_RH(RSP_MU) (page 6.1-38).
Set RSP_MU.DCF to the length of RSP_MU.RH.
Set the RH fields of the RSP_MU to (RSP, BC, EC).
If a negative response need to send then
   Set RSP_MU.SDI to SD, RSP_MU.RTI to NEG.
   Add the length of RU (contains sense data) to RSP_MU.DCF.
   Copy the input sense data to the RSP.MU.
Else (positive response)
   Set RTI to POS (indicate a positive response to be sent).

If input flow type indicates normal-flow then
   If input request MU has a null value (no request MU passed by the caller) then
      Copy the RU_CTGY, DR1I, DR2I, QRI from the correlation table (CT).
      If the RH.RU_CTGY = DFC  then
         Add the length of request code to RSP_MU.DCF.
         Set the last byte of the RSP_MU.RU to the RQ_CODE from correlation table.
         Record the RQ_CODE from correlation table in LOCAL.COMMON.RQ_CODE.
   Else (a request MU was passed as input)
      Copy the RH.RU_CTGY, DR1I, DR2I, QRI from the input request MU.
      Set LOCAL.COMMON.RQ_CODE to the request CODE value from input request MU.
      If the RU category = DFC then
         Add the length of request code to RSP_MU.DCF.
         Set the last byte of the RSP_MU.RU to the RQ_CODE from correlation table.
Else (expedited, the only expedited-flow response is for SIGNAL)
   Set EFI to expedited, RH.RU_CTGY to DFC, DR1, ¬DR2, and request code to SIGNAL
    in the RSP_MU.
   Add the length of request code to RSP_MU.DCF.
   Set the last byte of the RSP_MU.RU to the RQ_CODE from correlation table.
If the RH.RU_CTGY = DFC  then
   Set FI to FMH.
Save the current value of DIRECTION.
Set the DIRECTION to SEND.
If executing FSM_CHAIN_RCV_FMP19(response MU) (page 6.1-51)
 would not cause a state-check (>) condition then
   Call DFC_SEND_FSMS(RSP_MU) (page 6.1-27) to send the response.
Else
   Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
    containing the erroneous MU  (Appendix B).
Reset DIRECTION indicator to the saved DIRECTION value.
```

SEND_RSP_TO_RM_OR_PS

FUNCTION:	This procedure builds and sends records to RM or PS based on the received response MU.
INPUT:	MU containing a response; indicator that session is first speaker; information about the last sent request
OUTPUT:	The appropriate "response" record is sent to RM or PS. LOCAL.CURRENT_BRACKET_SQN is set to the sequence number of the last sent BB request. The ID of the PS connected to this HS may be saved.

Referenced procedures, FSMs, and data structures:

DFC_SEND_TO_PS	page 6.1-30
SEND_BID_POS_RSP	page 6.1-42
FSM_BSM_FMP19	page 6.1-50
CONFIRMED	page A-10
RECEIVE_ERROR	page A-10
BID_RSP	page A-11
RTR_RSP	page A-13
RSP_TO_REQUEST_TO_SEND	page A-11
MU	page A-29
LOCAL	page 6.0-7

```
If the input MU contains an RTR response then
   Create an RTR_RSP record with HS_ID set to LOCAL.HS_ID and send it to RM.
Else
   If the input MU contains a SIG response then
      Call DFC_SEND_TO_PS(MU pointer, RSP_TO_REQUEST_TO_SEND)  (page 6.1-30).
   Else
      If the response is positive (RTI = POS) then
         If last chain sent was a BB chain and the state of FSM_BSM_FMP19
          (page 6.1-50) is BETB then
            Call SEND_BID_POS_RSP(MU)  (page 6.1-42).

         If the form-of-response-requested of the last chain sent was RQD2 or RQD3 then
            Call DFC_SEND_TO_PS(MU pointer, CONFIRMED)  (page 6.1-30).

      Else (response is negative)
         If the response sense data is X'08460000' then
            Call DFC_SEND_TO_PS(MU pointer, RECEIVE_ERROR)  (page 6.1-30).

         Else (bracket reject, i.e., X'08130000', X'08140000', or X'088B0000')
            Create a BID_RSP record (indicates negative response and contains the
             sense data from the response) with HS_ID set to LOCAL.HS_ID and send it to RM.

      Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
       containing the received MU  (Appendix B).
```

SIGNAL_STATUS

FUNCTION:	Determine if a SIGNAL is for a past, current, or future bracket. The in-bracket (INB) state exists when this procedure is called.
INPUT:	LOCAL.SIG_SNF, LOCAL.CURRENT_BRACKET_SQN, LOCAL.PHS_BB_REGISTER, LOCAL.SHS_BB_REGISTER
OUTPUT:	Either CURRENT, FUTURE, or STRAY return code is set

Referenced procedures, FSMs, and data structures:
LOCAL page 6.0-7

```
If the sequence number of the last SIGNAL received ≠ LOCAL.CURRENT_BRACKET_SQN then
   Select based on the high-order bit of the SIGNAL SNF (indicates
    whether the primary or secondary half-session started the bracket):
      When PRI
         Use LOCAL.PHS_BB_REGISTER.NUMBER for the following calculation.

      When SEC
         Use LOCAL.SHS_BB_REGISTER.NUMBER for the following calculation.

   Calculate (SIG_SNF - [PHS|SHS]_BB_REGISTER.NUMBER) modulo 2**15.
   If result is < 0 then
      Set the result to the result plus 2**15 (full_wrap) to determine the
       wrap condition.
      Select based on result of above calculation:
         When result = 0
            Return STRAY signal.
         When result ≤ 2**14 (half_wrap)
            Return FUTURE signal.
         When result > 2**14 (half_wrap)
            Return STRAY signal.

Else (sequence number of the last SIGNAL received = LOCAL.CURRENT_BRACKET_SQN)
   Return CURRENT signal.
```

STRAY_RSP

FUNCTION:	Determines if a response is stray. (A stray response is one that was sent in a bracket (conversation) but received in a different (later) bracket.) Logs responses.
INPUT:	MU containing a response, information about the last request sent, LOCAL.CURRENT_BRACKET_SQN, LOCAL.COMMON.RQ_CODE
OUTPUT:	TRUE if stray response; otherwise, FALSE. If stray response represents a response correlation error, LOCAL.SENSE_CODE is set and a stray-response message is logged.
NOTE:	An outstanding request is a request that has not been responded or replied to.

Referenced procedures, FSMs, and data structures:

FSM_BSM_FMP19	page 6.1-50
LOCAL	page 6.0-7
MU	page A-29

```
If the response is RTR, there is an outstanding request chain, and the response SNF ≠ the
   sequence number of the outstanding (awaiting a response) request then
      Set LOCAL.SENSE_CODE to X'200E0000' (response correlation error).
      Indicate that the response is stray.

If the response is SIGNAL and its SNF ≠ LOCAL.CURRENT_BRACKET_SQN then
   Indicate that the response is stray.

If the response is LUSTAT or the RU category is FMD then
   If there is an outstanding request chain then
      If the outstanding chain carried BB and the BB SNF does not match
        that in the response then
         Indicate that the response is stray.
      Else
         If the response SNF ≠ LOCAL.CURRENT_BRACKET_SQN or the state of
          FSM_BSM_FMP19 (page 6.1-50) is BETB then
            Indicate that the response is stray.
   Else (no outstanding request chain)
      Indicate that the response is stray.

If the response is stray then
   If the response is positive (RTI = POS) and it is not a SIGNAL
    (no positive response other than SIGNAL can be stray) then
      Set LOCAL.SENSE_CODE to X'200E0000' (response correlation error).
   Else
      Optionally log the stray response to the system log.

   Return with a value of TRUE (stray response).
Else
   Return with a value of FALSE (not a stray response).
```

TRANSLATE

FUNCTION:	Translate FMD requests, if necessary, to and from the alternate code (e.g., ASCII). When receiving, translation is from the alternate code to EBCDIC. When sending, translation is from EBCDIC to the alternate code.
INPUT:	MU untranslated, LOCAL.DIRECTION
OUTPUT:	MU, translated if necessary and, if data is to be sent, MU.RH.CSI set to CODE1

Referenced procedures, FSMs, and data structures:

MU	page A-29
LOCAL	page 6.0-7

```
If MU is an FMD request containing RU data other than sense data then
   Select based on LOCAL.DIRECTION:
      When SEND
         If data should be sent as the alternate code (The way the decision
          is made is not architecturally defined) then
            Translate the MU RU data from EBCDIC to the alternate code.
             Translation details are not formally defined.
            Set MU.CSI to CODE1.

      When RECEIVE
         If MU.CSI = CODE1 then (RU is encoded as the alternate code)
            Translate the MU RU data from the aternate code to EBCDIC.
             Translation details are not formally defined.
```

These are the FSM input definitions used for all the FSMs in this chapter:

- R or S: MU that is being processed is being received or sent, respectively.
- RQ, RSP, BC, EC, CD, CEB, FMD, QR: Refer to the RH of the MU.
- BEGIN_CHAIN or END_CHAIN: Refer to values of CHAIN_INDICATOR. CHAIN_INDICATOR does not have to be specified. In that case, it is neither BEGIN_CHAIN nor END_CHAIN.
- RQD: RH set to RQD1, RQD2, or RQD3.
- RQE: RH set to RQE1, RQE2, or RQE3.
- REPLY: A call to OK_TO_REPLY(MU) (page 6.1-39) returns TRUE.
- BIS: MU contains a BIS RU.
- RTR: MU cotains an RTR RU.
- FMH5: MU contains an FMH5.
- FMH12: MU contains an FMH12.
- LUSTAT: MU contains an LUSTAT request or response.
- NOT_BID_REPLY: RH set to (BC, ¬BB) and either the last sent chain did not carry BB or a call to OK_TO_REPLY (page 6.1-39) returns a value of FALSE.
- CEB_UNCOND: RH set to (CEB, RQ*1).

FSM_BSM_FMP19

FUNCTION:	Enforce the bracket protocol. State transitions are forced via the input signals INB (go in brackets) and BETB (go between bracket). The inputs R, RQ,... are used for error checking only. INB state means DFC (the half-session) is connected to a PS; BETB state means DFC is not connected to a PS.
INPUT:	MU or a signal that the FSM should be set to the specified state
OUTPUT:	If an error is discovered, LOCAL.SENSE_CODE is set.
NOTE:	The state names mean the following: • BETB: between brackets • INB: in bracket

Referenced procedures, FSMs, and data structures:

MU	page A-29
LOCAL	page 6.0-7

INPUTS STATE NAMES----> STATE NUMBERS-->	BETB 01	INB 02
SIGNAL(INB)	2	-
SIGNAL(BETB)	-	1
R,RQ,(FMD|LUSTAT),NOT_BID_REPLY,¬FMH5,¬FMH12,¬CEB_UNCOND	>(R)	-

OUTPUT CODE	FUNCTION
R	Set LOCAL.SENSE_CODE to X'20030000' (bracket error).

FSM_CHAIN_RCV_FMP19

FUNCTION: Enforce the chaining protocol for received chains. A chain is "complete" when the end-of-chain (EC) request has been received and any required associated response or reply has been sent. A reply is a request sent after receiving an (RQE,CD) chain that has not been negatively responded to. A reply implies a positive response to the (RQE,CD) chain.

INPUT: MU, CHAIN_INDICATOR (possible values are BEGIN_CHAIN, END_CHAIN and NOT_SPECIFIED), information about the last received request

OUTPUT: If the bracket was ended by the request, the HS will be disconnected from PS; information recorded about the last received request may be erased; LOCAL.SENSE_CODE may be set.

NOTE: The state names mean the following:

- BETC: between chains
- INC: in the middle of a chain
- NEG RSP SENT: in the middle of a chain and a negative response has been sent
- PEND RSP: has received (EC,RQD) and is waiting for the response to be sent
- PEND SEND REPLY: has received (EC,RQE,CD) and is waiting for the reply or negative response to be sent

Referenced procedures, FSMs, and data structures:

OK_TO_REPLY	page 6.1-39
FSM_BSM_FMP19	page 6.1-50
MU	page A-29
FREE_SESSION	page A-12
LOCAL	page 6.0-7

INPUTS / STATE NAMES----> / STATE NUMBERS-->	BETC 01	INC 02	NEG RSP SENT 03	PEND RSP 04	PEND SEND REPLY 05
R,RQ,BEGIN_CHAIN	2	/	/	/	/
R,RQ,END_CHAIN,RQD	/	4	1(A)	/	/
R,RQ,END_CHAIN,RQE,CEB	/	1(A)	1(A)	/	/
R,RQ,END_CHAIN,RQE,CD	/	5	1(B)	/	/
R,RQ,END_CHAIN,BIS	/	1	/	/	/
S,-RSP,(FMD|LUSTAT)	>	3	>	1(A)	1(A)
S,+RSP,(FMD|LUSTAT)	>	/	>	1(A)	/
S,±RSP,RTR	/	/	/	1	/
S,RQ,REPLY	/	/	/	/	1
R,RQ,BC	-	>(R1)	>(R1)	>(R2)	>(R3)
R,RQ,¬BC	>(R1)	-	-	>(R2)	>(R1)
SIGNAL(RESET)	-	1	1	1	1

OUTPUT CODE	FUNCTION

A	If the last chain received did not carry BB or (it carried BB and it was accepted, i.e., there was no negative response to the BB chain with sense data X'08130000', X'08140000', or X'088B0000') then
	If the bracket has ended (the last received chain carried CEB and either [1] the form-of-response-requested was RQE or RQD1, or [2] no negative response was sent to the chain) then
	Create and send a FREE_SESSION record to RM.
	Call buffer manager (ADJUST_BUF_POOL, dynamic buffer pool ID, REPLENISHED pool size) to keep the same pool size (see Appendix B).
	Call FSM_BSM_FMP19 with a BETB signal (page 6.1-50).
	Reset LOCAL.BRACKET_ID to a null value.
	Reset correlation table CT_RCV entry to NO.

B	Reset correlation table CT_RCV entry to NO.
R1	Set LOCAL.SENSE_CODE to X'20020000' (chaining error).
R2	Set LOCAL.SENSE_CODE to X'200A0000' (immediate request mode error).
R3	Set LOCAL.SENSE_CODE to X'20040000' (half-duplex error) .

FSM_CHAIN_SEND_FMP19

FUNCTION: Enforce the chaining protocol for sending chains. A chain is "complete" when the end-of-chain (EC) request has been sent and any required associated response or reply has been received. A reply is a request received after sending an (RQE,CD) chain that has not received a negative response. A reply implies a positive response to the (RQE,CD) chain.

INPUT: MU, CHAIN_INDICATOR (possible values are BEGIN_CHAIN, END_CHAIN and NOT_SPECIFIED), information about the last received request.

OUTPUT: If the bracket was ended by the request, the HS will be disconnected from PS; information recorded about the last received request may be erased; LOCAL.SENSE_CODE may be set.

NOTE: The state names mean the following:

- BETC: between chains
- INC: in the middle of a chain
- NEG RSP RCVD: in the middle of a chain and a negative response has been received
- PEND RSP: has sent (EC,RQD) and is waiting for the response to be received
- PEND RCV REPLY: has sent (EC,RQE,CD) and is waiting for the reply or negative response to be received

Referenced procedures, FSMs, and data structures:

OK_TO_REPLY	page 6.1-39
FSM_BSM_FMP19	page 6.1-50
FREE_SESSION	page A-12
MU	page A-29
LOCAL	page 6.0-7

INPUTS — STATE NAMES----> / STATE NUMBERS-->	BETC 01	INC 02	NEG RSP RCVD 03	PEND RSP 04	PEND RCV REPLY 05
S,RQ,BEGIN_CHAIN	2	/	/	/	/
S,RQ,END_CHAIN,RQD	/	4	1(A)	/	/
S,RQ,END_CHAIN,RQE,CEB	/	1(A)	1(A)	/	/
S,RQ,END_CHAIN,RQE,CD	/	5	1(B)	/	/
S,RQ,END_CHAIN,BIS	/	1(B)	/	/	/
R,-RSP,(FMD|LUSTAT)	>(R)	3	>(R)	1(A)	1(A)
R,+RSP,(FMD|LUSTAT)	>(R)	>(R)	>(R)	1(A)	>(R)
R,±RSP,RTR	>(R)	>(R)	>(R)	1(B)	>(R)
R,RQ,REPLY	/	/	/	/	1(B)
SIGNAL(RESET)	-	1	1	1	1

OUTPUT CODE	FUNCTION

A	If the last chain sent did not carry BB or
	(it carried BB and it was accepted, i.e., there was no negative response to the BB chain with sense data X'08130000', X'08140000', or X'088B0000') then
	  If the bracket has ended (the last sent chain carried CEB and either [1] the form-of-response-requested was RQE or RQD1, or [2] no negative response was received for the chain) then
	    Create and send a FREE_SESSION record to RM.
	    Call FSM_BSM_FMP19 with a BETB signal (page 6.1-50).
	    Set LOCAL.BRACKET_ID to NULL.
	Set correlation entry to indicate no request chain outstanding.

B	Set correlation entry to indicate no request chain outstanding.
R	Set LOCAL.SENSE_CODE to X'200F0000' (response protocol error).

FSM_QRI_CHAIN_RCV_FMP19

FUNCTION:	Enforce the setting of the QRI indicator in the RH. This indicator is set the same for all MUs in a chain; i.e., all MUs in a chain have QRI=QR or have QRI=¬QR.
INPUT:	MU and information about the last received request
OUTPUT:	If a QRI state error is detected, LOCAL.SENSE_CODE is set.
NOTE:	1) The state names mean the following: • RESET: no chain is currently being received. • INC QR: the chain that is being received is a QR chain. • INC NOT QR: the chain that is being received is not a QR chain. 2) The implementation of this FSM is optional because it is used only to detect receive error conditions.

Referenced procedures, FSMs, and data structures:

MU	page A-29
LOCAL	page 6.0-7

STATE NAMES----> INPUTS STATE NUMBERS-->	RESET 01	INC QR 02	INC NOT QR 03
R,RQ, QR, EC	-	1	>(R)
R,RQ, QR,¬EC	2	-	>(R)
R,RQ,¬QR, EC	-	>(R)	1
R,RQ,¬QR,¬EC	3	>(R)	-
SIGNAL(RESET)	-	1	1

OUTPUT CODE	FUNCTION
R	Set LOCAL.SENSE_CODE to X'200B0000' (QRI state error).

FSM_RCV_PURGE_FMP19

FUNCTION:	Maintain a purging state for received BB chains that have been negatively responded to indicating a bracket error (0813, 0814, 088B). It is called with a PURGE signal when the negative response is sent and reset when the end-of-chain (EC) RU is received. When in the purging state, no records are generated for PS or RM as a result of receiving a request RU in the BB chain (i.e., the remainder of the BB chain is purged).
INPUT:	MU and information about the last received request
OUTPUT:	None

Referenced procedures, FSMs, and data structures:

MU	page A-29

INPUTS STATE NAMES----> / STATE NUMBERS-->	RESET 01	PURGE 02
R, EC	-	1
SIGNAL(PURGE)	2	-
SIGNAL(RESET)	-	1

CHAPTER 6.2. TRANSMISSION CONTROL

INTRODUCTION

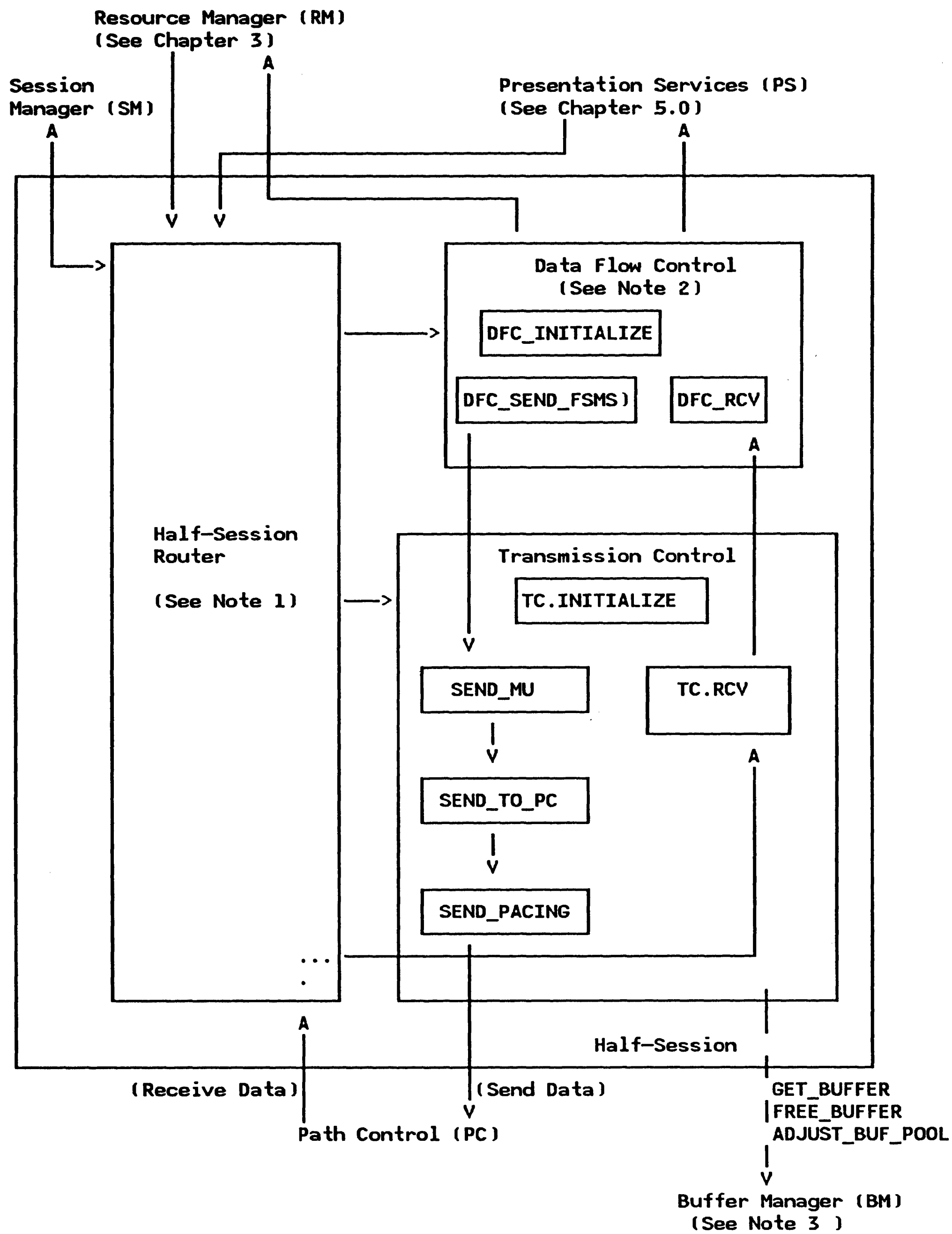
The basic function of the transmission control (TC) component is to control the flow of data between the half-session and path control. Transmission control participates in two activities:

- Initialization:
  - Variable initialization
  - Cryptography initialization
- Normal operation:
  - Sending data from data flow control (DFC) to path control (PC)
  - Receiving data from PC and sending it to DFC

TC.INITIALIZE (page 6.2-13), the procedure for session initialization, is invoked after the LU session manager (SM) processes a +RSP(BIND). TC.INITIALIZE provides session-specific support for starting data flows in the session. When session-level cryptography is used, TC.INITIALIZE checks that the enciphering and deciphering functions are operative before any user data is permitted to flow.

The SEND_MU and TC.RCV procedures manage the expedited and normal flows, and control sequence-number updating, receive-checking, session-level pacing, and data enciphering and deciphering.

The relationship of transmission control to the other elements of the half-session, after initialization, is shown in Figure 6.2-1 on page 6.2-2.



Notes:

1. See "Chapter 6.0. Half-Session" for details.
2. See "Chapter 6.1. Data Flow Control" for details.
3. HS Router and DFC also interact with the BM, but this is not shown in the figure. See HS, DFC chapters for details.

Figure 6.2-1. Structure of TC and Flow of Data within the Half-Session

INITIALIZATION PHASE

TC.INITIALIZE (page 6.2-13) is called by HS_INITIALIZATION ("Chapter 6.0. Half-Session") during initialization when a half-session is being activated. TC.INITIALIZE sets up session-level pacing and cryptography verification variables.

For sessions that support cryptography, the initialization procedure calls TC.EXCHANGE_CRV (page 6.2-15) to perform the message-unit exchanges necessary to enable data enciphering and deciphering.

CRYPTOGRAPHY VERIFICATION (CRV)

Flow: From primary LU to secondary LU (Expedited)

When session-level cryptography is specified in the BIND, CRV is sent by the primary LU TC to the secondary LU TC to enable sending and receiving of enciphered FMD requests by both half-sessions. CRV is a valid request only when session-level cryptography is selected in BIND. CRV carries an 8-byte field (see SNA Formats) that contains a transform of the deciphered test value (enciphered under the session cryptography key). The test value is received by the primary LU in the +RSP(BIND); the transform in CRV is the test value with each bit of its first four bytes inverted (i.e., a 1 becomes a 0 and a 0 becomes a 1). (The test value is also used as the session seed, or initial chaining value, when enciphering and deciphering FMD RUs while the session is active.) The secondary TC element obtains the returned test value by deciphering the aforementioned 8-byte field in CRV and inverting the first four bytes; it then compares it with the test value sent (enciphered) in +RSP(BIND). If the values donot match the session cryptography key and the session seed, the session is deactivated.

Valid cryptography options are defined under the BIND format in SNA Formats, which also describes the RH bits used for cryptography.

Where session cryptography is used, session key distribution is managed by the CP of the primary LU; session keys are conveyed (enciphered under LU master cryptography keys) to the PLU in a CINIT RU and then to the secondary LU in a BIND request (see SNA Formats and Figure 6.2-2 on page 6.2-4). The flows involved in distributing the session seed to the LU are shown in Figure 6.2-2 on page 6.2-4.

```
 CP                       PRIMARY LU             SECONDARY LU
----                      ----------             ------------

CINIT (MKp [SK] 0, MKs [SK] 0)                              [1]
----------------------------->
                             BIND (MKs [SK] 0)              [2]
                             ------------------------------>
                             RSP(BIND, SK [SS] 0)           [3]
                             <------------------------------
                             CRV(SK [transformed SS] 0)     [4]
                             ------------------------------>
                             RSP(CRV)                       [5]
                             <------------------------------
                             FMD request(SK [RU data] SS)   [6]
                             ------------------------------>
                                        .
                                        .
                                        .
                             FMD request(SK [RU data] SS)   [6]
                             <------------------------------

LEGEND:

    MKp     master cryptography key for primary LU (obtained from
                installation and implementation dependent system definition).
    MKs     master cryptography key for secondary LU (obtained from
                installation and implementation dependent system definition).
    SK      session cryptography key
    SS      session seed

NOTE: Enciphered data is represented in the diagram as follows:

          cryptography key [ data ] initial chaining value

For example, to show an RU that was enciphered using the session key
as the cryptography key and 0 as the initial chaining value,
the following string is used:

          SK [RU data] 0.
```

Figure 6.2-2. Distributing the Session Cryptography Key and Session Seed to the LU

The comments below correspond to the numbers in Figure 6.2-2.

1. In the CINIT RU, the session cryptography key is distributed to the primary LU in two enciphered formats: it is enciphered using the master cryptography key of the primary LU and in another field it is enciphered using the master cryptography key of the secondary LU. The initial chaining value is 0 for both cases.

2. In the BIND RU, the primary LU sends the session cryptography key to the secondary LU as it was received in the CINIT RU: enciphered using the master cryptography key of the secondary LU as the cryptography key and 0 as the initial chaining value.

3. The secondary LU deciphers the session cryptography key using its master cryptography key as the cryptography key and 0 as the initial chaining value. The secondary LU then generates a pseudo-random value, retains it for use as the session seed, and enciphers it using the session cryptography key as the cryptography key and 0 as the initial chaining value. This enciphered value is returned on the response to BIND. The value serves two purposes: it is used as a test value (i.e., when returned in CRV discussed below), and is subsequently used as the session seed, or initial chaining value, in enciphering and deciphering FMD requests within the session.

4. The primary LU deciphers the test value received in the RSP(BIND) using the session cryptography key as the deciphering key and 0 as the initial chaining value. The resulting value is retained for use as the session seed and then transformed by exclusive-ORing it with X'FFFFFFFF00000000'. This inverts the bit settings in the first four bytes. The transformed value is then enciphered using the session cryptography key as the key and 0 as the initial chaining value. This transformed, enciphered value is sent on the CRV request.

5. The secondary LU deciphers the enciphered, transformed test value using the session cryptography key as the key and 0 as the initial chaining value. The result is then exclusive-ORed with X'FFFFFFFF00000000' to recreate the original pseudo-random value sent by the secondary LU in RSP(BIND). The recreated value is compared with the actual value that was created by the secondary LU. If the recreated value matches the original value, a positive response is sent to CRV. The test value can then be used as the session seed.

6. From then on, all FMD requests are enciphered using the session cryptography key as the key and the session seed as the initial chaining value.

Cryptography verification is the only session control (SC) request handled by TC. SC requests for session activation and deactivation (for example, BIND and UNBIND) are routed from PC to SM (see "Chapter 4. LU Session Manager") without passing through TC. Session control requests and responses have the header bit-settings described below.

All session control requests are issued by TC or by SM. The following fields of the TH and RH are set for session control RUs.

<u>TH:</u> All session control requests and responses are sent expedited (the EFI bit is <u>on</u> in the TH).

<u>RH:</u> The RH settings for session control requests are defined in TC.BUILD_CRV on page 6.2-17 .

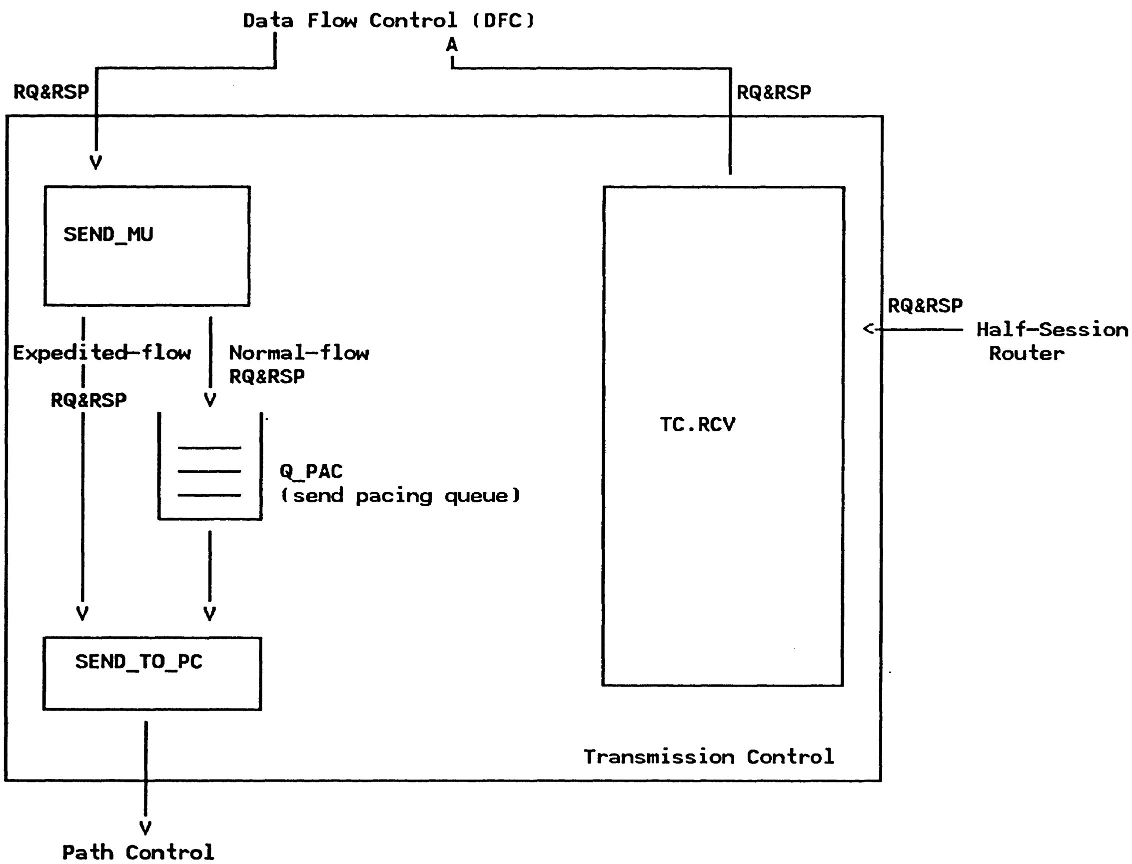


Figure 6.2-3. SEND_MU and TC.RCV Request/Response Flow

The request and response flow with SEND_MU and TC.RCV protocol machines are shown in Figure 6.2-3. Detailed definitions for SEND_MU and TC.RCV, the major TC procedures, are shown on page 6.2-20 and page 6.2-23 , respectively.

The protocols supported by TC include:

- Checking of sequence numbers on received normal-flow requests (Sequence numbers are assigned to normal-flow requests by DFC, see "Chapter 6.1. Data Flow Control")
- Proper separation of the normal flows from the expedited flows with respect to sequencing and pacing
- Sending of normal-flow requests using pacing, which involves a queue (LOCAL.Q_PAC) for temporarily holding outgoing requests
- Proper routing of requests and responses to PC and DFC
- Enciphering and deciphering control for all LU-LU FMD request RUs on sessions using session-level mandatory cryptography (see TC.DECIPHER_RU [page 6.2-32])

TC PROCEDURES INVOKED FROM OTHER COMPONENTS OF THE HALF-SESSION

Procedure TC.RCV (page 6.2-23) is invoked by the half-session router (see "Chapter 6.0. Half-Session" for details).

When the half-session router receives a message unit (MU) from path control, it calls TC.RCV to initiate TC processing of the message unit.

SEND_MU (page 6.2-20) is called by DFC when DFC has a full buffer to send or when DFC is flushing a partially filled buffer. The buffer is considered full when it reaches the maximum RU size as specified in BIND.

SEQUENCE NUMBERING OF REQUESTS AND RESPONSES

For TS profile 7 (see SNA Formats), each request that is sent on the normal flow is assigned a sequence number. The sequence number is initialized to 0 when a half-session is activated (BIND is sent or received); it is incremented by 1 before sending each request. Thus, the sequence number for the first request is 1. After reaching 65,535, the sequence number wraps to 0. (A sequence number of 0 is sent in the wrap situation only.) Sequence numbers are assigned in the sending half-session by DFC and are checked in the receiving half-session by TC.

For the expedited flow, an identifier is assigned to each request sent. The identifier is not necessarily managed as a sequence number, but is used to uniquely identify each outstanding expedited-flow request sent. The expedited-flow DFC request SIGNAL is assigned an identifier by DFC; the expedited-flow SC request CRV is assigned an identifier by TC; expedited-flow session-activation (BIND) and session-deactivation (UNBIND) requests are assigned identifiers by SM (see "Chapter 4. LU Session Manager").

The sequence number or the identifier, as appropriate, is given to path control with the associated BIU, to be carried in the TH.

The sequence number or identifier generated by the sending component is retained for use in correlating responses to requests (a response carries the sequence number or identifier of the corresponding request).

See "Sequence Numbering of Requests and Responses" in "Chapter 6.1. Data Flow Control". for further information on sequence numbering.

SESSIONS WITH CRYPTOGRAPHY

If session-level mandatory cryptography is selected when the session is activated, TC enciphers all FMD request RUs being sent and deciphers all those being received. The process of enciphering involves the following actions:

- The RU is padded, when necessary, to an integral multiple of eight bytes. The padding bytes are added at the end and contain unpredictable values, except for the last pad byte, which contains an unsigned 8-bit binary count of the pad bytes. If only one byte of pad is required, that byte is the pad byte and it contains a 1. If padding is performed, the padded data indicator (PDI) in the RH is set to PD. (The LU control operator checks that the system defined maximum RU size is a multiple of 8 during initialization time. See "Chapter 5.4. Presentation Services--Control-Operator Verbs" for details.)
- Prior to enciphering, the first eight bytes of an RU are exclusive-ORed with the session seed (i.e., the initial chaining value); the result is then enciphered. Each subsequent 8-byte block within the same RU is exclusive-ORed with the output of the previously enciphered block. This technique is referred to as "block chaining with cipher text feedback." When an enciphered RU is sent,

the Enciphered Data indicator (EDI) in the RH is set to ED.

- Enciphering employs an 8-byte block chain algorithm and an 8-byte key, the session cryptography key, and is in accordance with the Data Encryption Standard (DES) algorithm described in Federal Information Processing Standards Publication 46, dated January 15, 1977.

The deciphering process is simply the inverse of enciphering.

REQUEST AND RESPONSE CONTROL MODES

TC enforces the immediate request mode during cryptography verification (CRV) exchange as part of TC initialization. The last thing that the primary TC does during initialization is to send a CRV request and receive the CRV response. The last thing that the secondary TC does during initialization is to receive the CRV request and send the CRV response. TC accepts no other records from HS components, and nothing from path control (PC) except CRV, during this time. (See "Request/Response Mode Protocols" in Chapter 6.1 for details.)

TC is not involved in enforcing immediate request mode at any other time, and it is not involved in enforcing immediate response mode at any time.

BUFFER MANAGEMENT

On sending a normal-flow request, path control (or data link control, if segment generation is not supported) frees the limited buffer which the normal-flow request was in and sends the request to the partner LU. On receiving a normal-flow request, data link control stores the request in a link buffer. The data remains in the link buffer until TC receives it. TC frees the link buffer after moving the data to a fixed/varying dynamic buffer. (Dynamic buffers are used to receive normal-flow requests.)

TC calls the buffer manager (BM) to obtain buffer management services and specifies the following signals: (see Appendix B for more details)

- TC specifies FREE_BUFFER in the Call to the buffer manager to release a link buffer in which session data (i.e. MU, IPM) was received.

- TC specifies ADJUST_BUF_POOL in the Call to allow the BM to decrease the the number of limited buffers in a limited buffer pool when an unsolicited IPM (i.e. next-window size = 0) is received.

- TC specifies ADJUST_BUF_POOL in the Call to allow the BM to increase the number of limited buffers in a limited buffer pool when a solicited IPM (i.e. next-window size > 0) is received.

- TC specifies GET_BUFFER in the Call to get a permanent buffer to send IPM acknowledgment to its partner.

- TC specifies GET_BUFFER in the Call to get a fixed/varying dynamic buffer to store a received normal-flow request.

SESSION-LEVEL PACING

Session-level pacing allows TC to control the rate at which it receives requests on the normal flow. If pacing is selected when the session is activated, all normal-flow requests are paced. Send pacing controls the outbound flow of data. Receive pacing controls the inbound flow of data. The SEND_MU procedure (page 6.2-20) performs send pacing requests and has a session partner TC.RCV procedure (page 6.2-23) that is doing receive pacing requests. Requests and responses on the expedited flow are not paced and are unaffected by pacing on the normal flow. Pacing is generally used when the sending TC is capable of sending requests faster than the receiving TC can process them. A normal-flow response with the QR bit (in RH) set is also paced.

The pacing environment assumes that the receiving TC is able to accept no more than a certain number of requests at a time. This number, called the window size, is defined initially when the session is being activated.

For fixed pacing, the window size remains constant; for adaptive pacing, it varies from window to window, as explained later. Pacing operates according to the following cycle. At the start of a session, the sending TC may send a window whose size is 1 for adaptive pacing, or whose size is set in BIND or RSP(BIND) for fixed pacing (depending on whether nonnegotiable or negotiable BIND, respectively, is used). On the first request of any window, the sending TC turns on the Pacing Request indicator (PI). After the receiving TC receives the request that contains the Pacing Request indication, it can signal the sending TC (by using the Pacing Response indication) when it is ready to receive another group of requests.

The sending TC keeps a count of the residual number of requests that it can send before receiving a pacing response. (This value and all others related to session-level pacing and the maximum RU size are maintained in the transmission-control control block [LOCAL.TC.COMMON_CB] which is a substructure of the page 6.0-7.) Assume the current window size is N. When a pacing response is received, the sending TC is informed by the BM that additional window is available and therefore increases the pacing count by N for fixed pacing, or the value N' indicated by the pacing response (i.e. solicited IPM) for adaptive pacing. This makes the pacing count equal to the new window size (N or N') plus the residual pacing count (the remaining requests not yet sent from the previous window). If the pacing count drops to 0, the sender waits until a pacing response is received before sending any more requests. The value of the pacing count can range from 0 to 2N-1 (for fixed pacing) or N-1+N' (for adaptive pacing).

The pacing response may be returned as follows: for fixed pacing, on a normal-flow response header or on an ISOLATED PACING RESPONSE (IPR); for adaptive pacing, on an ISOLATED PACING MESSAGE (IPM), either solicited or unsolicited.

1. For fixed pacing, only one IPR is generated for each pacing request. the IPR may be used at any time when fixed pacing is supported; however, it is especially useful when no other response to a request is available in which to send the pacing response or when the available response is blocked on the pacing queue. IPR can be sent on the normal or expedited flow.

2. For adaptive pacing, a solicited or unsolicited IPM is returned for the pacing request. If it is necessary, an unsolicited IPM may be used even without receiving a pacing request previously. A reset acknowledgment IPM is generated as a response for each unsolicited IPM. The next unsolicited IPM can be used only when the reset acknowledgment IPM is returned for the current unsolicited IPM. If an unsolicited IPM carries a next window size (NWS) of 0, the sending TC uses a solicited IPM after receiving a IPM acknowledgement in order to allow paced normal-flow requests to resume flowing. An IPM is sent on the expedited-flow.

The decision on whether there are sufficient resources for sending a pacing response is implementation-dependent.

Normal-flow responses that have the Queued Response indicator (QRI) set to QR are placed on the pacing queue (Q_PAC), but do not cause the pacing count to be decremented when they are sent. When normal-flow responses indicate ¬QR, they can pass requests and responses marked QR at the queuing point in TC. If a request is held up by pacing, all responses marked QR and queued behind the request are also held up.

A Pacing Response indication is never added to a response held in Q_PAC; it is added only to a response(i.e., a "piggy backed" pacing response) with QRI=QR as it is dequeued from Q_PAC or to a response with QRI=¬QR when it is sent. An IPR can be generated and sent directly to PC to prevent session deadlock, which could occur when both TC pacing queues contain a request that cannot flow and that blocks the flow of the only available responses that might be used to carry the Pacing Response indication.

Although the no-pacing option exists, only T5 node LUs support receipt of unpaced data.

When a T2.1 node LU sends a BIND, it sets the staging indicators to specify one-stage in both directions, and sets the pacing window sizes to values determined by installation-dependent considerations. When a subarea LU sends an SSCP-mediated BIND, the values for the staging indicators and pacing windows are contained in the BIND image sent to the LU in CINIT, which the PLU may or may not place in the BIND RU. For the format and meanings of the pacing parameters in BIND, see <u>SNA Formats</u> for details.

An IPR or IPM is sent to return a Pacing indication as discussed in the preceding section. For IPR, no RU accompanies the TH and RH. The format of the IPR and IPM are defined in <u>SNA Formats</u>

Solicited and unsolicited IPMs flow at network priority between T2.1 nodes. Reset acknowledgment IPMs flow at the priority of the session (between T2.1 nodes or in a subarea network) so that it does not overtake any BIUs en route, thereby truly delimiting the end of the reset window. (See page 6.2-22 for priority setting, and see <u>SNA Type 2.1 Node Reference</u> for details on transmission priority.)

SESSION-LEVEL PACING ALGORITHMS

A session can be viewed as a succession of adjacent pairs of session-level processing points (involving half-sessions and boundary function components). The connection between a pair of these processing points is called a <u>session stage</u>.

While adaptive pacing is the preferred mode, the session pacing algorithm is able to support fixed-pacing protocols when the other partner of a session stage does not support adaptive pacing.

The session-level pacing modes for both sender and receiver are determined during the BIND negotiation time for each session stage. The sender TC sends data, pacing requests, and IPM acknowledgments. The receiver TC receives data and pacing requests, generates appropriage pacing responses, has its buffer manager reserve buffers, and determine the next-window size (NWS).

<u>Session-Level Adaptive Pacing Algorithm</u>

Session-level adaptive pacing uses the following flow-control messages:

- Pacing Requests
- ISOLATED PACING MESSAGES (IPMs)
  - Solicited
  - Unsolicited

- Reset Acknowledgments

A pacing request is a normal-flow request that has the pacing indicator (PI) in the RH set to PAC. A pacing request indicates that an RU is the first RU in the last window allowed, and requests that a new window be allowed.

A solicited IPM is generated by the receiver when the receiver has reserved the buffers for a new window in response to a pacing request sent by the sender. A solicited IPM informs the sender what its new pacing window size is that it can send following the current window it is sending.

An unsolicited IPM is generated by the receiver as a result of congestion in the node and is used to reset to 0 the residual pacing count in the sender and to start a new pacing window if the next-window size specified in the IPM is not 0. The sender immediately sends a reset acknowledgment IPM when an unsolicited IPM arrives; this delimits the end of the window being reset and allows the sender of the unsolicited IPM to reallocate resources.

A reset acknowledgment IPM is an immediate acknowledgment that an unsolicited IPM has reset the residual pacing count. The reset acknowledgment IPM also echoes the next-window size specified in the unsolicited IPM.

OPERATION OF THE SENDER

The following algorithm is implemented in all nodes that use adaptive pacing. The sender mechanism is unaffected by the algorithm used to determine the pacing window sizes.

Whenever the remaining number of the current window (hereafter referred to as the residual pacing count) is non 0 and a normal-flow request is at the head of the pacing queue, it is sent and the residual pacing count is decremented by 1. If, at any time, the residual pacing count is 0 and the next-window size is non 0, the residual pacing count is set to the next-window size, the next-window size is reset to 0, and the Pacing indicator is set to PAC in the RH of the next request sent. Whenever anything besides a normal-flow request is at the head of the queue it is sent right away.

When a solicited or unsolicited IPM is received, the new next-window size is obtained from the IPM.

When it is an unsolicited IPM, a reset acknowledgment IPM is returned immediately and the residual pacing count is reset to 0. The reset window indicator (RWI) bit in an unsolicited IPM is always set to 1 (indicating a reset action is needed).

The Request Larger Window indicator (RLWI) in the request RH is used by the pacing request sender to indicate to the receiver that it would like a larger window size. The RLWI has meaning only in pacing requests and is reserved in responses. When the sender receives a solicited IPM and the residual pacing count is 0 (the entire previous pacing window has been sent) and the send pacing queue is not empty (more requests are available to send), the sender will send the next pacing request with RLWI set to request a larger window (set to a value of 1, or RLW). Otherwise, the sender sends a pacing request with RLWI set to ¬RLW, indicating a larger window size is not needed.

The pacing request receiver uses the RLWI value in calculating the next-window size value to be sent in the next solicited IPM.

The sender is initialized by the session manager with a next-window size of 1 and a residual pacing count of 0 when the session is activated.

OPERATION OF THE RECEIVER

The receiver has control and responsibility for session-level pacing, as necessary to manage its buffers.

An unsolicited IPM with the RWI set to 1 is sent whenever the node becomes congested. The receiver may not have more than 1 outstanding unsolicited IPM. Another IPM (solicited or unsolicited) may not be sent until the reset acknowledgment IPM is received. When an unsolicited IPM is outstanding, the receiver's buffer manager does not reserve buffers for the next window as a result of a pacing request that was received before the reset acknowledgment IPM. (A solicited IPM will not be sent for pacing requests that are received before the reset acknowledgment IPM.)

Fixed pacing is implemented as a special case of adaptive pacing. The "piggybacked" pacing response (i.e. on a regular response--1 with (DR1I, DR2I)≠00--occurs only for fixed pacing.

When the data sender receives an unsolicited IPM with a next-window size of 0, it is stopped from sending normal-flow requests. When the data receiver gets the reset acknowledgment IPM (in response to the unsolicited IPM) the sender's window size (and its residual pacing count) has been reset to 0 and no further normal-flow requests may be sent by the sender. In order to allow the sending of more normal-flow requests, a solicited IPM, which always contains a non 0 next-window size, is sent by the receiver. The sender may resume sending requests when this solicited IPM has been received. The next-window size value used in this solicited IPM is typically small (e.g., 1).

Window sizes can vary from 1 to 32767 in solicited IPMs and 0 to 32767 in unsolicited IPMs.

Figure 6.2-4 and Figure 6.2-5 illustrate adaptive session-level pacing, with session data traffic flowing from left to right on a given session stage. See "Appendix B. Buffer Manager" for details of how buffers are managed to perform session-level pacing. An analogous set of exchanges could occur in the opposite direction. See "Chapter 4. LU Session Manager" for details of how session-level pacing is set up before these flows occur.

```
       (sender)                                            (receiver)
      NWS     RPC                                          NWS     RPC
    ---------------                                     ---------------
                        BIND
                       ------------------------>
                        +RPSP(BIND)
 1     1       0       <------------------------            1        0
                         RQ,PAC,RLW                                  0
 2     0       0       ------------------------>            0        0
                         IPM (solicited, NWS=2)
 3     2       0       <------------------------            2        0
                         RQ,PAC,RLW
 4     0       1       ------------------------>            0        1
                             IPM (solicited, NWS=3)
 5                                   ------------           3        1
                         RQ,¬PAC    |
 6     0       0       -------------|---------->            3        0
                                    |
 7     3       0       <------------
                         RQ,PAC,¬RLW
 8     0       2       ------------------------>            0        2
                         IPM (solicited, NWS=2)
 9     2       2       <------------------------            2        2
                         RQ,¬PAC
10     2       1       ------------------------>            2        1
                         RQ,¬PAC
11     2       0       ------------------------>            2        0
                         RQ,PAC,¬RLW
12     0       1       ------------------------>            0        1
                         RQ,¬PAC
13     0       0       ------------------------>            0        0
```

LEGEND

NWS	next-window size
RPC	residual pacing count
RQ	request
PAC	Pacing indicator set to 1
¬PAC	Pacing indicator set to 0
RLW	Request Larger Window indicator set to 1
¬RLW	Request Larger Window indicator set to 0

Figure 6.2-4. Session-Level Pacing with Solicited IPMs

The comments below correspond to the numbers in Figure 6.2-4 on page 6.2-10.

1. When a session using adaptive pacing is initialized, it starts with a next-window size of 1 and a residual pacing count of 0.

2. The first request in a window has the Pacing indicator set to PAC. The RLWI is set to RLW in this example.

3. The receiver increases the sender's next-window size to 2. The sender's residual pacing count is at 0 and the pacing queue is not empty when this IPM is received, so the RLWI is set to RLW in the next pacing request sent.

4. The first request in a window has the Pacing indicator set to PAC. The Request Larger Window indicator is set to RLW (determined when the last solicited IPM was received).

5. The receiver increases the sender's next window to 3.

6. This is not the first RU in a window so the Pacing indicator is set to ¬PAC.

7. The solicited IPM is received. The sender's residual pacing count was at 0, but the pacing queue was empty when this IPM was received, so the RLWI will be set to ¬RLW in the next pacing request sent.

8. The first request in a window sets the Pacing indicator to PAC. The RLWI is set to ¬RLW (determined when the last solicited IPM was received).

9. The receiver decreases the sender's next window to 2. (The receiver decreases or increases the sender's next-window size according to the availability of its buffer storage. The sender's residual pacing count was not at 0 when this IPM was received, so the RLWI will be set to ¬RLW in the next pacing request sent.

10. The first request in a window sets the Pacing indicator to PAC. The RLWI is set to ¬RLW (determined when the last solicited IPM was received).

```
      (sender)                                          (receiver)
       NWS     RPC                                       NWS     RPC
      ─────────────                                     ─────────────
        3       0                                         3       0
                         RQ,PAC,¬RLW
  1     0       2     ──────────────────────────>         0       2
                              IPM (unsolicited, NWS=0)
  2                               ┌──────────────         0       2
                         RQ,¬PAC  │
  3     0       1     ────────────│─────────────>         0       1
                                  │
  4     0       0     <───────────┘
                       IPM (reset acknowledgment)
  5     0       0     ──────────────────────────>         0       0
                         IPM (solicited, NWS=2)
  6     2       0     <──────────────────────────         2       0
                            IPM (unsolicited, NWS=0)
  7                                 ┌────────────         2       0
                       RQ,PAC,RLW   │
  8     0       1     ──────────────│───────────>         0       1
                                    │
  9     0       0     <─────────────┘
                       IPM (reset acknowledgment)
 10     0       0     ──────────────────────────>         0       0
                        IPM (solicited, NWS=1)
 11     1       0     <──────────────────────────         1       0
```

LEGEND

NWS	next-window size
RPC	residual pacing count
RQ	request
PAC	Pacing indicator set to 1
¬PAC	Pacing indicator set to 0
RLW	Request Larger Window indicator set to 1
¬RLW	Request Larger Window indicator set to 0

Figure 6.2-5. Session-Level Pacing with Unsolicited IPMs

The comments below correspond to the numbers in Figure 6.2-5 on page 6.2-11.

1. The session has been active for a while.

2. The first request in a window sets the pacing indicator to PAC.

3. The receiver is congested and sends an unsolicited IPM with a next-window size of 0.

4. The sender sends, and the receiver receives, another request before the unsolicited IPM arrives at the sender.

5. The sender receives the unsolicited IPM causing it to reset its residual pacing count, use the value 0 in the IPM for its next-window size, and send a reset acknowledgment IPM. The sender cannot send anything more until it receives another IPM.

6. When the IPM reset acknowledgment is received, the receiver can then free the

buffers (1 in this case) that were not used by the sender by resetting the residual pacing count to 0. By having sent an unsolicited IPM with a next-window size of 0, the reset acknowledgment IPM acts as a request by the sender to allow it to send a new window; when the receiver node is no longer congested, it will send a solicited IPM to the sender.

7. The sender is restarted when it receives a solicited IPM with a next-window size of 2.

8. The receiver goes back into a congested state shortly after sending the solicited IPM, and sends an unsolicited IPM. The unsolicited IPM specifies a next-window size of 0.

9. The sender sends, and the receiver receives, a request before the unsolicited IPM is received. The receiver ignores the Pacing indicator because an unsolicited IPM is outstanding.

10. The sender receives the unsolicited IPM causing it to reset its next-window size, use the value (0) in the IPM for its next-window size, and send a reset acknowledgment IPM. The sender cannot send anything more until it receives another IPM.

11. When it receives the reset acknowledgment IPM, the receiver can free the buffers (1, in this case) that were not used by the sender by resetting the residual pacing count to 0.

12. The sender is restarted when it receives a solicited IPM with a next-window size of 1.

Session-Level Fixed Pacing Algorithm

The session-level fixed pacing algorithm is similar to the adaptive pacing algorithm, but with the following differences:

1. The buffer manager always generates the same fixed number for the next-window size, as determined at session activation time.

2. When the receiver would generate an IPM, it generates an IPR or a "piggybacked" pacing response instead.

3. The sender saves the value for the fixed window size determined at session activation time.

4. The sender receives an IPR instead of an IPM and takes its next-window size to be the value it saved at session activation time.

5. The buffer manager cannot cause an unsolicited IPM to be sent. The window size is fixed and IPR is always sent as a solicited pacing response.

SEGMENT REASSEMBLY FUNCTION

Conceptually, segment reassembly is a path control (PC) function; however, the inbound segment reassembly function is modeled in this book (see page 6.2-28). If the half-session supports segment reassembly, it reassembles segments (associated with the same BIU) into a whole BIU. When segments are received, the half-session checks for the maximum receive RU size (See TC.SEGMENT_RCV_CHECKS (page 6.2-24)).

See <u>SNA Type 2.1 Node Reference</u> for more details on BIU segmentation.

TC.INITIALIZE

FUNCTION:	This procedure sets up session parameters needed by TC. This procedure is called by the half-session initialization procedure (see Chapter 6.0) when the session is being activated. The LOCAL data structure fields that are used only by TC are initialized by this procedure.
INPUT:	INIT_HS from SM, containing BIND information
OUTPUT:	The LOCAL fields used only by TC are initialized
NOTES:	1. The identifier of the path control with which this half-session is associated, the role (primary or secondary) of the half-session, and LOCAL.SENSE_CODE are initialized prior to calling this procedure. 2. LOCAL.SENSE_CODE is set to X'00000000' by called routines, if the TC initialization was successful. Otherwise, it is set to a nonzero sense data value by called routines. 3. RCV_PACING.NWS is not set to 0 by SM in any case. Send pacing queue (Q_PAC) should be created in this procedure. This queue is used to send normal-flow requests to path control.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
TC.EXCHANGE_CRV	page 6.2-15
LOCAL	page 6.0-7
INIT_HS	page A-13

```
If this is a primary half-session then
   Set LOCAL.MAX_RCV_RU_SIZE to INIT_HS.SHORT_BIND_IMAGE.SEC_SEND_MAX_RU_SIZE.
Else (secondary half-session)
   Set LOCAL.MAX_RCV_RU_SIZE to INIT_HS.SHORT_BIND_IMAGE.PRI_SEND_MAX_RU_SIZE.
Set LOCAL.SQN_RCV_CNT to 0.

Initialize LOCAL.COMMON_CB fields to the following values:
   Set CALLER to HS, LFSID to INIT_HS.LFSID.
   Set PATH_CONTROL_ID to INIT_HS.PATH_CONTROL_ID.
   Set DYNAMIC_BUFFER_POOL_ID to INIT_HS.DYNAMIC_BUFFER_POOL_ID.
   Set LIMITED_BUFFER_POOL_ID to INIT_HS.LIMITED_BUFFER_POOL_ID.
   Set TRANSMISSION_PRIORITY to INIT_HS.TRANSMISSION_PRIORITY.
   Set NUM_BUFS_PER_RU to 1, SET_RLWI to ¬RLW.
   Set SEND_PACING.RPC and RECEIVE_PACING.RPC to 0.
   If this is a primary half-session then
      Set SEND_PACING.NWS to INIT_HS.SHORT_BIND_IMAGE.PRI_SEND_WINDOW_SIZE.
      Set RECEIVE_PACING.NWS to INIT_HS.SHORT_BIND_IMAGE.PRI_RCV_WINDOW_SIZE.
   Else (this is a secondary half-session)
      Set SEND_PACING.NWS to INIT_HS.SHORT_BIND_IMAGE.SEC_SEND_WINDOW_SIZE.
      Set RECEIVE_PACING.NWS to INIT_HS.SHORT_BIND_IMAGE.SEC_RCV_WINDOW_SIZE.
```

Following are valid send/receive pacing type combinations:

SENDER	RECEIVER
adaptive	adaptive
fixed	fixed
none	fixed

```
If INIT_HS.SHORT_BIND_IMAGE.ADAPTIVE_PACING = SUPPORTED then
   Set SEND_PACING.TYPE and RECEIVE_PACING.TYPE to ADAPTIVE
Else
   Set RECEIVE_PACING.TYPE to FIXED.
   If LOCAL.COMMON_CB.SEND_PACING.NWS > 0 then
      Set SEND_PACING.TYPE to FIXED.
   Else
      Set SEND_PACING.TYPE to NONE.
Set FIRST_WS to SEND_PACING.NWS.
Set UNSOLICITED_IPM_OUTSTANDING to FALSE.
Set ADJUST_FOR_IPM_ACK_OUTSTANDING to FALSE.
Set UNSOLICITED_NWS to 0.
Set RESERVE_FLAG to NO.
If INIT_HS.SHORT_BIND_IMAGE.WHOLE_BIU_REQUIRED = YES then
   Set LOCAL.SEQMENTING_SUPPORTED to FALSE.
Else
   Set LOCAL.SEQMENTING_SUPPORTED to TRUE.
Set LOCAL.CRYPTOGRAPHY to NO.
If INIT_HS.SHORT_BIND_IMAGE.CRYPTO_SESSION_LEVEL = MANDATORY then
   Set LOCAL.CRYPTOGRAPHY to YES.
   Call TC.EXCHANGE_CRV(INIT_HS) (page 6.2-15)
    to exchange cryptography verification (CRV) information.
```

TC.EXCHANGE_CRV

FUNCTION: This procedure handles the exchange of cryptography verification (CRV).

This procedure is called from a primary half-session to initiate the exchange CRV request with a secondary and receive RSP(CRV), or called from a secondary half-session to receive CRV request from a primary and return RSP(CRV).

INPUT: INIT_HS, containing the enciphered pseudo-random value to be used as a test value (and later as the session seed)

This value is enciphered using the session key as the cryptography key and 0 as the initial chaining value.

OUTPUT: LOCAL.SENSE_CODE is set to the sense data carried on the negative RSP(CRV), if CRV exchange failed; else it is set to X'00000000'

The secondary half-session sends a RSP(CRV) to the primary. A positive RSP(CRV), if CRV exchange successful; else a negative RSP(CRV)

NOTES:

1. LOCAL.HALF_SESSION is initialized before this procedure is called. LOCAL.SENSE_CODE may be changed by the procedures called from this procedure.

2. The initialization of a primary TC instance involves sending an MU containing a CRV request and receiving an MU containing a RSP(CRV). The initialization of a secondary TC instance involves sending an MU containing a RSP(CRV) and receiving an MU containing a CRV request.

3. The buffer that the CRV request was in is reused by secondary half-session (with appropriate RH bit settings) to send a RSP(CRV).

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
SEND_MU	page 6.2-20
TC.BUILD_CRV	page 6.2-17
TC.CRV_FORMAT_CHECK	page 6.2-18
LOCAL	page 6.0-7
INIT_HS	page A-13
MU	page A-29
PATH CONTROL	T2.1 Node Reference

```
If primary half-session then
   Call TC.BUILD_CRV(INIT_HS,MU_PTR) (page 6.2-17)
    to build a CRV exchange request.
   Call SEND_MU(MU, LOCAL.COMMON_CB) (page 6.2-20)
    to send CRV exchange request to path control (to secondary half-session).
   Receive RSP(CRV) from path control (sent from secondary half-session).
   Call TC.CRV_FORMAT_CHECK(MU) (page 6.2-18).
   If LOCAL.SENSE_CODE = X'00000000' and MU.RH.RTI = NEG then
      Set LOCAL.SENSE_CODE to the sense data carried on the negative RSP(CRV).
   Call buffer manager (FREE_BUFFER, buffer address)
    to release the buffer containing the RSP(CRV).  (Appendix B)
```

```
Else (secondary half-session)
   Receive CRV request from path control (sent from primary half-session).
   Call TC.CRV_FORMAT_CHECK(MU) (page 6.2-18).
   If LOCAL.SENSE_CODE = X'00000000' then
      (Check that the CRV test value was correctly encoded by the session partner)
      Decipher the test value (bytes 2-9 of MU.RU).  The cryptography key is the
       session key, the initial chaining value is 0.
      Invert the bits in the first 4 bytes of the deciphered test value
       (i.e., exclusive-OR the deciphered test value with X'FFFFFFFF00000000').
      Compare the resulting value with the value
       that was generated by the session manager in the positive RSP(BIND).
      If values are not equal then
         Set LOCAL.SENSE_CODE to X'08350001' (indicating Invalid Parameter).
      Else
         Set LOCAL.SENSE_CODE to X'00000000' (test value was correctly encoded by
          primary half-session).

   If LOCAL.SENSE_CODE = X'00000000' then
      Set RH.RRI to RSP, RH.RTI to POS to indicate a positive response.
      Call SEND_MU(MU, LOCAL.COMMON_CB) (page 6.2-20)
       to send a positive RSP(CRV) to path control (to primary half-session).
   Else
      Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
       containing the erroneous CRV request.  (Appendix B)
      (The half-session router will cause UNBIND to be sent.)
```

TC.BUILD_CRV

FUNCTION:	This procedure builds an MU (containing the CRV request) by appropriately initializing the TH, RH and RU fields.
INPUT:	INIT_HS, containing the enciphered pseudo-random value to be used as a test value
OUTPUT:	The address of the MU, it contains a CRV request The MU is initialized by this procedure (MU.RU contains the cryptography seed).
NOTES:	1. For the actual TH and RH bit settings see <u>SNA Formats</u>. 2. If a permanent buffer is not available, a demand buffer is requested by this procedure instead. 3. Both CRV request and RSP(CRV) are sent expedited. 4. The session cryptography seed is retained from INIT_HS record.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
CRV_RQ_RU	page A-27
INIT_HS	page A-13
MU	page A-29


```
Call buffer manager (GET_BUFFER, permanent buffer pool ID, no wait)
 to request a permanent buffer to build an MU.  (Appendix B)
If permanent buffer is not available then
   Call buffer manager (GET_BUFFER, demand buffer size, no wait)
     to request a demand buffer.  The demand buffer size is maximum RU size
     plus MU overhead.  (Appendix B)
       If demand buffer is not available then
          Perform half-session ABEND processing.

Initialize MU.TH reserved fields and constant fields.
Set EFI to EXP, SNF to any 16-bit unique value (implementation-dependent).
(CRV is sent expedited- rather than normal-flow, so is not related to the
 HS normal-flow send sequence number [LOCAL.SQN_SEND_CNT].)
Set BBIUI to BBIU, EBIUI to EBIU.
(Expedited-flow request [or response] should be sent as a whole BIU.)

Initialize MU.RH reserved fields, constant fields and set the rest of
 RH bits to the following values:
   RQ, SC, FMH, ¬SD, BC, EC, RQD1, ¬RLW, ¬QR, ¬PAC, ¬BB, ¬CD, CODE0, ¬ED, ¬PD, ¬CEB


Decipher the test value in the INIT_HS record.  Use the session key
 as the cryptography key, and 0 as the initial chaining value.
Transform the result by inverting the bits in the first four bytes (i.e.,
 exclusive-OR the test value with X'FFFFFFFF00000000').
Enciphering above transformed value.  Use the session key as the cryptography key
 and 0 as the initial chaining value.
Set CRV_RQ_RU.CRYPTO_SEED to above enciphered value.
Set RU to CRV_RQ_RU.            (page A-27)
Set the MU.DCF to indicate the length of RH and RU (CRV_RQ_RU).
```

TC.CRV_FORMAT_CHECK

FUNCTION:	This procedure checks the RH bits of the CRV request or RSP(CRV) received from path control (from the partner half-session). All of these checks are optional. An implementation may choose to do all, some, or none of them.
INPUT:	MU, containing a CRV request or RSP(CRV)
OUTPUT:	LOCAL.SENSE_CODE is set to X'00000000' if all RH bits are properly set; otherwise, it is set to a nonzero value which indicates error.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
LOCAL	page 6.0-7
MU	page A-29
PATH CONTROL	T2.1 Node Reference


```
Calculate the length of RU data.
If RRI = RQ then
   Select in the following order, based on the RH bits:
      When LOCAL.HALF_SESSION = PRI
         Set LOCAL.SENSE_CODE to X'20090000'.
      When RU_CTGY ≠ SC
         Set LOCAL.SENSE_CODE to X'20090000'.
      When (SDI ≠ SD and the length of RU data < 1) or
       (SDI = SD and the length of RU data < 5)
         Set LOCAL.SENSE_CODE to X'10020000'.
      When (SDI ≠ SD and CRV request code ≠ X'C0') or
       (SDI = SD and CRV request code ≠ X'C0')
         Set LOCAL.SENSE_CODE to X'20090000'.
      When FI ≠ FMH
         Set LOCAL.SENSE_CODE to X'400F0000'.
```

> A request containing sense code is an exception request. When path control receives an anonymous request and not be able to pass the response back to the anonymous request sender, path control sets the sense code and appends it to RU data to inform the receiver about the error.

```
      When SDI = SD
         Set LOCAL.SENSE_CODE to the sense code
          carried in the RU.
      When BCI ≠ BC
         Set LOCAL.SENSE_CODE to X'400B0000'.
      When ECI ≠ EC
         Set LOCAL.SENSE_CODE to X'400B0000'.
      When response category ≠ RQD1
         Set LOCAL.SENSE_CODE to X'40140000'.
      When EFI ≠ EXP
         Set LOCAL.SENSE_CODE to X'40110000'.
      When QRI = QR
         Set LOCAL.SENSE_CODE to X'40150000'.
```

```
      When PI = PAC
         Set LOCAL.SENSE_CODE to X'40080000'.
      When BBI = BB
         Set LOCAL.SENSE_CODE to X'400C0000'.
      When EBI = EB
         Set LOCAL.SENSE_CODE to X'400C0000'.
      When CDI = CD
         Set LOCAL.SENSE_CODE to X'400D0000'.
      When CSI = CODE1
         Set LOCAL.SENSE_CODE to X'40100000'.
      When EDI = ED
         Set LOCAL.SENSE_CODE to X'40160000'.
      When PDI = PD
         Set LOCAL.SENSE_CODE to X'40170000'.
      When CEBI = CEB
         Set LOCAL.SENSE_CODE to X'400C0000'.
Else (CRV response)
   Select in the following order, based on the RH bits:
      When LOCAL.HALF_SESSION = SEC
         Set LOCAL.SENSE_CODE to X'20090000'.
      When RU_CTGY ≠ SC
         Set LOCAL.SENSE_CODE to X'20090000'.
      When (RTI = POS and the length of RU data < 1) or
       (RTI = NEG and the length of RU data < 5)
         Set LOCAL.SENSE_CODE to X'10020000'.
      When (SDI ≠ SD and CRV request code ≠ X'C0') or
       (SDI = SD and CRV request code ≠ X'C0')
         Set LOCAL.SENSE_CODE to X'20090000'.
      When FI ≠ FMH
         Set LOCAL.SENSE_CODE to X'400F0000'.
      When BCI ≠ BC
         Set LOCAL.SENSE_CODE to X'400B0000'.
      When ECI ≠ EC
         Set LOCAL.SENSE_CODE to X'400B0000'.
      When EFI ≠ EXP
         Set LOCAL.SENSE_CODE to X'40110000'.
      When DR1I ≠ DR1 or DR2I = DR2
         Set LOCAL.SENSE_CODE to X'40140000'.
      When (RTI = POS and SDI = SD) or
       (RTI = NEG and SDI = NOT_SD)
         Set LOCAL.SENSE_CODE to X'40130000'.
      When QRI = QR
         Set LOCAL.SENSE_CODE to X'40150000'.
      When PI = PAC
         Set LOCAL.SENSE_CODE to X'40080000'.
```

SEND_MU

FUNCTION:		This procedure sends the input MU to path control.
INPUT:		MU, containing normal-flow or expedited-flow request (or response); LOCAL.COMMON_CB, containing appropriate pacing bits setting
OUTPUT:		The MU is sent to PC or placed on Q_PAC, if no errors are found. If send pacing is active, the send pacing counts in the CB may be updated.
NOTES:	1.	If pacing is supported, the MU may be placed on Q_PAC (send pacing queue) rather than sent directly to path control.
	2.	If required the MU data is enciphered before sending out.
	3.	The pacing counts in the LOCAL.COMMON_CB (page 6.0-8) are set before this procedure is called.

Referenced procedures, FSMs, and data structures:

SEND_TO_PC	page 6.2-22
MU	page A-29
LOCAL	Chapter 6.0
PATH CONTROL	T2.1 Node Reference

```
Select, in the following order, based on the TH and RH bits:
   When TH.EFI = EXP (indicating the MU is sent expedited-flow)
      Call SEND_TO_PC(MU, LOCAL.COMMON_CB) (page 6.2-22)
       to send the MU to path control directly (expedited-flow is not paced).
   When TH.BBIUI = ¬BBIU or RH.RRI = RQ  (normal-flow request)
      If RH.RU_CTGY=FMD, RU data is present and cryptography is required then
         If the RU length is not an even multiple of 8 then
            Pad the RU to an integral number of eight bytes.  The padding bytes are
             padded to the end and contain unpredictable values, except for the last
             pad byte, which contains an unsigned 8-bit binary count of the pad bytes
             preceding it.  If only one byte of pad is required, it is the count byte
             itself and contains 1.
            Set RH.PDI to PD (indicating pad character string is present).
         Else (multiple of 8)
            Set RH.PDI to ¬PD.

      Encipher the RU data.
       (Execute the Data Encryption Standard [DES] algorithm, using the session key
       as the cryptography key and the session seed as the initial chaining value.
       The manner in which the session key and the session seed are made available
       to this procedure is implementation-defined.)
      If data enciphering fails then
         Set LOCAL.SENSE_CODE to X'08480000' (cryptography function inoperative).
         Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
          containing the erroneous MU.  (Appendix B)
      Else
         Set LOCAL.SENSE_CODE to X'00000000'.
         Set RH.EDI to ED (indicating RU data is enciphered).
         If LOCAL.COMMON_CB.SEND_PACING.TYPE ≠ NONE (indicating pacing is active) then
            If the Q_PAC contains any MUs or TH.BBIUI = BBIU then
               If the sum of LOCAL.COMMON_CB.SEND_PACING.RPC and
                LOCAL.COMMON_CB.SEND_PACING.NWS is 0 then
                  Put the request MU on the pacing queue (the send pacing count
                   has gone to zero).
         Else
            Call SEND_TO_PC(MU, LOCAL.COMMON_CB) (page 6.2-22).

   When RRI = RSP
      If LOCAL.COMMON_CB.SEND_PACING.TYPE = NONE or
       the Q_PAC does not contain any MUs or
       QRI = ¬QR  then
         Call SEND_TO_PC(MU, LOCAL.COMMON_CB) (page 6.2-22).
      Else
         Put the response MU on the send pacing queue.
```

SEND_PACING

FUNCTION:	This procedure updates the send pacing counts in the common control block and sets the pacing bits (RH.PI and RH.RLWI of the MU being sent) to the appropriate value. This procedure will never be called when both the residual pacing count and the next window size are 0 (see page 6.2-20).
INPUT:	MU, containing a normal-flow request (beginning BIU); LOCAL.COMMON_CB, containing appropriate pacing bits setting
OUTPUT:	The pacing bits in the request MU may be changed; the SET_RLWI bit and pacing counts in the LOCAL.COMMON_CB may be changed.

Referenced procedures, FSMs, and data structures:

MU	page A-29
LOCAL	Chapter 6.0

```
If LOCAL.COMMON_CB.SEND_PACING.RPC > 0 then
   Decrement LOCAL.COMMON_CB.SEND_PACING.RPC by 1.
Else (start the next window)
   Set LOCAL.COMMON_CB.SEND_PACING.RPC to (LOCAL.COMMON_CB.SEND_PACING.NWS - 1).
   Set LOCAL.COMMON_CB.SEND_PACING.NWS to 0.
   Set PI to PAC (to show a pacing response is required).
   Set MU.RLWI to LOCAL.COMMON_CB.SET_RLWI (SET_RLWI value was stored in the
    LOCAL.COMMON_CB when the pacing response was received).
   Reset LOCAL.COMMON_CB.SET_RLWI to ¬RLW.
```

SEND_TO_PC

FUNCTION:	This procedure sends an MU to path control.
INPUT:	MU, LOCAL.COMMON_CB
OUTPUT:	The input MU is sent to path control with the HS_TO_PC header filled in and, if necessary, the send pacing counts in LOCAL.COMMON_CB may also be updated.

Referenced procedures, FSMs, and data structures:

SEND_PACING	page 6.2-21
MU	page A-29
IPM_RU	page 6.2-33
LOCAL	Chapter 6.0
PATH CONTROL	T2.1 Node Reference

```
Set MU.HEADER_TYPE to HS_TO_PC.
Fill in HS_TO_PC header with the LFSID and TRANSMISSION_PRIORITY
 values from the LOCAL.COMMON_CB
 (TRANSMISSION_PRIORITY indicates the priority of the session).

If this is the beginning of a BIU (BBIUI = BBIU) then
   If this MU contains an IPM (MU.RH.RRI=RSP, RH.PI=PAC, ¬DR1, ¬DR2 and
    LOCAL.COMMON_CB.SEND_PACING.TYPE is ADAPTIVE then
      If IPM_RU.TYPE is not set to indicate an IPM reset acknowledgement then
         Set HS_TO_PC.TRANSMISSION_PRIORITY to NETWORK.
         (Solicited or unsolicited IPM flows at network priority).
   Else (MU doesn't contain an IPM)
      If this is a normal-flow request and
       is being paced (LOCAL.COMMON_CB.SEND_PACING.TYPE ≠ NONE) then
         Call SEND_PACING(MU, LOCAL.COMMON_CB) (page 6.2-21).

   Send the MU to path control.
   If sending MU failed (i.e., the path control doesn't exist any more) then
      Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
       that the MU was in.  (Appendix B)
       (The session will be brought down soon by SM).
```

TC.RCV

FUNCTION:	This procedure receives MUs sent from path control. The format and state checks are made in this procedure.
INPUT:	MU, containing a request or response is received from the HS router (see Chapter 6.0)
OUTPUT:	If a pacing response is received, it is processed in TC and will not be passed to DFC. TC updates the send pacing counts in the LOCAL.COMMON_CB and the pacing response is discarded; else, DFC is called to receive and process the MU
NOTES:	1. If the RU data (received from path control process) has been segmented, the segments are reassembled in this procedure. The data that goes out (from half-session process) is not segmented. 2. The receive pacing counts in the CB may be changed according to the pacing bits (RH.PI, RH.RLW) setting in the request. 3. Upon receiving the request MU, the sequence number (LOCAL.SQN_RCV_CNT) may be updated. 4. If an error is encountered, LOCAL.SENSE_CODE is set to a nonzero value by a called routine, the HS router causes an UNBIND to be generated; otherwise, it is set to X'00000000'.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
TC.DECIPHER_RU	page 6.2-32
TC.SEGMENT_RCV_CHECKS	page 6.2-24
SEGMENT_REASSEMBLY	page 6.2-28
TC.BIU_RCV_CHECKS	page 6.2-25
RECEIVE_PACING	page 6.2-27
RCV_PACING_RSP	page 6.2-29
DFC_RCV	page 6.1-24
LOCAL	page 6.0-7
MU	page A-29
PATH CONTROL	T2.1 Node Reference

```
Call TC.SEGMENT_RCV_CHECKS(MU) (page 6.2-24).
If LOCAL.SENSE_CODE = X'00000000' then
   If normal-flow request and BBIUI = BBIU then
      Call RECEIVE_PACING(MU, LOCAL.COMMON_CB) (page 6.2-27)
   If normal-flow (request or response) MU then
      If (request MU and BBIUI = BBIU) or (BBIUI = ¬BBIU) then
         Call SEGMENT_REASSEMBLY(the MU address) (page 6.2-28).
         If LOCAL.SENSE_CODE = X'00000000' then
            If a reassembled MU is present then
               Call TC.BIU_RCV_CHECKS(MU) (page 6.2-25).
               If LOCAL.SENSE_CODE = X'00000000' then
                  If normal-flow request then
                     If cryptography is required, RU data is present,
                      RH.RU_CTGY = FMD and no sense data is present then
                        Call TC.DECIPHER_RU(MU) (page 6.2-32).
                        If LOCAL.SENSE_CODE = X'00000000' then
                           If LOCAL.SQN_RCV_CNT = 65535 then
                            (max sequence number is [2**16 - 1])
                              Set LOCAL.SQN_RCV_CNT to 0. (sequence number wrapped)
                           Else
                              Increment LOCAL.SQN_RCV_CNT by 1.
                        Call DFC_RCV(MU) (page 6.1-24).
                  Else
                     If response MU and RH.PI = PAC then
                        Call RCV_PACING_RSP(MU_PTR, LOCAL.COMMON_CB)
                        (page 6.2-29)
                        If an MU is present then
                           Call DFC_RCV(MU) (page 6.1-24).
```

TC.SEGMENT_RCV_CHECKS

FUNCTION:	This procedure performs receive checks on all segments received from PC.
INPUT:	MU, containing a segment (perhaps the only segment in a BIU)
OUTPUT:	LOCAL.SENSE_CODE is set to reflect the receive checks.
NOTE:	Expedited-flow MU may not be segmented.

Referenced procedures, FSMs, and data structures:


```
Select, in the following order, based on the following conditions:

   When ¬LOCAL.SEGMENTING_SUPPORTED and MU contains one of the segments (¬[BBIU and EBIU])
      Set LOCAL.SENSE_CODE to X'80070001' (receipt of segment not supported).
   When BBIU, ¬EBIU, ¬(NORMAL, RQ)
      Set LOCAL.SENSE_CODE to X'80070003' (cannot reassemble response or expedited-flow
       request).
   When NORMAL, RQ, BBIU and segment reassembly is in progress
      Set LOCAL.SENSE_CODE to X'80070000' (BBIU segment not preceded by EBIU segment).
   When ¬BBIU and (segment reassembly is not in progress or this MU flows expedited)
      Set LOCAL.SENSE_CODE to X'80070000' (¬BBIU segment not preceded by BBIU segment).
   When ¬BBIU and MU.TH.SQN (in the BBIU segment) ≠ MU.TH.SQN (in the ¬BBIU segment)
    (¬BBIU segment has a sequence number different from the BBIU segment's
    sequence number.)
      Set LOCAL.SENSE_CODE to X'80070000'.
   When BBIU, ¬EBIU and DCF < 10 (first in segment must have at least 10 bytes)
      Set LOCAL.SENSE_CODE to X'80070000'.

If BBIU, LOCAL.SENSE_CODE = X'00000000'  then
   Call MU_PACING_CHECKS(MU, COMMON_CB, LOCAL.SENSE_CODE) (page 6.2-26).
```

TC.BIU_RCV_CHECKS

FUNCTION:	This procedure performs receive checks on BIUs.
INPUT:	MU, containing a BIU
OUTPUT:	LOCAL.SENSE_CODE is set to reflect the receive checks.
NOTE:	LOCAL.CRYPTOGRAPHY is properly set before this procedure is called.

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
LOCAL	page 6.0-7
MU	page A-29
PATH CONTROL	T2.1 Node Reference

```
If MU.RH.RU_CTGY is not set to FMD or DFC then
   Set LOCAL.SENSE_CODE to X'10070000'.
Else (FMD or DFC)
   Set RU length to 0 (If RU data is not present in the receiving MU,
    then 0 is the minimum RU length).
   If sense data is present (MU.RH.SDI = SD) then
      Increment RU length by 4 (the length of sense data).
   If MU.RH.RU_CTGY = DFC then
      Increment RU length by 1 (the length of request code).
   If DCF < length of (RU + RH) then
      Set LOCAL.SENSE_CODE to X'10020000'.
   Else
      If sense data is present then
         If received MU contains a request then
```

> A request containing a sense code is an exception request. When path control receives an anonymous request and is not be able to pass the response back to the anonymous request sender, path control sets the sense code and appends it to RU data to inform the receiver about the error.

```
            Set LOCAL.SENSE_CODE to the sense data from input MU.
      Else
         If normal-flow request then
            If sequence number from the input MU does not match
             current sequence number (LOCAL.SQN_RCV_CNT + 1) then
             (The current sequence number must be a multiple of 65536.)
               Set LOCAL.SENSE_CODE to X'20010000' (sequence number check).

            If cryptography is active then
               If MU.RH.RU_CTGY = FMD then
                  If RU data is present and LOCAL.SENSE_CODE = X'00000000' then
                     If encryption bit is not set (RH.EDI = ¬ED) then
                        Set LOCAL.SENSE_CODE to X'08090000'.
                     Else
                        If RU data length is not a multiple of 8 then
                           Set LOCAL.SENSE_CODE to X'10010000'.
                           (the maximum RU size must be a multiple of 8)
```

MU_PACING_CHECKS

FUNCTION:	This procedure performs the format checks for pacing responses and checks that the receive pacing counts are acceptable for begin BIU normal-flow requests. This procedure is called only when the BBIUI bit in TH is set to BBIU. The checks in this procedure are required checks.
INPUT:	MU, LOCAL.COMMON_CB
OUTPUT:	LOCAL.SENSE_CODE is set if an error is found; else, remains unchanged

Referenced procedures, FSMs, and data structures:

MU	page A-29
IPM_RU	page 6.2-33
LOCAL	Chapter 6.0

```
If normal-flow request then
   If LOCAL.COMMON_CB.RECEIVE_PACING.RPC = 0 then
      If LOCAL.COMMON_CB.RECEIVE_PACING.NWS = 0 then
         Set LOCAL.SENSE_CODE to X'20110000'.
         (Sending node has overrun the pacing count.)
      Else
         If pacing indicator is set to ¬PAC (PI = ¬PAC) then
            Set LOCAL.SENSE_CODE to X'20110000'.
            (In the first request of a new window, PI should be set to PAC.)
   Else
      If pacing indicator is set to PAC (PI = PAC) then
         Set LOCAL.SENSE_CODE to X'20110002'.
         (In ¬first request in a new window, PI should be set to ¬PAC)
Else (response)
   If adaptive pacing is being used and
    the MU contains a response with pacing indicator set to PAC then
      If this MU is an IPM (RSP, ¬DR1, ¬DR2) then
         If MU.DCF contains a length that is too short for
          an IPM (RH + IPM_RU) then
            Set LOCAL.SENSE_CODE to X'10020000'.
         Else
            If IPM_RU.FORMAT_INDICATOR is not set to FORMAT0 then
               Set LOCAL.SENSE_CODE to X'10010003'.
            Else
               Select, based on IPM_RU.TYPE:
                  When SOLICITED IPM
                     If IPM_RU.NWS = 0  or IPM_RU.RWI = RESET_WINDOW then
                        Set LOCAL.SENSE_CODE to X'10010003'.
                     Else
                        If LOCAL.COMMON_CB.SEND_PACING.NWS > 0 then
                           Set LOCAL.SENSE_CODE to X'20110001'.
                  When UNSOLICITED IPM
                     If IPM_RU.RWI = ¬RESET_WINDOW then
                        Set LOCAL.SENSE_CODE to X'10010003'.
                  When RESET_ACKNOWLEDGEMENT IPM
                     If ¬LOCAL.COMMON_CB.UNSOLICITED_IPM_OUTSTANDING then
                        Set LOCAL.SENSE_CODE to X'20110001'.
                  Otherwise (invalid IPM type)
                     Set LOCAL.SENSE_CODE to X'10010003'.
      Else (this is not an IPM)
         Set LOCAL.SENSE_CODE to X'20110003'.
```

RECEIVE_PACING

FUNCTION:	This procedure updates the receive pacing counts in the LOCAL.COMMON_CB and determines the type of buffer reserve action to request of the buffer manager. This procedure is never called when the residual pacing count and the next-window size are both 0.
INPUT:	MU, LOCAL.COMMON_CB
OUTPUT:	The receive pacing counts and the RESERVE_FLAG in the LOCAL.COMMON_CB may be changed.

Referenced procedures, FSMs, and data structures:

MU	page A-29
LOCAL	Chapter 6.0

```
If pacing indicator in the RH is not set to PAC then
(Current window is not exhausted.)
   Decrement LOCAL.COMMON_CB.RECEIVE_PACING.RPC by 1.
   Set LOCAL.COMMON_CB.RESERVE_FLAG to NO (don't reserve a new window).
Else (this is the first request in a new window)
   Set LOCAL.COMMON_CB.RECEIVE_PACING.RPC to LOCAL.COMMON_CB.RECEIVE_PACING.NWS minus 1.
   Set LOCAL.COMMON_CB.RECEIVE_PACING.NWS to 0.
   If larger window is required (RH.RLWI = RLW) then
      Set LOCAL.COMMON_CB.RESERVE_FLAG to MORE (request a larger new window).
   Else
      Set LOCAL.COMMON_CB.RESERVE_FLAG to ALL (request a new window).
```

SEGMENT_REASSEMBLY

FUNCTION:	This procedure copies normal-flow request MUs from link buffer to dynamic buffer. If segment reassembly is supported, this procedure is called to reassemble segments into a whole BIU.
INPUT:	A pointer to MU (the MU contains a segment that received from path control)
OUTPUT:	When segment reassembly is complete, all segments (for the same BIU) are reassembled and stored in the dynamic buffer. The dynamic buffer address is returned to the calling procedure. If there is an IPM reset acknowledgment outstanding, dynamic buffer pool may be adjusted and ADJUST_FOR_IPM_ACK_OUTSTANDING bit in the LOCAL.COMMON_CB may be changed. If an error is detected, LOCAL.SENSE_CODE is set to indicate the error
NOTES:	1. SM reserves the first buffer for HS. 2. MUs received from path control are stored in link buffers. A link buffer is a special type of permanent buffer that the buffer manager allocates to DLC to send network data between two LUs. The receiving LU calls buffer manager to get a dynamic buffer to copy data from the link buffer to dynamic buffer, and releases the link buffer. The dynamic buffer size is set to maximum RU size by SM at buffer pool create time. (See Appendix B for details.)

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
SM	page 4-48
LOCAL	page 6.0-7
MU	page A-29
PATH CONTROL	<u>T2.1 Node Reference</u>

```
If TH.BBIUI = BBIU then
   If RU length of the input MU is greater than the maximum RU size then
      Set LOCAL.SENSE_CODE to X'10020000' (RU length in error).
   Else
      Call buffer manager (GET_BUFFER, dynamic buffer pool ID, no wait,
       buffer rereserve indication) to get a dynamic buffer for segment reassembly.
       Copy MU data from link buffer to dynamic buffer.  (Appendix B)
   Call buffer manager (FREE_BUFFER, link buffer address) to release the link buffer
    containing the MU.  (Appendix B)
Else (TH.BBIU ≠ BBIU)
   (The segments are reassembled in the same order they are received, i.e., the starting
   offset of the 2nd segment is immediately after the first segment.)
   If ending offset of a segment is greater than the maximum receive RU size than
      Set LOCAL.SENSE_CODE to X'10020000'.
   Else
      Update the MU.DCF after each data copy (from link buffer to dynamic buffer).

   Call buffer manager (FREE_BUFFER, link buffer address) to release
    the link buffer containing the MU.  (Appendix B)

If reassembly is complete (TH.EBIUI = EBIU) then
   Set TH.EBIUI to EBIU.
   (When the first segment was copied from link buffer
   to dynamic buffer, the whole segment [containing TH, RH and RU]
   was copied.  The TH bits setting indicates BBIU and ¬EBIU.  When the
   second segment arrives, only the RU portion was copied to dynamic buffer
   not TH and RH.  If the second segment is the last segment for this BIU,
   after copying the RU this procedure needs to update the TH bits setting to
   indicate EBIU.)

   If LOCAL.COMMON_CB.ADJUST_FOR_IPM_ACK_OUTSTANDING is set to TRUE  then
      Call buffer manager (ADJUST_BUF_POOL, dynamic buffer pool ID, REDUCED
       pool size) to reduce the size of the dynamic buffer pool.
       The REDUCED pool size is sent by the buffer manager via a BUFFERS_RESERVED signal.
       (Appendix B)
      Set LOCAL.COMMON_CB.ADJUST_FOR_IPM_ACK_OUTSTANDING to FALSE.

   Return the address of the dynamic buffer to the calling procedure.
   (The dynamic buffer contains the reassembled whole BIU.)
```

RCV_PACING_RSP

FUNCTION:	This procedure updates pacing counts in the LOCAL.COMMON_CB, sends reset acknowledgments to unsolicited IPMs and sends MUs to path control if possible.
INPUT:	A pointer to the MU; LOCAL.COMMON_CB, the common control block for the pacing routines
OUTPUT:	The pacing response is discarded after being processed and the MU is freed; LOCAL.COMMON_CB, the pacing counts, SET_RLWI, UNSOLICITED_IPM_OUTSTANDING, and ADJUST_FOR_IPM_ACK_OUTSTANDING may be updated; IPM ACK may be sent to path control; MUs from the pacing queue sent to path control; the dynamic buffer pool and the limited buffer pool sizes may be adjusted

Referenced procedures, FSMs, and data structures:

SEND_TO_PC	page 6.2-22
IPM_RU	page 6.2-33
LOCAL	Chapter 6.0
MU	page A-29

```
Select, in the following order, based on the pacing type:
   When adaptive pacing
      Select, based on the type of IPM_RU:
         When reset acknowledgment IPM
            Set LOCAL.COMMON_CB.RECEIVE_PACING.RPC to 0.
            Set LOCAL.COMMON_CB.RECEIVE_PACING.NWS to
             LOCAL.COMMON_CB.UNSOLICITED_NWS (the next window size
             carried in the previous unsolicited IPM).
            Set LOCAL.COMMON_CB.UNSOLICITED_IPM_OUTSTANDING to FALSE.
            Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
             containing the reset acknowledgment IPM.  (Appendix B)
            If segment reassembly is not in progress then
               Call buffer manager (ADJUST_BUF_POOL, dynamic buffer pool ID, REDUCED
                pool size) to adjust the dynamic buffer pool size to the size informed by
                the buffer manager (via a BUFFERS_RESERVED signal).
                (Appendix B)
            Else (segment reassembly is taking place)
               Set LOCAL.COMMON_CB.ADJUST_FOR_IPM_ACK_OUTSTANDING to TRUE .

         When solicited IPM
            If data waiting on the pacing queue (Q_PAC) to be sent then
               If LOCAL.COMMON_CB.SEND_PACING.RPC = 0 then
                (the queued data is waiting for the NWS carried in this IPM)
                  Set LOCAL.COMMON_CB.SET_RLWI to RLW.
                  (Request a larger window on the next pacing request.)
               Call buffer manager (ADJUST_BUF_POOL, limited buffer pool ID, change amount)
                to increase the size of the limited buffer pool based on IPM_RU.NWS.  The
                change amount is IPM_RU.NWS.  (Appendix B)
               Set LOCAL.COMMON_CB.SEND_PACING.NWS to IPM_RU.NWS.
               Call buffer manager (FREE_BUFFER, buffer address) to release the
                buffer containing the solicited IPM.  (Appendix B)
```

```
            When unsolicited IPM
               Call buffer manager (GET_BUFFER, permanent buffer pool ID, no wait)
                to request a permanent buffer to store the unsolicited IPM.
                 (Appendix B)
                  If permanent buffer is not available then
                     Call buffer manager (GET_BUFFER, demand, buffer size, no wait )
                      to request a demand buffer to store the unsolicited IPM
                        If demand buffer is not available then
                           Perform half-session ABEND processing.
               Copy MU data from the link buffer (the unsolicited IPM was in) to
                the permanent (or demand) buffer.
               Call buffer manager (FREE_BUFFER, link buffer address)
                to release the link buffer containing unsolicited IPM.
               Call buffer manager (ADJUST_BUF_POOL, limited buffer pool ID, change amount)
                to reduce the size of the limited buffer pool to no buffers.
                The change amount is a negative value of (NWS plus RPC).
                (Appendix B)
               Set LOCAL.COMMON_CB.SEND_PACING.NWS to IPM_RU.NWS.
               Reset LOCAL.COMMON_CB.SEND_PACING.RPC to 0.
               Set IPM_RU.TYPE to RESET_ACKNOWLEDGMENT.
               Set IPM_RU.RWI to ¬RESET_WINDOW.
               Call SEND_TO_PC (page 6.2-22)
                to send reset acknowledgment IPM to path control (to the partner LU).

      When no pacing
         If ¬DR1,¬DR2 (received MU is a pacing response) then
            Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
             containing the erroneous MU.

      When fixed pacing
         Set LOCAL.COMMON_CB.SEND_PACING.NWS to LOCAL.COMMON_CB.SEND_PACING.FIRST_WS.
         If ¬DR1,¬DR2 (received MU is a pacing response) then
            Call buffer manager (FREE_BUFFER, buffer address) to release the buffer
             containing the pacing response.
         Call buffer manager (ADJUST_BUF_POOL, limited buffer pool ID, change amount)
          to increase the size of the limited buffer pool.  The change amount is
          LOCAL.COMMON_CB.SEND_PACING.FIRST_WS.  (Appendix B)

   If LOCAL.COMMON_CB.SEND_PACING.TYPE ≠ NONE then
      Do the following while the pacing queue is not empty and the first MU is not a BBIU
       or the first MU is a response or
       (LOCAL.COMMON_CB.SEND_PACING.RPC + LOCAL.COMMON_CB.SEND_PACING.NWS) >0.

         Remove the next MU from Q_PAC (pacing queue).
         Call SEND_TO_PC (MU, LOCAL.COMMON_CB) to send the MU just removed from pacing queue.
```

BUFFERS_RESERVED_PROCESSING

FUNCTION:	This procedure receives BUFFERS_RESERVED signals from the buffer manager (BM), updates the appropriate pacing counts in the LOCAL.COMMON_CB, builds and sends the appropriate pacing response. The BUFFERS_RESERVED signal will have a reserve action of REDUCED, REPLENISHED, or RESTART (reserve action of ADJUSTED will never be received by this routine); REDUCED is for an unsolicited IPM, while REPLENISHED and RESTART are for solicited IPMs
INPUT:	BUFFERS_RESERVED signal, LOCAL.COMMON_CB
OUTPUT:	The pacing counts in the LOCAL.COMMON_CB are updated; pacing response is sent to the appropriate path control

Referenced procedures, FSMs, and data structures:

SEND_TO_PC	page 6.2-22
LOCAL	Chapter 6.0
PATH CONTROL	_T2.1 Node Reference_
MU	page A-29
IPM_RU	page 6.2-33

```
If the reserved buffer pool size needs to be reduced (reserve action is assigned) then
   Set LOCAL.COMMON_CB.UNSOLICITED_IPM_OUTSTANDING to TRUE.
   Set LOCAL.COMMON_CB.UNSOLICITED_NWS to the value (as
    received in the BUFFERS_RESERVED signal) that will be sent in the IPM.
   Set MU.DCF for a unsolicited IPM (RH + IPM_RU).
   Set IPM_RU.TYPE to UNSOLICITED.
   Set IPM_RU.RWI to RESET_WINDOW.
   Set IPM_RU.FORMAT_INDICATOR to FORMAT0.
   Set IPM_RU.NWS to CB.UNSOLICITED_NWS.
Else (solicited IPM or IPR)
   Set LOCAL.COMMON_CB.RECEIVE_PACING.NWS to the value that will be sent in the IPM.
   If LOCAL.COMMON_CB.RECEIVE_PACING.TYPE = ADAPTIVE then
      Set MU.DCF for an solicited IPM (RH + IPM_RU).
      Set IPM_RU.TYPE to SOLICITED.
      Set IPM_RU.RWI to ¬RESET_WINDOW.
      Set IPM_RU.FORMAT_INDICATOR to FORMAT0.
      Set IPM_RU.NWS to CB.RECEIVE_PACING.NWS.
   Else (an IPR)
      Set MU.DCF to the length of the RH (IPR contains no RU data).

Set MU.TH bits to indicate BBIU, EBIU.
Set TH.EFI to EXP (this bit can be set to either expedited or normal for IPRs).
Set TH.SNF.SQN to 0.
Set RH bits to indicate the following:
   RSP, FMD, ¬FMH, ¬SD, BC, EC, ¬DR1, ¬DR2, POS, ¬RLW, ¬QR, PAC, ¬BB, ¬EB, ¬CD,
   CODE0, ¬ED, ¬PD, ¬CEB.
Call SEND_TO_PC(MU.LOCAL.COMMON_CB) (page 6.2-22)
 to send a pacing response (IPM or IPR) to path control.
```

TC.DECIPHER_RU

FUNCTION:	Deciphers an enciphered message.
INPUT:	Enciphered MU, session seed
OUTPUT:	Deciphered MU is returned or nonzero sense data is set in LOCAL.SENSE_CODE, and if the MU was padded, pad bytes are dropped and Padded Data indicator set to ¬PD

Referenced procedures, FSMs, and data structures:

HS	page 6.0-3
LOCAL	page 6.0-7
MU	page A-29

```
Decipher the RU data using the DES algorithm.  Use the session key
 as the cryptography key.  Use either the session seed or 0 as the
 initial chaining value.  If DES deciphering fails, the RC is set
 to NG; otherwise, OK.

If DES deciphering RC = NG  then
   Set LOCAL.SENSE_CODE to X'08480000' to indicate cryptography function inoperative.
Else
   If PDI = PD (data was padded) then
      Save the length of RU data (MU.DCF - [the length of RH]).
      Extract the pad count from the last byte of the RU and assign it to PAD_COUNT.
      If (PAD_COUNT < 1) or (PAD_COUNT > 7)  then
         Set LOCAL.SENSE_CODE to X'10010000' to indicate RU data length in error.
      Else
         Decrement MU.DCF by PAD_COUNT (drop pad bytes from RU).
         Set PDI to ¬PD.
```

IPM
This defines the RU format for an IPM

```
IPM_RU
   TYPE:  possible values:  SOLICITED, UNSOLICITED, RESET_ACKNOWLEDGMENT
   RWI:  possible values:  RESET_WINDOW, NOT_RESET_WINDOW
   FORMAT_INDICATOR:  possible value:  FORMAT0
   NWS:  next-window size
```

This page intentionally left blank

APPENDIX A. NODE DATA STRUCTURES

This appendix contains the shared data structures for LU 6.2.

LUCB

The LUCB_LIST contains information about local LUs. One LUCB_LIST exists per node and one LUCB per local LU.

The LUCB_LIST is created at system-definition time. The initial values of the fields in each LUCB entry are implementation-specific.

NOTES:

1. Network-qualified LU names consist of type-A symbol strings. Transaction program names consist of type-AE up through type-GR symbol strings, depending on the implementation. See SNA Formats for symbol-string definitions.

2. If the LU name is not present, the FULLY_QUALIFIED_LU_NAME field is null. Subarea LUs, LUs doing sync point, and LUs using parallel sessions always know their own names.

3. The FULLY_QUALIFIED_LU_NAME contains no trailing space (X'40') characters.

LUCB

Shared Data

LU_ID: identifier of the local LU
FULLY_QUALIFIED_LU_NAME: network-qualified name of the local LU (see Notes)
PARTNER_LU_LIST: list of PARTNER_LU data structures (see page A-2)
TRANSACTION_PROGRAM_LIST: list of TRANSACTION_PROGRAM data structures (see page A-5)
PENDING_RANDOM_DATA_LIST: list of random data (used for LU-LU verification) that has been sent on a BIND to a partner

Data Unique to PS.COPR

LU_SESSION_LIMIT: maximum number of LU-LU sessions the local LU can have

PARTNER_LU

The PARTNER_LU_LIST is a list contained within each LUCB entry. There is one PARTNER_LU_LIST per local LU and one PARTNER_LU entry for each LU name known by a given local LU. Each PARTNER_LU entry contains information that is LU name specific (i.e., information that is constant across all mode names for a given LU name).

The PARTNER_LU_LIST is created at system-definition time. The initial values of the fields in each PARTNER_LU entry are implementation specific.

NOTES:

1. The (partner) LOCAL_LU_NAME is the name that a transaction program specifies in conjunction with the MODE_NAME when requesting the allocation of a conversation. It is a local name by which one local LU knows another (partner) LU and is not sent outside the local LU. The maximum length of the LU_NAME is implementation-defined.

   There may be an entry in the PARTNER_LU_LIST whose LOCAL_LU_NAME is the same as the LU name of this (local) LU. This allows for cases when the partner transaction program is located in the same LU as the local transaction program.

2. Local LU names consist of type-G symbol strings. Fully-qualified LU names consist of type-A symbol strings. See SNA Formats for symbol-string definitions.

3. The (partner) FULLY_QUALIFIED_LU_NAME is the LU name that is sent on external flows, e.g., BIND.

4. The LOCAL_LU_NAME and FULLY_QUALIFIED_LU_NAME fields contain no trailing space (X'40') characters.

PARTNER_LU

Shared Data

LOCAL_LU_NAME: local name of the partner LU (see Notes 1, 2, and 4)
FULLY_QUALIFIED_LU_NAME: network-qualified name of the partner LU (see Notes 2, 3, and 4)
MODE_LIST: list of MODE data structures (see page A-3) used with this partner LU
ACTIVE_SESSION_PARAMETERS:
  PARALLEL_SESSIONS: possible values: SUPPORTED, NOT_SUPPORTED

MODE

The MODE_LIST is a list contained within each PARTNER_LU entry. One MODE entry exists in the MODE_LIST for each mode name that is associated with PARTNER_LU.LOCAL_LU_NAME. Each MODE entry contains mode-name-specific information.

The MODE_LIST is created at system-definition time. The initial values of the fields in each MODE entry are implementation specific.

NOTES:

1. The WAITING_REQUEST_LIST contains requests for sessions sent by PS.CONV ("Chapter 5.1. Presentation Services--Conversation Verbs") that the resources manager cannot presently fulfill, because no free sessions are available. Entries are removed from the list when an existing session becomes free or when a new session is activated.

2. The FREE_SCB_LIST is a list of sessions that are currently not in use by any conversation. The list is an ordered list in that all first-speaker half-sessions are grouped at the front of the list with all bidder half-sessions following. A new first-speaker entry is inserted at the beginning of the list, while a new bidder entry is inserted at the end.

   The FREE_SCB_LIST and the WAITING_REQUEST_LIST are mutually exclusive. An entry in the FREE_SCB_LIST precludes there being an entry in the WAITING_REQUEST_LIST, and vice versa.

3. Mode names consist of type-A symbol strings. See SNA Transaction Programmer's Reference Manual for LU Type 6.2 for a list of valid symbol-string types for the Transaction program name. See SNA Formats for symbol-string definitions.

4. TERMINATION_COUNT is the count of the number of sessions that this local LU is responsible for deactivating. PENDING_TERMINATION_* counts sessions that are pending termination. A session is pending termination from the time that RM ("Chapter 3. LU Resources Manager") sends BIS(RQE1) or BIS(RQE3) to the time that RM sends DEACTIVATE_SESSION or receives SESSION_DEACTIVATED.

5. ACTIVE_*_COUNT counts active sessions. These counts are maintained by RM ("Chapter 3. LU Resources Manager"). A session is active from the time that RM receives SESSION_ACTIVATED or +ACTIVATE_SESSION_RSP to the time that RM sends DEACTIVATE_SESSION or receives SESSION_DEACTIVATED. ACTIVE_*_COUNT includes sessions that are pending termination (see below). ACTIVE_SESSION_COUNT is the sum of ACTIVE_CONWINNERS_COUNT and ACTIVE_CONLOSERS_COUNT.

6. PENDING_*_COUNT counts pending-active sessions. These counts are maintained by RM ("Chapter 3. LU Resources Manager"). A session is pending active from the time that RM sends ACTIVATE_SESSION to the time that RM receives ACTIVATE_SESSION_RSP. PENDING_SESSION_COUNT is the sum of PENDING_CONWINNERS_COUNT and PENDING_CONLOSERS_COUNT.

7. The SESSION_DEACTIVATED_TP is started when an UNBIND is sent or received. The related half-session and any PS that was using the session may still be active when the SESSION_DEACTIVATED_TP actually runs. The SESSION_DEACTIVATED_TP must take this into account.

8. The range of possible maximum RU sizes is delimited at the high end by the values that can be encoded in BIND byte 9 or 10. At the low end, the architectural limit is determined by the BIND encoding, but the implementation limit is determined by the requirement that an FM header fit entirely in one RU. This is to avoid deadlock complications that could occur if an incomplete FMH-5 arrives at the half-session because it is sent spanning more than one RU.

MODE

Shared Data

NAME: mode name (see Note 3)
SESSION_LIMIT: maximum number of sessions allowed for this partner (LU, mode) pair
MIN_CONWINNERS_LIMIT: minimum number of contention winner sessions
MIN_CONLOSERS_LIMIT: minimum number of contention loser sessions
CNOS_NEGOTIATION_IN_PROGRESS: possible values: TRUE, FALSE
LIMIT_BEING_NEGOTIATED: when CNOS negotiation is in progress, the tentative new session limit

ACTIVE_SESSION_COUNT: (see Note 5)
ACTIVE_CONWINNERS_COUNT:
ACTIVE_CONLOSERS_COUNT:

PENDING_SESSION_COUNT: (see Note 6)
PENDING_CONWINNERS_COUNT:
PENDING_CONLOSERS_COUNT:

DRAIN_SELF: possible values: YES, NO
DRAIN_PARTNER: possible values: YES, NO
AUTO_ACTIVATIONS_LIMIT:

Data Unique to Resources Manager

TERMINATION_COUNT: (see Note 4)
PENDING_TERMINATION_CONWINNERS:
PENDING_TERMINATION_CONLOSERS:
SINGLE_SESSION_POLARITY: possible values: FIRST_SPEAKER, BIDDER
LOCAL_MAX_SESSION_LIMIT: maximum MODE session limit value

TRANSACTION_PROGRAM
Each LUCB contains a TRANSACTION_PROGRAM_LIST. This list contains one entry for each transaction program known at the local LU. Each TRANSACTION_PROGRAM entry in the TRANSACTION_PROGRAM_LIST contains information describing one transaction program.
The TRANSACTION_PROGRAM_LIST is created at system-definition time . The initial values of the fields in each TRANSACTION_PROGRAM entry are implementation-defined.
NOTE: See SNA Transaction Programmer's Reference Manual for LU Type 6.2 for a list of valid symbol-string types for the Transaction program name.

TRANSACTION_PROGRAM

Shared Data

TRANSACTION_PROGRAM_NAME: (up to 64 bytes long)
PRIVILEGED_FUNCTIONS_LIST: possible values: ATTACH_SERVICE_TP, CHANGE_NUMBER_OF_SESSIONS, DEFINE_LU_PARAMETERS, DISPLAY_LU_PARAMETERS, SESSION_CONTROL
RESOURCES_SUPPORTED_LIST: possible values: BASIC_CONVERSATION, MAPPED_CONVERSATION
VERIFY_PIP: possible values: YES, NO
NUMBER_OF_PIP_SUBFIELDS: number of PIP subfields required by the TP

Data Unique to RM

SYNC_LEVELS_SUPPORTED_LIST: possible values: NONE, CONFIRM, SYNCPT
INSTANCE_LIMIT: maximum number of TPs that can be brought up (1 is the minimum)
INSTANCE_COUNT: current number of TPs executing (initialized to 0)
STATUS: possible values: ENABLED, DISABLED_TEMPORARY, DISABLED_PERMANENT
WAITING_INITIATION_RQ_LIST: possible values: ATTACH_RECEIVED, START_TP

Data Unique to PS.MC

MC_FUNCTIONS_SUPPORTED_LIST: possible values: MAPPING, FMH_DATA

RCB

The RCB_LIST contains information about resources. There is one RCB_LIST per local LU and one RCB per resource known by that LU. The RCB_LIST is managed by RM ("Chapter 3. LU Resources Manager"). Entries are added to, and deleted from, the RCB_LIST by the resources manager. The RCB_LIST is also referenced by presentation services, e.g., PS.CONV ("Chapter 5.1. Presentation Services--Conversation Verbs"). The RCB_LIST contains entries for all the resources associated with all the transaction program instances active at a particular LU.

NOTES:

1. The (partner) LU_NAME is the name that a transaction program specifies in conjunction with the MODE_NAME when requesting the allocation of a conversation. It is a local name by which one local LU knows another (partner) LU and is not sent outside the local LU. The maximum length of the LU_NAME is implementation-defined.

2. LU names consist of type-G symbol strings. Mode names consist of type-A symbol strings. See SNA Formats for symbol string definitions.

3. When the resources manager receives a GET_SESSION (page A-16) from PS.CONV and determines that only a bidder half-session is available (i.e., all first-speaker half-sessions are in use), it has to request permission to use the half-session. Because permission may be denied, SESSION_PARMS_PTR points to the GET_SESSION record while the request for permission to use the session is outstanding. If permission is denied, the GET_SESSION record is used to issue a new request for a session. After permission has been granted, or if a first-speaker session can be allocated, SESSION_PARMS_PTR has a value of NULL.

4. Used to purge Attaches from the TRANSACTION_PROGRAM.WAITING_INITIATION_RQ_LIST when HS terminates.

RCB

Shared Data

RCB_ID: ID of this RCB
TCB_ID: ID of the transaction that owns this RCB
HS_ID: ID of the half-session associated with this RCB
SESSION_IDENTIFIER: session ID assigned to the conversation
LU_NAME: partner LU name (see Notes 1 and 2)
MODE_NAME: (see Note 2)
CONVERSATION_CORRELATOR: (see Note 2)
BRACKET_ID: unique value generated by RM to identify all records for a given conversation.
SYNC_LEVEL: possible values: NONE, CONFIRM, SYNCPT
SECURITY_SELECT: possible values: NONE, SAME, PGM

Data Unique to RM

SESSION_PARMS_PTR: (see Note 3)
TP_NAME: TP name sent on ALLOCATE or received on Attach (see Note 4)

Data Unique to PS.CONV

CONVERSATION_TYPE: possible values: BASIC_CONVERSATION, MAPPED_CONVERSATION
LIMITED_BUFFER_POOL_ID: buffer pool ID
PERMANENT_BUFFER_POOL_ID: buffer pool ID
POST_CONDITIONS:
   FILL: possible values: BUFFER, LL
   MAX_LENGTH: maximum number of bytes in incoming logical record or buffer
LOCKS: possible values: SHORT, LONG
SEND_RU_SIZE: maximum number of bytes for outgoing MU record
RQ_TO_SEND_RCVD: possible values: YES, NO
HS_TO_PS_BUFFER_LIST: list of MU data structures (see page A-29)

Data Unique to PS.MC

MC_RECEIVE_BUFFER: contains RECEIVED_INFO (see page A-7)
MAPPER_SAVE_AREA: contains information used in mapping
MC_MAX_SEND_SIZE: maximum number of bytes in a mapped-conversation logical record
MC_RQ_TO_SEND_RCVD: possible values: YES, NO

RECEIVED_INFO
RECEIVED_INFO is the structure that is inserted into the MC_RECEIVE_BUFFER_LIST. The MC_RECEIVE_BUFFER_LIST is contained within an RCB and consists of information received by PS.MC ("Chapter 5.2. Presentation Services--Mapped Conversation Verbs") but not yet passed to the transaction program.

RECEIVED_INFO
   TYPE: possible values: MAP_NAME, MAP_NAME_AND_DATA_CONTINUED,
    DATA_CONTINUED, MAPPED_DATA, INDICATOR, RC

SCB

There is one SCB per half-session. SCBs are maintained by the resources manager.

NOTES: 1. The (partner) LU_NAME is the name that a transaction program specifies in conjunction with the MODE_NAME when requesting the allocation of a conversation. It is a local name by which one local LU knows another (partner) LU and is not sent outside the local LU. The maximum length of the LU_NAME is implementation-defined.

2. LU names consist of type-G symbol strings. Network-qualified LU names and mode names consist of type-A symbol strings. See SNA Formats for symbol-string definitions.

SCB

Data Unique to RM

HS_ID: unique SCB identifier
SESSION_IDENTIFIER: ID used to identify session in BIND
LU_NAME: partner LU name (see Notes)
MODE_NAME: mode name (see Note 2)
RCB_ID: ID of RCB representing the conversation that is using this session; null if no conversation is using this session
FIRST_SPEAKER: possible values: YES, NO
SEND_RU_SIZE: maximum number of bytes for an outgoing MU record
LIMITED_BUFFER_POOL_ID: buffer pool ID
PERMANENT_BUFFER_POOL_ID: buffer pool ID
BRACKET_ID: unique value generated by RM to identify all records for a given conversation.
RTR_OWED: possible values: TRUE, FALSE
FULLY_QUALIFIED_LU_NAME: partner network-qualified LU name (see Note 2)
RANDOM_DATA: used to validate FMH-12

TCB

The TCB_LIST contains information about active transaction program instances. There is one TCB_LIST per local LU and one TCB per active transaction program instance running at that LU. The TCB_LIST is managed by RM ("Chapter 3. LU Resources Manager"). Entries are added to and deleted from the TCB_LIST by the resources manager. The TCB_LIST is also referenced by presentation services, e.g., PS.CONV ("Chapter 5.1. Presentation Services--Conversation Verbs").

Each TCB contains an embedded RESOURCES_LIST, which contains one (pointer) entry for each resource associated with a particular transaction program instance.

NOTES: 1. See SNA Transaction Programmer's Reference Manual for LU Type 6.2 for a list of valid symbol-string types for the Transaction program name and Access Security Information.

2. Each entry in the RESOURCES_LIST has a corresponding entry in the RCB_LIST. The RCB_LIST contains entries for all the resources associated with all the transaction program instances running at the LU. In contrast, the RESOURCES_LIST contains entries for only those resources associated with a particular transaction program instance.

TCB

Shared Data used by RM and PS. Initialized by RM.

```
TCB_ID:  identifies the PS process
TRANSACTION_PROGRAM_NAME:  (see Note 1)
OWN_LU_ID:
LUW_IDENTIFIER
   FULLY_QUALIFIED_LU_NAME:
   LUW_INSTANCE:
   LUW_SEQUENCE_NUMBER:
RESOURCES_LIST:  (see Note 2)
CONTROLLING_COMPONENT:  possible values:  TP, SERVICE_COMPONENT
INITIATING_SECURITY: initiating security information is received on the Attach
 that started this TP (see Note 1)
   PROFILE:
   USERID:
```

ABORT_HS

ABORT_HS indicates to the session manager that the half-session has found a severe error and cannot continue processing. This will cause an UNBIND to be sent for the aborted half-session.

```
ABORT_HS
   HS_ID:  identifies the half-session sending this record
   SENSE_CODE:  indicates the reason the half-session aborted
```

INIT_HS_RSP
This record is a response to the INIT_HS record that was sent from the session manager to the half-session to initialize the half-session. The response indicates whether or not the initialization was successful (POS) or not (NEG). When NEG, the reason is indicated by the sense data in SENSE_CODE.

INIT_HS_RSP
TYPE: possible values: POS, NEG
SENSE_CODE: indicating the type of error (reserved when TYPE=POS)
HS_ID: identifies the half-session sending the record

CONFIRMED
CONFIRMED is sent by the half-session to PS_CONV to inform PS_CONV that a positive response to the previous request for confirmation has been received.

CONFIRMED
BRACKET_ID: unique value generated by RM to identify all records for a given conversation.

RECEIVE_ERROR
RECEIVE_ERROR is sent by the half-session to PS_CONV to inform PS_CONV that a -RSP(0846) has been received.

RECEIVE_ERROR
BRACKET_ID: unique value generated by RM to identify all records for a given conversation.

REQUEST_TO_SEND
REQUEST_TO_SEND is sent by the half-session to PS_CONV to inform PS_CONV that the transaction program at the partner LU has requested to enter the send state for the conversation.

REQUEST_TO_SEND
BRACKET_ID: unique value generated by RM to identify all records for a given conversation.

RSP_TO_REQUEST_TO_SEND
RSP_TO_REQUEST_TO_SEND is sent by the half-session to PS_CONV to inform PS_CONV that the response to the previous MU (with PS_TO_HS.PS_TO_HS_VARIANT=REQUEST_TO_SEND -- page A-29) has been received.

RSP_TO_REQUEST_TO_SEND
  BRACKET_ID: unique value generated by RM to identify all records for a given conversation.

BID
BID is sent by the half-session to the resources manager to inform the resources manager that the partner LU has requested permission to use the half-session for a conversation. The resources manager replies with a BID_RSP record (page A-11). The half-session sends a BID record to the resources manager even if the partner LU is the first-speaker.

BID
  HS_ID: identifies the half-session sending this record

BID_RSP
BID_RSP is sent by the half-session to the resources manager to inform the resources manager of the partner LU's response to the local LU's request to use the session (see BID_WITHOUT_ATTACH [page A-17]). BID_RSP is sent by the half-session only if the local LU is the bidder. If RTI = NEG, SENSE_CODE contains the sense data carried on the negative response. BID_RSP is also sent by the resources manager to the half-session in response to a previous BID record (page A-11) from the half-session. If RTI = POS, the partner LU is granted permission to use the session. If RTI = NEG, permission is denied and SENSE_CODE contains the sense data to be sent on the negative response.

BID_RSP
  HS_ID: identifies the half-session sending this record
  RTI: type of response—possible values: POS, NEG
  SENSE_CODE: indicates the type of error (reserved when RTI=POS)

BIS_RQ
BIS_RQ is sent by the half-session to the resources manager to inform the resources manager that a BIS(RQE1) request unit was received. BIS_RQ is also sent by the resources manager to the half-session to request the half session to send a BIS(RQE1) request unit.

BIS_RQ
HS_ID: identifies the half-session sending this record

BIS_REPLY
BIS_REPLY is sent by the half-session to the resources manager to inform the resources manager that a BIS(RQE3) request unit was received. BIS_REPLY is also sent by the resources manager to the half-session to request the half-session to send a BIS(RQE3) request unit.

BIS_REPLY
HS_ID: identifies the half-session sending this record

FREE_SESSION
FREE_SESSION is sent by the half-session to the resources manager to inform the resources manager that the half-session has become free (i.e., not in use by a conversation).

FREE_SESSION
HS_ID: identifies the half-session sending this record (the half-session that has become free)

RTR_RQ
RTR_RQ is sent by the half-session to the resources manager to inform the resources manager that an RTR request unit was received. RTR_RQ is also sent by the resources manager to the half-session to request the half-session to send an RTR request unit.

RTR_RQ
HS_ID: identifies the half-session sending this record

RTR_RSP
RTR_RSP is sent by the half-session to the resources manager to inform the resources manager that an RTR response unit was received. If RTI = NEG, SENSE_CODE contains the sense data carried on the negative response. RTR_RSP is also sent by the resources manager to the half-session to request the half-session to send an RTR response unit. If RTI = NEG, SENSE_CODE contains the sense data to be sent with the negative response.

RTR_RSP
  HS_ID: identifies the half-session sending this record
  RTI type of response: possible values: POS, NEG
  SENSE_CODE: indicates the type of error (reserved when RTI=POS)

INIT_HS
This record contains the information necessary for the half-session to initialize itself. It is sent when a successful session activation occurs and contains information from the BIND RU.

INIT_HS
  PATH_CONTROL_ID: identifies the path control the half-session communicates with
  TYPE of half-session: possible values: PRI, SEC
  DYNAMIC_BUFFER_POOL_ID: buffer pool ID
  LIMITED_BUFFER_POOL_ID: buffer pool ID
  LSFID: see page A-28
  TRANSMISSION_PRIORITY: possible values: LOW, MEDIUM, HIGH, NETWORK
  SHORT_BIND_IMAGE: the first 26 bytes of the negotiated BIND image
   (see BIND in SNA Formats)

ACTIVATE_SESSION_RSP
ACTIVATE_SESSION_RSP is sent by the session manager to the resources manager in reply to an ACTIVATE_SESSION record (page A-20). ACTIVATE_SESSION_RSP records need not be sent in the same order as the ACTIVATE_SESSION records, so CORRELATOR is used to correlate the ACTIVATE_SESSION_RSP to the ACTIVATE_SESSION. If TYPE = POS (a session was activated), SESSION_INFORMATION specifies session characteristics. If TYPE = NEG (a session was not activated), ERROR_TYPE contains a retry/no-retry indication.

ACTIVATE_SESSION_RSP
  CORRELATOR: as supplied in ACTIVATE_SESSION (see page A-20)
  TYPE of response: possible values: POS, NEG
  SESSION_INFORMATION: (reserved when TYPE=NEG--see page A-32)
  ERROR_TYPE: possible values: RETRY, NO_RETRY (reserved when TYPE=POS)

SESSION_ACTIVATED
SESSION_ACTIVATED is sent by the session manager to the resources manager to notify the resources manager that the partner LU named by LU_NAME and MODE_NAME has activated a session to this LU. The characteristics of the session are specified in SESSION_INFORMATION.
NOTES: 1. The (partner) LU_NAME is the name that a transaction program specifies in conjunction with MODE_NAME when requesting the allocation of a conversation. It is a local name by which one local LU knows another (partner) LU and is not sent outside the local LU. The maximum length of the LU_NAME is implementation-defined.
2. LU names consist of type-G symbol strings. Mode names consist of type-A symbol strings. See <u>SNA Formats</u> for symbol-string definitions.

```
SESSION_ACTIVATED
   SESSION_INFORMATION:  (see page A-32)
   LU_NAME:  (see Notes 1 and 2)
   MODE_NAME:  (see Note 2)
```

SESSION_DEACTIVATED
SESSION_DEACTIVATED is sent by the session manager to the resources manager to notify the resources manager that the session identified by HS_ID has been deactivated. It is also used internally in the resources manager.

```
SESSION_DEACTIVATED
   HS_ID:  identifies the half-session that was deactivated
   REASON for deactivation:  possible values:  NORMAL, ABNORMAL_RETRY, ABNORMAL_NO_RETRY
   SENSE_CODE: provides additional information on
               session deactivation (reserved when REASON = NORMAL).
```

SEND_ERROR
SEND_ERROR is sent by PS_CONV to the half-session to request the half-session to send a -RSP(0846).

```
SEND_ERROR
```

ALLOCATE_RCB

ALLOCATE_RCB is sent by PS.CONV to the resources manager to request creation and initialization of a resource control block. The resources manager also attempts to reserve a first-speaker session if IMMEDIATE_SESSION = YES. The resources manager replies to the ALLOCATE_RCB with an RCB_ALLOCATED record (page A-21).

NOTES:
1. The (partner) LU_NAME is the name that a transaction program specifies in conjunction with the MODE_NAME when requesting the allocation of a conversation. It is a local name by which one local LU knows another (partner) LU and is not sent outside the local LU. The maximum length of the LU_NAME is implementation-defined.
2. LU names consist of type-G symbol strings. Mode names consist of type-A symbol strings. See SNA Formats for symbol-string definitions.

```
ALLOCATE_RCB
   TCB_ID:  ID of PS process that sent ALLOCATE_RCB
   LU_NAME:  (see Notes 1 and 2)
   MODE_NAME:  (see Note 2)
   IMMEDIATE_SESSION:  possible values:  YES, NO
   SYNC_LEVEL:  possible values:  NONE, CONFIRM, SYNCPT
   SECURITY_SELECT:  possible values:  NONE, SAME, PGM
```

CHANGE_SESSIONS

CHANGE_SESSIONS is sent by PS.COPR to the resources manager to inform the resources manager of a change in the session limits for (LU_NAME, MODE_NAME). PS.COPR changes the session limits in the MODE control block (page A-3) before sending this record to the resources manager. RESPONSIBLE = YES if this LU is responsible for deactivating sessions to satisfy the new session limits. DELTA contains the (signed) difference between the current MODE.SESSION_LIMIT and the previous MODE.SESSION_LIMIT.

NOTES:
1. The (partner) LU_NAME is the name that a transaction program specifies in conjunction with the MODE_NAME when requesting the allocation of a conversation. It is a local name by which one local LU knows another (partner) LU and is not sent outside the local LU. The maximum length of the LU_NAME is implementation-defined.
2. LU names consist of type-G symbol strings. Mode names consist of type-A symbol strings. See SNA Formats for symbol string definitions.

```
CHANGE_SESSIONS
   TCB_ID:  ID of the PS process that sent CHANGE_SESSIONS
   RESPONSIBLE:  possible values:  YES, NO
   LU_NAME:  (see Notes 1 and 2)
   MODE_NAME:  (see Note 2)
   DELTA:  change in MODE.SESSION_LIMIT
```

DEALLOCATE_RCB

DEALLOCATE_RCB is sent by PS.CONV to the resources manager to request destruction of the resource control block identified by RCB_ID. The resources manager replies to the DEALLOCATE_RCB with an RCB_DEALLOCATED record (page A-21).

```
DEALLOCATE_RCB
   TCB_ID:  ID of the PS process that sent DEALLOCATE_RCB
   RCB_ID:  ID of the RCB to deallocate
```

GET_SESSION

GET_SESSION is sent by PS.CONV to the resources manager to request the allocation of a session to the conversation identified by RCB_ID. The resources manager replies to the GET_SESSION with a SESSION_ALLOCATED record (page A-22).

```
GET_SESSION
   TCB_ID:  ID of the PS process that sent GET_SESSION
   RCB_ID:  ID of the conversation
```

RM_ACTIVATE_SESSION

RM_ACTIVATE_SESSION is sent by PS.COPR to the resources manager to request activation of a new session with the partner LU identified by LU_NAME on the mode name identified by MODE_NAME. This record is sent as a result of the ACTIVATE_SESSION control operator verb.

NOTES:

1. The (partner) LU_NAME is the name that a transaction program specifies in conjunction with the MODE_NAME when requesting the allocation of a conversation. It is a local name by which one local LU knows another (partner) LU and is not sent outside the local LU. The maximum length of the LU_NAME is implementation-defined.
2. LU names consist of type-G symbol strings. Mode names consist of type-A symbol strings. See <u>SNA Formats</u> for symbol-string definitions.

```
RM_ACTIVATE_SESSION
   TCB_ID:  ID of the PS process that sent RM_ACTIVATE_SESSION
   LU_NAME:  (see Notes 1 and 2)
   MODE_NAME:  (see Note 2)
```

RM_DEACTIVATE_SESSION
RM_DEACTIVATE_SESSION is sent by PS.COPR to the resources manager to request deactivation of the session identified by SESSION_ID. This record is sent as a result of the DEACTIVATE_SESSION control-operator verb.

RM_DEACTIVATE_SESSION
 TCB_ID: ID of the PS process that sent RM_DEACTIVATE_SESSION
 SESSION_ID: identifies the session
 TYPE: possible values: NORMAL, CLEANUP

TERMINATE_PS
TERMINATE_PS is sent by PS_INITIALIZE to the resources manager to request termination of the process that comprises presentation services and the transaction program.

TERMINATE_PS
 TCB_ID: ID of the PS process to be terminated

UNBIND_PROTOCOL_ERROR
UNBIND_PROTOCOL_ERROR is sent by PS_CONV or PS_INITIALIZE to the resources manager to request abnormal termination of the session identified by HS_ID. The record is sent when the partner LU commits a serious protocol error. The sense data to be carried on the UNBIND is in SENSE_CODE.

UNBIND_PROTOCOL_ERROR
 TCB_ID: ID of the PS process that sent UNBIND_PROTOCOL_ERROR
 HS_ID: ID of the half-session to be deactivated
 SENSE_CODE:

BID_WITHOUT_ATTACH
BID_WITHOUT_ATTACH is sent by the resources manager to the half-session to request permission (from the partner LU) to use the session. The request for permission is not accompanied by any other data. The resources manager sends BID_WITHOUT_ATTACH only if this LU is the bidder, since it does not need permission from the partner LU to use a first-speaker session. The half-session informs the resources manager of the partner LU's response with a BID_RSP record (page A-11).

BID_WITHOUT_ATTACH

BRACKET_FREED
BRACKET_FREED is sent by the resources manager to the half-session to inform the half-session that it may purge records from PS with a matching BRACKET_ID. This signal is sent after PS has sent DEALLOCATE_RCB for the bracket.

BRACKET_FREED
  BRACKET_ID: unique value generated by RM to identify all records for a given conversation.

ENCIPHERED_RD2
ENCIPHERED_RD2 is sent by RM to the half-session to request the half-session to send an FMH-12.

ENCIPHERED_RD2
  SEND_PARM: (see page A-32)

HS_PS_CONNECTED
HS_PS_CONNECTED is sent by the resources manager to the half-session to inform the half-session that it has been connected to a presentation services process. This occurs as a result of the allocation of a session to a conversation.

HS_PS_CONNECTED
  PS_ID: ID of a presentation services process
  BRACKET_ID: unique value generated by RM to identify all records for a given conversation.

RM_HS_CONNECTED
RM_HS_CONNECTED is sent by the resources manager to the half-session to inform the half-session that it may forward incoming data (e.g., FMH-5s) to RM. This occurs after the session manager has informed RM that HS has been successfully initialized.

RM_HS_CONNECTED

YIELD_SESSION

YIELD_SESSION is sent by the resources manager to the half-session to end the open bracket in a newly activated session. When a session is activated, the session comes up in the "in-bracket" state, with the primary LU in control. If the resources manager at the primary LU does not have a waiting session-allocation request (see GET_SESSION, page A-16), it sends YIELD_SESSION to the half-session; the half-session then reverts to contention state.

```
YIELD_SESSION
```

START_TP

START_TP is sent from a local node component to the resources manager to request initiation of a transaction program. It is also sent from the resources manager to presentation services during the initiation processing.

NOTES:

1. The local LU name is the name by which the receiver of the START_TP knows the originator of the START_TP. This field is used for security "come-from" checking of START_TP records. If come-from checking is not required by the target transaction program, this field is not referenced by the receiver.
2. The network-qualified LU name is specified by the issuer of START_TP. The name specified uniquely identifies an LU. If the receiver of the START_TP has a network-qualified LU name, the name specified is always network-qualified. This LU name is used to generate logical-unit-of-work identifiers (LUW IDs). The non-LU 6.2 request originator never generates its own LUW IDs. The LUW ID is used for sync point conversations, network problem determination, and network accounting functions. Node component procedures that are able to send START_TP records to RM are considered privileged, protected processes with content security (i.e., integrity). These procedures may supply the network-qualified LU name of the requester or an Already-Verified indication for security (i.e., a user ID indicated as already verified, thus eliminating the need for a password).

```
START_TP
  REPLY:  possible values:  YES, NO
  TARGET_TP_NAME:  name of the TP to be started
  SECURITY_SELECT:  possible values:  NONE, PGM, ALREADY_VERIFIED
  SECURITY (reserved when SECURITY_SELECT=NONE)
    PROFILE:
    PASSWORD:
    USER_ID:
  TCB_ID:  transaction program instance identifier
  PIP_LIST:  list of PIP_DATA
  FULLY_QUALIFIED_LU_NAME:  initiator network-qualified LU name
```

START_TP_REPLY
START_TP_REPLY is sent by the resources manager to the local node component that sent the START_TP record.

```
START_TP_REPLY
   RESPONSE_CODE:  possible values:  OK, PIP_NOT_ALLOWED, PIP_NOT_SPECIFIED_CORRECTLY,
    TPN_NOT_RECOGNIZED, TRANS_PGM_NOT_AVAILABLE_RETRY, INVALID_FULLY_QUALIFIED_LU_NAME,
    PS_CREATION_FAILURE, TRANS_PGM_NOT_AVAIL_NO_RETRY, SECURITY_NOT_VALID
   TCB_ID:  transaction program instance identifier
```

SEND_RTR
SEND_RTR is sent to the resources manager to prompt the resources manager to send an RTR request on the session specified by HS_ID.

```
SEND_RTR
```

ACTIVATE_SESSION
ACTIVATE_SESSION is sent by the resources manager to the session manager to request the activation of a session of type SESSION_TYPE with the partner LU identified by LU_NAME and mode name identified by MODE_NAME. Session manager replies to ACTIVATE_SESSION with an ACTIVATE_SESSION_RSP record (page A-13) that has the same CORRELATOR value as that in the ACTIVATE_SESSION. NOTES: 1. The (partner) LU_NAME is the name that a transaction program specifies in conjunction with the MODE_NAME when requesting the allocation of a conversation. It is a local name by which one local LU knows another (partner) LU and is not sent outside the local LU. The maximum length of the LU_NAME is implementation-defined. 2. LU names consist of type-G symbol strings. Mode names consist of type-A symbol strings. See <u>SNA Formats</u> for symbol-string definitions.

```
ACTIVATE_SESSION
   CORRELATOR:
   SESSION_TYPE:  possible values:  FIRST_SPEAKER, BIDDER
   LU_NAME:  (see Notes 1 and 2)
   MODE_NAME:  (see Note 2)
```

DEACTIVATE_SESSION
DEACTIVATE_SESSION is sent by the resources manager to session manager to request the deactivation of a session. If STATUS = ACTIVE, the session is identified by HS_ID. If STATUS = PENDING, the session is identified by CORRELATOR, which contains the same value used in the ACTIVATE_SESSION request.

```
DEACTIVATE_SESSION
   STATUS:  possible values:  ACTIVE, PENDING
   CORRELATOR:  (reserved when STATUS=ACTIVE)
   HS_ID:  (reserved when STATUS = PENDING)
   TYPE of deactivation:  possible values:  NORMAL, CLEANUP, ABNORMAL
    (CLEANUP or ABNORMAL imply STATUS=ACTIVE)
   SENSE_CODE:  reason for deactivation
```

CONVERSATION_FAILURE
CONVERSATION_FAILURE is sent by the resources manager to PS_CONV to notify presentation services of the failure of the conversation identified by RCB_ID.

```
CONVERSATION_FAILURE
   RCB_ID:  ID of failed conversation
   REASON:  possible values:  SON, PROTOCOL_VIOLATION
```

RCB_ALLOCATED
RCB_ALLOCATED is sent by the resources manager to PS_CONV in reply to an ALLOCATE_RCB (page A-15). RETURN_CODE indicates the success of the allocation. If RETURN_CODE = OK, RCB_ID contains the ID of the newly created resource control block.

```
RCB_ALLOCATED
   RETURN_CODE:  possible values:  OK, UNSUCCESSFUL, SYNC_LEVEL_NOT_SUPPORTED
   RCB_ID:  ID of newly created resource control block (reserved when RETURN_CODE≠OK)
   SEND_RU_SIZE:  maximum size of an RU on this session
   LIMITED_BUFFER_POOL_ID:  buffer pool ID
   PERMANENT_BUFFER_POOL_ID:  buffer pool ID
```

RCB_DEALLOCATED
RCB_DEALLOCATED is sent by the resources manager to PS_CONV in reply to a DEALLOCATE_RCB record (page A-16).

```
RCB_DEALLOCATED
```

RM_SESSION_ACTIVATED
RM_SESSION_ACTIVATED is sent by the resources manager to PS_COPR in reply to an RM_ACTIVATE_SESSION record (page A-16). The success or failure of the session activation is indicated in the RETURN_CODE field.

RM_SESSION_ACTIVATED
   RETURN_CODE: possible values: OK, ACTIVATION_FAILURE_NO_RETRY,
    ACTIVATION_FAILURE_RETRY, LU_MODE_SESSION_LIMIT_EXCEEDED

SESSION_ALLOCATED
SESSION_ALLOCATED is sent by the resources manager to PS_CONV in reply to a GET_SESSION record (page A-16). RETURN_CODE indicates the success or failure of the session allocation.

SESSION_ALLOCATED
   SEND_RU_SIZE: maximum size or an RU on this session
   LIMITED_BUFFER_POOL_ID: buffer pool ID
   PERMANENT_BUFFER_POOL_ID: buffer pool ID
   IN_CONVERSATION: possible values: YES, NO
   RETURN_CODE: possible values: OK, UNSUCCESSFUL_RETRY, UNSUCCESSFUL_NO_RETRY,
    SYNC_LEVEL_NOT_SUPPORTED

ASSIGN_PCID
SM sends this message to SS to request the assignment of a FQPCID. The FQPCID that is assigned is returned in an ASSIGN_PCID_RSP message. If the DUPLICATE_PCID field in the ASSIGN_PCID message has a value of YES, it indicates that the last FQPCID assigned by SS to the requesting SM had the same value as another FQPCID already being used by that SM. This can occur when two nodes in the network have the same CP names.

ASSIGN_PCID
   SM_PROCESS_ID: requesting SM process ID
   DUPLICATE_PCID: possible values: YES, NO

ASSIGN_PCID_RSP
SS returns this message in response to an ASSIGN_PCID message, received from SM. It contains the FQPCID that is assigned.

ASSIGN_PCID_RSP
   FQPCID: fully qualified procedure correlator identifier

INIT_SIGNAL_NEG_RSP
SS sends this message in response to an INIT_SIGNAL message received from SM. It is sent when the session initiation request in the INIT_SIGNAL message cannot be satisfied.

INIT_SIGNAL_NEG_RSP
   FQPCID: fully qualified procedure correlator identifier

CINIT_SIGNAL
SS sends this message in response to an INIT_SIGNAL message received from SM. It contains the information that SM needs to build and send a BIND.

CINIT_SIGNAL
   FQPCID: fully qualified procedure correlator identifier
   PATH_CONTROL_ID: session path control ID
   PC_CHARACTERISTICS: see page A-32
   COS_TPF_PRESENT: possible values: YES, NO
   COS_TPF: see SNA Formats for format

INIT_SIGNAL
SM sends this message to SS in order to obtain the information necessary to build and send a BIND. SS sends either an INIT_SIGNAL_NEG_RSP message or a CINIT_SIGNAL message in response.

INIT_SIGNAL
   SM_PROCESS_ID: requesting SM process identifier
   FQPCID: fully qualified procedure correlator identifier
   SLU_NAME: secondary LU name
   PLU_NAME: primary LU name
   MODE_NAME: mode name

SESSST_SIGNAL
SM sends this message to SS whenever it successfully processes a received BIND.

SESSST_SIGNAL
PATH_CONTROL_ID: path control identifier

SESSEND_SIGNAL
SM sends this message to SS whenever a session is terminated. The SENSE_CODE field has a nonzero value only when the session terminated abnormally.

SESSEND_SIGNAL
SENSE_CODE: nonzero when session terminated abnormally
FQPCID: fully qualified procedure correlator identifier
PATH_CONTROL_ID: path control identifier

PC_HS_DISCONNECT
PC_HS_DISCONNECT instructs path control to detach the half-session.

PC_HS_DISCONNECT
PATH_CONTROL_ID: path control identifier
LFSID: see page A-28

SESSION_ROUTE_INOP
SESSION_ROUTE_INOP informs all the SMs that a particular route (identified by PATH_CONTROL_ID) is lost and that all sessions (active and pending) using that route must be taken down.

SESSION_ROUTE_INOP
PATH_CONTROL_ID: path control identifier

ABEND_NOTIFICATION
   ABENDING_PROCESS: name of the abending process
   PROCESS_ID: ID of the abending process
   REASON: cause of abend

ASSIGN_LFSID
ASSIGN_LFSID requests the procurement of a local LFSID from the specified address space. The LFSID is to be associated with the specified SM. The assigned LFSID is returned to SM in the ASSIGN_LFSID_RSP signal.

ASSIGN_LFSID
   PATH_CONTROL_ID: path control identifier
   SM_PROCESS_ID: SM identity for the LFSID
   PROCESS_ID_TYPE: possible values: SM
   SENSE_CODE:
   LFSID: see page A-28

FREE_LFSID
FREE_LFSID requests that a previously assigned LFSID be freed (marked not in-use). No response is returned.

FREE_LFSID
   PATH_CONTROL_ID: path control identifier
   LFSID: ID to be freed--see page A-28

LFSID_IN_USE_RSP
The LFSID_IN_USE_RSP signal resolves the question of whether or not another node has tried to use a LFSID already marked in-use by ASM. The LFSID_IN_USE signal has chased any work queued to the SM, and by the time ASM gets this LFSID_IN_USE_RSP, the question of duplicate LFSID usage has been correctly ascertained (any UNBIND processing that was in progress has been completed).

LFSID_IN_USE_RSP
   PATH_CONTROL_ID: path contol identifier
   LFSID: see page A-28
   ANSWER: possible values: YES, NO

ASSIGN_LFSID_RSP
ASM sends this message in response to an ASSIGN_LFSID message, requesting the assignment of an LFSID to a particular SM. It contains the LFSID that was assigned. If the SENSE_CODE field has a nonzero value, it indicates that an LFSID was not assigned.

```
ASSIGN_LFSID_RSP
   PATH_CONTROL_ID:  path control identifier
   SM_PROCESS_ID:  SM identity for the LFSID
   PROCESS_ID_TYPE:  possible values:  SM
   SENSE_CODE:
   LFSID:  see page A-28
```

LFSID_IN_USE
The LFSID_IN_USE signal asks the SM if an LFSID is still in-use. A BIND was received with an LFSID that appears to be still in-use. However, the node on the other end of the link may just be faster. This signal will chase any work queued to the SM, and by the time ASM gets the LFSID_IN_USE_RSP, the question of duplicate LFSID usage can be correctly ascertained.

```
LFSID_IN_USE
   PATH_CONTROL_ID:  path control identifier
   LFSID:  see page A-28
   ANSWER:  possible values:  YES, NO
```

HS_CREATE_PARMS
This signal contains the data that HS requires to initialize itself.

```
HS_CREATE_PARMS
   LU_ID:  LU, RM, and SM process ID
   HS_ID:  ID of the newly created half-session
```

PS_CREATE_PARMS
This signal contains the data that PS requires to initialize itself.

PS_CREATE_PARMS
  LUCB_LIST_PTR: pointer to the LUCB_LIST
  LU_ID: ID of this PS's LU and RM process
  TCB_LIST_PTR: pointer to the TCB_LIST
  TCB_ID: ID of this PS process
  RCB_LIST_PTR: pointer to the RCB_LIST

RM_CREATE_PARMS
This signal contains the data that RM requires to initialize itself.

RM_CREATE_PARMS
  LUCB_LIST_PTR: pointer to the LUCB_LIST
  LU_ID: LU process ID

SM_CREATE_PARMS
This signal contains the data that SM requires to initialize itself.

SM_CREATE_PARMS
  LU_ID: LU process ID
  LUCB_LIST_PTR: pointer to the list of LUCBs

RM_CREATED
  LU_ID: ID of the created RM

CRV_RQ_RU

CRV_RQ_RU
  RQ_CODE: possible values: X'C0' (signifying CRV)
  CRYPTO_SEED: enciphered transform of test value from RSP(BIND)

LFSID
Local-Form Session Identifier (LFSID) associated with a half-session.

```
LFSID
   SESSION_ID:
      SIDH:  high-order byte of session identifier
      SIDL:  low-order byte of session identifier
   ODAI:
```

MU

A message unit (MU) is an interprocess signal that always contains a PIU (TH-RH-RU). The MU contains two types of PIUs:

- session traffic: regular data flow
- nonsession traffic: BIND, ±RSP(BIND), UNBIND, and ±RSP(UNBIND)

The MU contains a header that varies depending upon the type of PIU.

```
MU
   HEADER_TYPE:  possible values:  BIND_RQ_SEND, BIND_RSP_SEND, UNBIND_RQ_SEND,
    UNBIND_RQ_RCV, UNBIND_RSP_SEND, BIND_RQ_RCV, BIND_RSP_RCV, HS_TO_RM, RM_TO_PS,
    HS_TO_PS, PS_TO_HS
```

Header type BIND_RQ_SEND fields.

```
   BIND_RQ_SEND
      LU_ID:  LU identifier
      SENDER:
         ID:  sending process ID
         TYPE:  possible values:  SM
      HS_ID:  half-session ID for the session
      TRANSMISSION_PRIORITY:  possible values:  LOW, MEDIUM, HIGH, NETWORK
      LFSID:  see page A-28
      PATH_CONTROL_ID:  path control identifier
      PARALLEL_SESSIONS:  parallel session support indicator
      ADAPTIVE_PACING:  adaptive pacing support indicator
```

Header type BIND_RSP_SEND fields.

```
   BIND_RSP_SEND
      SENDER:
         ID:  sending process ID
         TYPE:  possible values:  SM
      LFSID:  see page A-28
      HS_ID:  half-session ID for the session
      TRANSMISSION_PRIORITY:  possible values:  LOW, MEDIUM, HIGH, NETWORK
      PATH_CONTROL_ID:  path control identifier
      FREE_LFSID:
      PARALLEL_SESSIONS:  parallel session support indicator
      ADAPTIVE_PACING:  adaptive pacing support indicator
```

Header type UNBIND_RQ_SEND fields.

```
   UNBIND_RQ_SEND
      SENDER:
         ID:  sending process ID
         TYPE:  possible values:  SM
      LFSID:  see page A-28
      HS_ID:
      TRANSMISSION_PRIORITY:  possible values:  LOW, MEDIUM, HIGH, NETWORK
      FREE_LFSID:  possible values:  YES, NO
      PATH_CONTROL_ID:  path control identifier
      PARALLEL_SESSIONS:  parallel session support indicator
      ADAPTIVE_PACING:  adaptive pacing support indicator
```

Header type UNBIND_RQ_RCV fields.

```
UNBIND_RQ_RCV
   SENDER:
      ID:  sending process ID
      TYPE:  possible values:  SM
   LFSID:  see page A-28
   HS_ID:
   TRANSMISSION_PRIORITY:  possible values:  LOW, MEDIUM, HIGH, NETWORK
   FREE_LFSID:  possible values:  YES, NO
   PATH_CONTROL_ID:  path control identifier
   PARALLEL_SESSIONS:  parallel session support indicator
   ADAPTIVE_PACING:  adaptive pacing support indicator
```

Header type UNBIND_RSP_SEND fields.

```
UNBIND_RSP_SEND
   LU_ID:
   SENDER:
      ID:  sending process ID
      TYPE:  possible values:  SM
   HS_ID:  half-session ID for the session
   TRANSMISSION_PRIORITY:  possible values:  LOW, MEDIUM, HIGH, NETWORK
   FREE_LFSID:  possible values:  YES, NO
   PATH_CONTROL_ID:  path control identifier
   PARALLEL_SESSIONS:  parallel session support indicator
   ADAPTIVE_PACING:  adaptive pacing support indicator
```

Header type BIND_RQ_RCV fields.

```
BIND_RQ_RCV
   SENDER:
      ID:  sending process ID
      TYPE:  possible values:  SM
   LFSID:  see page A-28
   HS_ID:  half-session ID for the session
   TRANSMISSION_PRIORITY:  possible values:  LOW, MEDIUM, HIGH, NETWORK
   FREE_LFSID:  possible values:  YES, NO
   PATH_CONTROL_ID:  path control identifier
   PC_CHARACTERISTICS:  see page A-32
   PARALLEL_SESSIONS:  parallel session support indicator
   ADAPTIVE_PACING:  adaptive pacing support indicator
```

Header type BIND_RSP_RCV fields.

```
BIND_RSP_RCV
   SENDER:
      ID:  sending process ID
      TYPE:  possible values:  SM
   LFSID:  see page A-28
   HS_ID:  half-session ID for the session
   TRANSMISSION_PRIORITY:  possible values:  LOW, MEDIUM, HIGH, NETWORK
   FREE_LFSID:  possible values:  YES, NO
   PATH_CONTROL_ID:  path control identifier
   PARALLEL_SESSIONS:  parallel session support indicator
   ADAPTIVE_PACING:  adaptive pacing support indicator
```

Header type HS_TO_RM fields.

HS_TO_RM
   HS_ID: identifies the half-session that sent the record to RM

Header type RM_TO_PS fields.

RM_TO_PS
   HS_ID: half-session ID associated with the conversation
   TCB_ID: PS id
   RCB_ID: conversation ID
   SEND_RU_SIZE: maximum number of bytes for outgoing MU record
   LIMITED_BUFFER_POOL_ID: buffer pool ID
   PERMANENT_BUFFER_POOL_ID: buffer pool ID
   RETURN_CODE: result of the RM checks of the Attach

Header type HS_TO_PS fields.

HS_TO_PS
   BRACKET_ID: unique value generated by RM to identify all records for a given conversation.
   FMH: possible values: YES, NO
   TYPE: possible values: NOT_END_OF_DATA, CONFIRM, PREPARE_TO_RCV_CONFIRM, PREPARE_TO_RCV_FLUSH, DEALLOCATE_CONFIRM, DEALLOCATE_FLUSH

Header type PS_TO_HS fields.

PS_TO_HS
   BRACKET_ID: unique value generated by RM to identify all records for a given conversation.
   PS_TO_HS_VARIANT: possible values: CONFIRMED, REQUEST_TO_SEND, SEND_DATA_RECORD, SEND_ERROR
   ALLOCATE: possible values: YES, NO
   FMH: possible values: YES, NO
   TYPE: possible values: CONFIRM, DEALLOCATE_CONFIRM, DEALLOCATE_FLUSH, FLUSH, PREPARE_TO_RECEIVE_FLUSH, PREPARE_TO_RECEIVE_CONFIRM_SHORT, PREPARE_TO_RECEIVE_CONFIRM_LONG

Fields present on all header types.

DCF: data count field, length of RH and data fields
BTU:
   PIU:
      TH: see SNA Formats for additional information
      BIU: basic information unit
         RH: see SNA Formats for additional information
            RU: data being sent or received

PC_CHARACTERISTICS.
Defines the characteristics of path control.

PC_CHARACTERISTICS
  MAX_SEND_BTU_SIZE: maximum basic transmission unit send size
  MAX_RCV_BTU_SIZE: maximum basic transmission unit receive size
  ADJACENT_NODE_BIND_REASSEMBLY: possible values: SUPPORTED, NOT_SUPPORTED

SEND_PARM
SEND_PARM is a substructure that is embedded in ENCIPHERED_RD2 (page A-18). It contains the data to be sent to the half-session as well as an encoding of the RH bit-settings. If ALLOCATE = YES, this data is the first to be sent on a conversation. If FMH = YES, DATA begins with an FM header (FMH-5 or FMH-7).

SEND_PARM
  ALLOCATE: possible values: YES, NO (if ALLOCATE=YES, DATA is first in bracket)
  FMH: possible values: YES, NO (if FMH=YES, DATA begins with FM header)
  TYPE: possible values: NOT_END_OF_DATA, FLUSH, CONFIRM, DEALLOCATE_CONFIRM, DEALLOCATE_FLUSH, PREPARE_TO_RCV_FLUSH, PREPARE_TO_RCV_CONFIRM_SHORT, PREPARE_TO_RCV_CONFIRM_LONG
  DATA: data to be sent on the session

SESSION_INFORMATION
SESSION_INFORMATION is a substructure that is embedded in SESSION_ACTIVATED (page A-14) and ACTIVATE_SESSION_RSP (page A-13). Sent from the session manager to the resources manager, SESSION_INFORMATION contains data about the session that has just been activated.

SESSION_INFORMATION
  HS_ID: half-session identifier
  HALF_SESSION_TYPE: possible values: PRI, SEC
  BRACKET_TYPE: possible values: FIRST_SPEAKER, BIDDER
  SEND_RU_SIZE: maximum send RU size for this session
  LIMITED_BUFFER_POOL_ID: buffer pool ID
  PERMANENT_BUFFER_POOL_ID: buffer pool ID
  SESSION_IDENTIFIER: unique (for this LU) 8-byte session identifier
  RANDOM_DATA: used to validate FMH-12

SNF

This data structure defines the Sequence Number field in the TH.

SNF: a 16-bit sequence number field.
   SQN: a 16-bit sequence number whose value wraps to 0 after 65535.
      BRACKET_STARTED_BY: possible values are PRI (1) or SEC (0).
       The high-order bit of the sequence number field is set when the bracket is started by the primary half-session and reset when the bracket is started by the secondary half-session. This is done so that sequence numbers on BB requests are unique.
      NUMBER: a 15-bit sequence number whose value wraps to 0 after 32767.

This page intentionally left blank

APPENDIX B. BUFFER MANAGER

INTRODUCTION

Each node has a buffer manager (BM), which controls the buffers used for the storage of data that flows to and from the network. This appendix describes the buffer manager functions, services, and protocol boundary with LU components. (Other node components that interact with BM generally are not discussed in this book.)

BM provides the following functions:

1. Controls allocation and deallocation of storage used for buffers

2. Limits allocation of buffers when available storage runs low.

3. Allows components to reserve storage in buffer pools to guarantee availability of enough buffers to complete distinct tasks.

4. Allows processes to suspend processing pending availability of storage for buffers.

Two pacing indicators are used by BM.

1. Pacing indicator (PI): A pacing request is a normal-flow request that has the PI bit (in the RH) set to PAC. When it is set, it indicates that an RU is the first RU in the last window granted permission to be sent, and a new send window is requested. Otherwise, the PI bit is set to ¬PAC, indicating a new send window is not needed.

2. Request Larger Window indicator (RLWI): The RLWI field in the RH is used by the sending side to indicate to the receiving side that it would like to have a larger window (i.e., larger than the current window). The RLWI bit has meaning only in a pacing request and is reserved in a response. When the HS (send side) receives a solicited IPM, the send residual pacing count gets updated (see Figure B-4 on page B-16 and Figure B-5 on page B-17 as an example). If the send pacing queue has more requests to be sent (i.e., more than the current send pacing window), the HS (send side) sends the next pacing request with RLWI set (to RLW) to request a larger window. Otherwise, the RLWI is set to ¬RLW, indicating a larger window is not needed.

When a new pacing window is needed, HS sends a normal-flow request to its partner HS; this request is flagged with PAC (and RLW, if a larger window is requested). When this normal-flow request is received by HS, HS requests a buffer from a dynamic buffer pool to store this normal-flow request and later sends it to PS. HS (on the receive side) passes the PI and RLWI values to BM when issuing the GET_BUFFER call, to inform BM that a new send window (and larger window, if RLW is set) has been requested by HS on the send side. (See Figure B-2 on page B-12, Figure B-3 on page B-13, and the GET_BUFFER call in this appendix for more details.) BM then replenishes the pool when the same buffer is subsequently freed.

TYPES OF BUFFERS

BM provides five different types of buffers, four of which it organizes in buffer pools:

- Demand buffers
- Limited buffers
- Permanent buffers
- Fixed dynamic buffers
- Varying dynamic buffers

All except demand buffers are in pools.

A buffer pool is a set of buffers with the same characteristics (e.g., size, use, owner, pool identification). BM creates a buffer pool upon request from SM, but HS is designated the owner of the buffer pool. After SM requests that BM create a buffer pool, LU components can get buffers from the created buffer pool, free buffers that they have removed from the buffer pool, adjust the size of the buffer pool, return (release) buffers to BM, and destroy a buffer pool or a specified number (1 to all) of the buffers in a buffer pool. (Destroying a buffer means removing the buffer from the pool and making it unavailable to the LU components from that pool. Destroying the buffer pool means making the pool itself no longer available to the LU components.)

For three of the four types of buffer pools (permanent, fixed dynamic, and varying dynamic), BM reserves buffers for the associated buffer pool when it creates the buffer pool. Creation of these three pools involves setting the size of each buffer in the pool (to some fixed value) and the number of buffers initially reserved for the pool in accordance with parameters in the CREATE_BUF_POOL used to create the pool, e.g., for adaptive pacing, 1 buffer, and, for fixed pacing, the number of buffers in the pacing window (negotiated during session initiation).

BM allocates storage for demand and limited buffers individually at GET_BUFFER time.

The following sections describe the different types of buffers and pools provided by BM.

- DEMAND BUFFERS

  Demand buffers are not reserved ahead of time and are not associated with a pool. Demand buffers are requested when requests for other buffer cannot be met. For example, normally, HS requests a permanent buffer to send expedited-flow requests (i.e., a CRV) and PS will request a permanent buffer to send normal-flow and expedited-flow responses. If no permanent buffer is available, they will request a demand buffer instead. When BM receives a request for a demand buffer and sufficient storage exists, BM allocates a demand buffer to the buffer requester. If the buffer is not available, and the buffer requester is willing to wait (as specified on the GET_BUFFER call), BM queues the request (and the requesting process is suspended) until a demand buffer is available; otherwise, the request is rejected.

- LIMITED BUFFER POOLS

  SM sets up a limited buffer pool for the session when it is created. SM creates the pool, designating HS as its owner, and initializes its limit count to the number of buffers needed for the first send pacing window. Buffers "belonging" to a limited buffer pool are allocated on demand like demand buffers; the difference is that the limit count can be used to limit those allocated for a given purpose, as represented by the associated limited buffer pool ID.

  SM passes the limited buffer pool ID to HS. When HS receives a pacing response from its pacing partner with a new send pacing window size, it adjusts the limited pool limit count to include the new send pacing window size.

  RM passes the limited buffer pool identifier to PS after receiving it from SM. PS uses a limited buffer to send a normal-flow request to HS. If the limited pool is empty (i.e., the limit count is 0), PS waits until permission to send a new pacing window is received for the session and HS adjusts the pool. This keeps PS from flooding HS with normal-flow requests that cannot be sent.

  The limited pool is set up to allow limited-buffer requests (from PS). When PS requests a limited buffer and the send residual pacing count (which the limit count represents) has gone to 0, the request is queued until HS adjusts the send residual pacing count to a value greater than 0, which occurs when HS get a pacing response from its partner HS.

- PERMANENT BUFFER POOL

  The unique characteristic of the permanent buffer pool is that, after a buffer is gotten from a permanent buffer pool, it is returned to that pool when it is freed. If all the buffers in the pool are in use, no more buffers can be taken from the pool until one of the buffers is freed. After a permanent buffer pool is created, the number of buffers associated with it is adjusted, by SM, when HS instances are created (increased) or destroyed (decreased).

  Link buffers are a special kind of permanent buffer with additional restrictions. When a DLC is created, it is assigned a pool of link buffers. When a BTU is received from the link connection, it is placed in a link buffer from the assigned pool and sent to path control. When the buffer is freed, it is returned to the pool assigned to the DLC (the same as other permanent buffers). Link buffers cannot be sent to a process that waits for an external event to occur (for this could potentially cause a deadlock). The process always has to be able to run, process the link buffer, and free it without waiting on any outside event to occur. Received link buffers are not reused for responses or acknowledgments, since the DLC may delay sending them. link buffers are used within the session layers whenever they will be quickly freed.

  The LU uses other permanent buffers for sending responses, expedited-flow requests, and IPM acknowledgments. A node component may call BM to get a permanent buffer first; if the buffer is not available, it may then ask for a demand buffer instead. In such a case, if the demand buffer is not available either, the calling process (if it does not wait) ends abnormally.

- FIXED AND VARYING DYNAMIC BUFFER POOLS

  Fixed and varying dynamic buffers are used for session-level fixed and adaptive pacing, respectively. HS uses dynamic buffers to store normal-flow requests that it receives from PC in link buffers.

When a normal-flow request is received, a dynamic buffer is taken from the pool and the request is copied to this buffer by HS.

For adaptive pacing, when the buffer flagged PAC is freed by PS, BM will then (when sufficient storage exists) send a BUFFERS_RESERVED signal to HS indicating how many new buffers have been reserved for its pacing partner to fill. Based on the BUFFERS_RESERVED signal, HS builds and sends a pacing response (i.e., IPM) to its pacing partner, informing it that it now has permission to send another pacing window. (An IPM carries the next-window size, corresponding to the number of buffers reserved by BM, which reflects the availability of storage and the RLWI setting BM detected in the freed buffer carrying PAC.) The pacing partner updates its send residual pacing count upon receiving the pacing response. (See "Chapter 6.2. Transmission Control" for details.)

The new window size reserved by BM for adaptive pacing can change from one FREE_BUFFER time to the next. At FREE_BUFFER time, if the buffer freed is flagged with PAC and RLW, it actually notifies BM to not only allocate a new window but also a bigger one than the current one. Depending upon the availability of dynamic buffers, BM may delay the reservation of the new window, it may decrease the size of the new window, it may maintain the size of the new window, or, if buffers are available and the buffer freed is flagged with PAC and RLW, it may increase the size of the new window. When BM runs critically short of varying dynamic buffers, it notifies HS via the REDUCED version of BUFFERS_RESERVED. This causes TC to reset to 0 the size of the current window by sending an unsolicited pacing message (see "Chapter 6.2. Transmission Control" in Chapter 6.2 for details).

For fixed pacing, when the buffer flagged with PAC is freed by PS, BM sends a BUFFERS_RESERVED signal to HS indicating that enough dynamic buffers are reserved to receive another window from its partner HS. The size of the window is fixed during session initiation. When a shortage of fixed dynamic buffers exists, BM delays sending BUFFERS_RESERVED to HS until enough buffers are again available. Having received the BUFFERS_RESERVED signal, HS builds and sends a pacing response to its partner HS informing it that it now has a grant for another send window.

For either adaptive or fixed pacing, the partner HS updates its send residual pacing count after receiving the pacing response and increases the number of buffers in its limited buffer pool by the size of the new (varying or fixed) pacing window (see "Chapter 6.2. Transmission Control").

When a dynamic buffer is freed, it is not returned to the pool it was taken from; instead, BM makes it available for reallocating wherever it is then needed.

Figure B-1 summarizes the different types of session-managed data and the usage of buffers.

Buffer Usage	Component Requesting Buffers	Buffer Type Required
Sending normal-flow data requests: PS uses limited buffers to send normal-flow data requests to HS, and HS passes them on to other processes.	PS	limited (Note 1)
Sending expedited-flow requests: HS uses permanent buffers to send CRV. PS uses permanent buffers to send SIGNAL RUs to HS for forwarding to the partner LU.	HS, PS	permanent (Notes 2, 3)
Sending responses: All buffers for sending responses come from the permanent buffer pool (e.g., IPM, IPR, IPM_ACK, RTR_RSP, SIGNAL_RSP, SEND_ERROR, CONFIRMED)	HS, PS	permanent (Notes 2, 3)
Sending normal-flow DFC requests: Special normal-flow requests (i.e., BIS, RTR, and LUSTAT) are sent using permanent buffers. When each such request needs to be sent out and limited buffers are not available, HS cannot wait for a buffer; hence, HS uses a permanent buffer to send the record instead of a limited buffer. However, the limited buffer pool size is adjusted by HS to reflect the correct send residual pacing count.	HS	permanent (Note 2)
Receiving normal-flow requests: HS uses a fixed or varying dynamic buffer to hold the received normal-flow request received from PC so that the link buffer can be released.	HS	dynamic

NOTES:

1. If a limited buffer is not available, PS suspends processing and waits for one unless the limited buffer pool has been destroyed. (A race condition may have occurred in which HS was destroyed and all buffer pools HS owned were also destroyed.) In the absence of the limited buffer pool, PS requests a demand buffer instead.

2. If no permanent buffer is available, a demand buffer is requested instead. If that fails, the calling process ends abnormally (which causes an UNBIND to flow).

3. Upon receipt, these RUs remain in the link buffers from DLC until they are processed and freed, or HS transfers them to local (not paced) storage that is not managed by the buffer manager.

Figure B-1. Send/Receive Buffer Usage (for Session Data)

BUFFER MANAGER PROTOCOL BOUNDARY

LU components (i.e., RM, SM, PS, HS) and BM communicate using the following protocol boundary.

LU TO BM

The following signals are included as parameters in synchronous (Call) invocations of BM:

- ADJUST_BUF_POOL
- CREATE_BUF_POOL
- DESTROY_BUF_POOL
- FREE_BUFFER
- GET_BUFFER

With each call to BM, other appropriate parameters are also passed. When BM adds buffers to a fixed or varying dynamic pool as part of a FREE_BUFFER or ADJUST_BUF_POOL action, BM notifies HS by sending a BUFFERS_RESERVED signal, indicating that the number of buffers available in the pool has changed. If the HS instance no longer exists at FREE_BUFFER time, BM does not send the BUFFERS_RESERVED signal.

Following are detailed descriptions of the LU-BM calls.

ADJUST BUF POOL

The ADJUST_BUF_POOL call to BM is a request to change the capacity of an existing pool, or to change the number of buffers in a pool without changing the pool capacity. The new capacity can be specified to be larger or smaller than the current pool capacity, to be restored to its original capacity (at pool creation time), or (for varying dynamic pools) to reduce the capacity to the amount requested by BM to conserve system dynamic storage.

ADJUST_BUF_POOL can also request immediate replenishment action in order to bring the number of buffers available in the pool up to the current pool capacity.

An ADJUST_BUF_POOL call to BM contains the following parameters:

- Pool ID
- The number of buffers to be added to or subtracted from the pool capacity (for permanent pools) or the limit count (for limited pools). If buffer resources are limited and cannot be allocated to the process at ADJUST_BUF_POOL time, the ADJUST_BUF_POOL Call fails with no change to the number of buffers associated with the pool.
- An indication to reset the number of buffers in the pool (and capacity) to the initial value set by CREATE_BUF_POOL, and canceling any pending replenishment action triggered by a previous GET_BUFFER specifying PAC, but whose buffer has not yet been freed; or to leave the current pool size (for fixed and varying dynamic pools) unchanged, but to perform a pending replenishment action immediately (rather than at the later FREE_BUFFER time); or (for varying dynamic pools) an indication that the pool size may be reduced to the size specified by BM in its previous BUFFERS_RESERVED signal, e.g., because an IPM acknowledgment has since been received.

CREATE BUF POOL

The CREATE_BUF_POOL call to BM requests BM to create a buffer pool of a specified type: permanent, fixed dynamic, varying dynamic, or limited. For permanent, fixed, and varying pools, BM creates the specified number and size of buffers, and holds them until they are called for (see GET_BUFFER call). For limited pools, the buffers are created (allocated) at GET_BUFFER time.

The BUFFERS_RESERVED signal is not sent for this initial reserve action.

When a process is destroyed for any reason, all buffer pools owned by that process are automatically unreserved. If a buffer pool is being reserved by one process for another process, the latter process already exists (has already been created) when the CREATE_BUF_POOL call is made.

A CREATE_BUF_POOL call to the BM contains the following parameters:

- Pool type (permanent, fixed, varying, or limited).
- The capacity (pool size) of the buffer pool. This pool size does not represent the number of buffers in the pool at any one time but rather the capacity of the pool to hold buffers (analogous to the size of a swimming pool versus the amount of water in it).
  - For permanent and fixed pools, it is the number of buffers initially put in the pool.
  - For fixed pools, this is also the number of buffers that BM adds to the pool when replenishing it.
  - For varying pools, this parameter is a performance-related "target," not a minimum or maximum pool size.
  - For limited pools, this pool size is used to set an initial value in a limit count field, N. If N = 0, BM suspends the caller, if the Wait option on the GET_BUFFER call was specified, until it can supply the buffer; otherwise, the GET_BUFFER call fails to return a buffer. Each GET_BUFFER call decrements N by 1. An ADJUST_BUF_POOL call to BM is used to increment or decrement N. The running value of N is equal to the current send residual pacing count.
  - For limited pools, "no-limit" may be specified, in which case BM does not limit the number of buffers that can be associated with, and thus obtained from, the limited pool. Any ADJUST_BUF_POOL call for the limited pool is ignored.
- The size of each buffer (e.g., 256 bytes) to be reserved by the BM for the pool, where all buffers in a pool are the same size.
- For varying pools, an initial number (or "increment") of reserved buffers (default of 1) in the pool. (The model described in this book sets this to 1.) A multiple of this increment value is used at pool replenishment time.
- The pool owner, which becomes the recipient of the BUFFERS_RESERVED signal (from BM). The specified owning process instance exists at reserve time. The pool is deleted when the owning process issues a DESTROY_BUF_POOL call (with All specified).

BM sets a return code indicating whether the buffer pool was created successfully or not (e.g., buffer storage not available); if successful, BM returns the pool ID.

DESTROY_BUF_POOL

The DESTROY_BUF_POOL call to the BM requests that a specified buffer pool or all buffer pools owned by the caller be destroyed.

A DESTROY_BUF_POOL call contains the following parameters:

- A specific pool ID, or an indication that the request applies to all owned pools.

FREE BUFFER

The FREE_BUFFER call to BM notifies BM that the specified buffer is no longer required and needs to be released. A process does not have to "own" the pool to free a buffer from it. Any process with addressability to the buffer can free it. For example,

- If the buffer is a demand buffer, it is returned to general system storage (destroyed).
- If the buffer is a permanent buffer, it is immediately returned to its permanent pool (rereserved).
- If the buffer is from a fixed or varying dynamic pool, it is returned to the system. When the buffer being freed is one that held a pacing request, BM adds a number of buffers to the pool from which the freed buffer was obtained, where the number is the size of the new pacing window that can be received. BM informs the process (e.g., HS) owning the pool of the new buffers by sending it a BUFFERS_RESERVED signal.
- If the buffer is a limited buffer, its freeing does not increment the limit count being maintained for its limited pool. (The value of the limit count is incremented only with the ADJUST_BUF_POOL call to the BM.)

A FREE_BUFFER call to BM contains a pointer to the buffer that needs to be freed. This pointer was returned from a previous GET_BUFFER call to BM.

GET BUFFER

The GET_BUFFER call to BM requests one or more buffers for use. The calling process becomes the owner of the buffer. However, HS is the owner of all buffer pools used by the LU.

A GET_BUFFER call to BM contains the following parameters:

- A pool ID, returned from a previous CREATE_BUF_POOL call, which is used to identify the pool from which the buffer is to be allocated; or a demand buffer request (including buffer size needed).
- For fixed and varying dynamic pools, information specifying the Pacing (PAC or ¬PAC) and Reserve Larger Window (RLW or ¬RLW) indicator values in the MU for which the buffer is to be used. If PAC is specified, it means that when the buffer is later freed, BM should reserve buffers for the next pacing window and add them to the pool. For fixed pools, if the required number of buffers cannot be created at free-buffer time, the requested replenishment action is delayed until they are available. For varying pools, the number of buffers BM adds to the pool depends on multiple factors: the availability of general system storage, the status (held or freed) of buffers previously obtained from the pool, the setting of RLWI, and the value of the increment (I) specified in the call creating the pool. When general system storage is low, BM may reduce the number of new buffers added to the pool by a multiple of I, to allow immediate replenishment of the pool. When sufficient storage exists, BM adds C buffers to the pool (C = current window size) if RLWI = ¬RLW, or C + nI buffers (where n is some positive integer) if RLWI = RLW. When BM does replenish a pool following the subsequent freeing of the corresponding buffer, it sends a BUFFERS_RESERVED signal to the pool owner.
- An indication ("wait" or "no wait") whether or not the calling process may be suspended pending availability of a buffer to honor the request. When the process may not be suspended, BM returns an appropriate return code to indicate that the pool is empty or that a demand buffer cannot be created.

When the requested buffer is available, BM returns the buffer pointer to the calling process.

BM TO LU

The BUFFERS_RESERVED signal is sent asynchronously from BM to HS. A detailed description follows.

BUFFERS RESERVED

The BUFFERS_RESERVED signal is sent by BM to a dynamic buffer pool owner (e.g., HS) to report a change, other than as a result of a GET_BUFFER, in the number of buffers available in a dynamic buffer pool. The BUFFERS_RESERVED signal contains the following parameters:

- The pool ID
- The number of buffers added to the pool if this signal follows a FREE_BUFFER of a pacing request buffer, or to be subtracted from the pool if a REDUCED action is being reported (see below)
- The number of buffers currently in the pool (available for GET_BUFFERs)
- The type of action being reported:
  - ADJUSTED: An ADJUST_BUF_POOL request has been honored.
  - REPLENISHED: A dynamic buffer pool has been replenished as a result of a FREE_BUFFER of a pacing request buffer.
  - REDUCED: BM has detected a critical scarcity of system storage and will reduce the number of buffers assigned to the varying dynamic pool (possibly to 0) when the pool owner issues an ADJUST_BUF_POOL (following exchange of unsolicited and reset acknowledgment IPMs with the partner LU).
  - RESTART: BM has changed the number of buffers in the varying dynamic pool from 0 to a positive value (i.e., the value of the increment specified in the CREATE_BUF_POOL for the pool); this signal follows the REDUCED (to 0) type BUFFERS_RESERVED signal when node congestion eases.

The BUFFERS_RESERVED message is of sufficient size that it can itself be reused to send an IPR or IPM. If this message is not reused to send an IPR or IPM, it is freed by the message receiver.

```
┌────────────────────────────────────────────────────────────────────┐
│            LU(A)                                           Node A  │
│      ┌──────────────────────┐                                      │
│ TP   LU.PS               LU.HS           BM          PC       DLC  │
└────────────────────────────────────────────────────────────────────┘
  SEND_DATA                  .             .           .          .
  (80 bytes)   GET_BUFFER(limited pool ID, wait)       .          .
1 ────────>o- - - - - - - - - - - - - - - >o           .          .
          |(80 bytes copied into the limited buffer)   .          .
2         └■■■■┴─────────┘RU1(size=200)    .           .          .
  SEND_DATA                  .             .           .          .
  (80 bytes)                 .             .           .          .
3 ────────>o                 .             .           .          .
          |(80) (80)         .             .           .          .
4         └■■■■┴■■■■┴───┘RU1(RU size=200)  .           .          .
  SEND_DATA                  .             .           .          .
  (80 bytes)                 .             .           .          .
5 ────────>o                 .             .           .          .
          |(80) (80) (40) .                .           .          .
6         ├■■■■┴■■■■┴■■┘RU1(RU size=200)   .           .          .
          |MU(SEND_DATA_RECORD)  MU(SEND_DATA_RECORD, PAC, RLW)   .
7         ├──────────────>o───────────────────────────>o          .
          |               .          GET_BUFFER(no wait, |        .
          |               .           demand buffer size)|(100 bytes)  .
8         |GET_BUFFER(limited pool ID,    o<- - - - - - - o-■■■■■┘(BTU size=100)
          |             wait)             .            |          .
9         └- - - - - - - - - - - - - - - >o            |  data segment #1
10        |(40)           .               .            └──────────>o──>
11        └■■──┴─────────┘RU2         FREE_BUFFER(pointer to demand buffer)|
12        .              (RU size=200)    o<- - - - - - - - - - - - - - - ┘
          .               .          GET_BUFFER(no wait, .                .
          .               .           demand buffer size). (100)          .
13        .               .               o<- - - - - - - o-■■■■■┘(BTU size=100)
          .               .               .            |                  .
          .               .               .            |                  .
          .               .               .            |                  .
          .               .               .            |  data segment #2
14        .         FREE_BUFFER(pointer to limited buffer)├──────────>o──>
15        .               .               o<- - - - - - - ┘           |
          .               .          FREE_BUFFER(pointer to demand buffer)|
16        .               .               o<- - - - - - - - - - - - - - - ┘
          .               .               .            .                  .
```

Notes:

1. The signal names (e.g., GET_BUFFER) above the dashed lines refer to parameters (identifying request types) passed in Call invocations of the buffer manager; the additional parameters in parentheses are also passed to BM.

2. The symbol "■■■■" denotes the data stored in the buffer.

Figure B-2. LU Interactions with BM When Sending Data

```
 Node B                                              LU(B)
                                              ┌───────────────────┐
 DLC              PC              BM         LU.HS              LU.PS       TP

  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
  .                .               .           .                  .          .
data segment #1   . MU (PAC, RLW)  .           .                  .          .
──>o──────────────>o──────────────────────────>o                  .          .10
  .                .  GET_BUFFER(dynamic pool ID,│                .          .
  .                .            no wait, PAC, RLW)│               .          .
  .                .               o<- - - - - - ┤                .          .11
  .                .               .  (Copy record from link      .          .
  .                .               .   buffer to dynamic buffer.) .          .
  .                .               o<- - - - - - -├■■■■■┘ seg #1  .          .12
  .                .  FREE_BUFFER(pointer to link buffer)         .          .
  .                .               o<- - - - - - ┘                .          .13
data segment #2   . MU             .           . seg #2  seg #1.             .
──>o──────────────>o──────────────────────────>o─■■■■■─┴─■■■■■┘.  data       .14
  .         FREE_BUFFER(pointer to link buffer)├─────────────────>o──────────>o15
  .                .               o<- - - - - - ┘                │          .16
  .                .          FREE_BUFFER(pointer to dynamic buffer)         .
  .                .               o<- - - - - - - - - - - - - - -┘          .17
  .                .               .           .                  .          .
  .                .              BUFFERS_RESERVED(REPLENISHED,   .          .
  .                .               total number in pool, pool ID, .          .
  .                .               number of new buffers reserved).          .
  .                .               o──────────>o                  .          .18
  .                .               .           .                  .          .
```

Figure B-3. LU Interactions with BM When Receiving Data

The comments below correspond to the numbers in Figure B-2 on page B-12 and Figure B-3 on page B-13.

1. In node A, a transaction program (TP) issues a SEND_DATA verb; this causes PS to call BM to request a limited buffer to store the data from TP. If a limited buffer is not available at this time, PS will request a demand buffer instead (see Chapter 5.1). The amount of data that PS can copy to its limited buffer depends on the size of the limited buffer, which cannot be greater then the maximum RU size. (The session manager [SM] previously created, via a CREATE_BUF_POOL Call to BM, a limited buffer pool for the session components to send normal-flow requests, and set the buffer size to the maximum RU size obtained from the BIND.) In this example, the maximum RU size is 200 bytes.

2. In node A, PS copies the data from the TP storage to the PS-owned limited buffer obtained from the limited buffer pool. (In this example, 80 bytes have been sent by the TP.) PS copies the 80 bytes of data to its limited buffer but does not send it until the whole buffer is full.

3. The TP requests more data be sent to its partner TP in node B.

4. In node A, PS copies another 80 bytes of data from the TP storage to the limited buffer (the same buffer as above). In this example, 160 bytes of data have now been stored in the limited buffer.

5. The TP requests more data be sent to its partner TP in node B.

6. In node A, PS copies another 40 bytes of data from TP storage to the same limited buffer as above. The limited buffer now contains 200 bytes of data and is full. Any additional data (i.e., 40 bytes in this example) in the TP storage is copied to a new limited buffer; PS sends the full buffer first, then gets a new buffer to copy the rest of data. (If no more data is left in the TP storage when the limited buffer is filled, PS does not send the contents of the filled buffer until the end-of-chain indication is received from the local TP.)

7. In node A, PS sends the RU, MU(SEND_DATA_RECORD), to HS. In this example, HS sets the Pacing indicator and Request Larger Window indicator to PAC and RLW (see "Chapter 6.2. Transmission Control"), respectively. The RLW value requests that the new send pacing window be larger than the current one.

8. In node A, PC receives the MU from HS and calls BM, requesting the first demand buffer to begin segment generation. The number of segments that are generated depends on what the maximum send BTU size is (i.e., 100 bytes in this example). (See SNA Type 2.1 Node Reference for more details on segment generation.) This example assumes that segment generation is supported. If segment generation is not supported, the MU will remain in the same buffer (i.e., the PS-owned limited buffer) and be sent to DLC by PC. DLC then frees that limited buffer after the data is sent to node B.

9. In node A, PS calls BM to obtain another limited buffer to store the remaining TP data.

10. In node A, PC sends the first BTU (which contains 100 bytes of data) to DLC, and DLC sends it to node B. In node B, when the first segment (which contains the first 100 bytes of the record, MU[SEND_DATA_RECORD] is received by PC (in a link buffer allocated by DLC, a process not defined in detail in this book), the data remains in the link buffer (a DLC-owned buffer) and is passed to HS. Note: Segment reassembly is a PC function (on the receive side) and is done only when segment generation is done by the PC on the send side. Segment reassembly function is actually done in the transmission control (TC) process (see "Chapter 6.2. Transmission Control" for more details on segment reassembly).

11. In node A, PS copies and stores in a new limited buffer the unsent 40 bytes of data. This RU will be sent to HS when its buffer becomes full. In node B, HS calls BM to request a dynamic buffer to store the record, MU(SEND_DATA_RECORD), and sends it to PS later. The dynamic buffer pool was previously created by SM for the session. In node B, the size of the dynamic buffer (receiving the paced normal-flow request) is the same as the size of the limited buffer (sending the paced normal-flow request) in node A. HS also informs BM that node A has requested a new and larger send pacing window by passing the PAC and RLW indications in its GET_BUFFER request.

12. In node A, the demand buffer that the BTU was in is freed (returned to BM) by DLC after the first BTU is sent. This freeing action is asynchronous with the PC allocation of the demand buffer for the next segment. The order of their occurrence may vary from the example.) In node B, HS copies the record (in this example, it contains 100 bytes) from the link buffer to the newly obtained HS-owned dynamic buffer.

13. In node A, PC calls BM for another demand buffer to send the remaining 100 bytes. In node B, the link buffer is released (not reused by HS).

14. In node A, PC sends the second BTU to DLC. In node B, when the second segment that contains the last 100 bytes of the record is received by PC, the data remains in the link buffer (a DLC-owned buffer) and is passed to HS. HS copies the second segment from the link buffer

to the same dynamic buffer (which contains the first segment). The buffer contains 200 bytes and is now full.

15. In node A, PC returns the limited buffer that the MU(SEND_DATA_RECORD) was in to BM. In node B, HS sends the whole RU (which was reassembled from the first and second segments) to PS. PS then passes it to TP.

16. In node A, the demand buffer that the second BTU was in is returned to BM by DLC after the second BTU has been sent. In node B, the link buffer is released.

17. In node B, PS returns the dynamic buffer to BM after passing the whole RU to TP.

18. In node B, BM sends a BUFFERS_RESERVED signal to HS, which indicates a new send window for node A. This is because the buffer freed in step 17 was the one that received the PAC and RLW indications.

```
 ┌──────────────────────────────────────────────────────────────────────────┐
 │          LU(A)                                                 Node A    │
 │      ┌───────────────┐                                                   │
 │ TP   LU.PS           LU.HS            BM              PC            DLC  │
 └──────────────────────────────────────────────────────────────────────────┘
    .      .               .               .               .              .
    .      .               .               .               .              .
    .      .               .               .               .              .
    .      .               .               .               .              .
    .      .               .               .GET_BUFFER(link pool ID, no wait)  IPM
1   .      .               .               o<- - - - - - - - - - - - - - - o<──────
    .      .               .               .               . IPM(solicited,│
    .      .               .  MU(IPM, solicited, NWS=2)    .  NWS=2)       │
2   .      .               o<──────────────────────────────o<──────────────┘
    .      .               │ ADJUST_BUF_POOL(limited pool ID, + NWS buffers)
3   .      .               └ - - - - - - - ->o             .              .
    .      .               │ FREE_BUFFER(pointer to link buffer)          .
4   .      .               └ - - - - - - - ->o             .              .
    .      .               .               .               .              .
```

Note:

The signal names (e.g., GET_BUFFER, ADJUST_BUF_POOL, FREE_BUFFER) above the dashed lines refer to parameters (identifying request types) passed in Call invocations of the buffer manager; the additional parameters in parentheses are also passed to BM.

Figure B-4. Receiving a Solicited IPM

```
 Node B                                           LU(B)
                                          ┌────────────────────┐
 DLC             PC              BM       LU.HS               LU.PS      TP

    .              .            BUFFERS_RESERVED(REPLENISHED,      .        .
    .              .             total number in pool, pool ID,    .        .
    .              .             number of new buffers reserved)   .        .
    .              .             o──────────────>o                 .        . 1
 IPM(solicited, NWS=2)    MU(IPM, solicited,NWS=2)|                .        .
<───o<──────────────o<────────────────────────────┘                .        . 2
    |  FREE_BUFFER(pointer to demand buffer)         .             .        .
    └ - - - - - - - - - - - - - - - ->o              .             .        . 3
    .              .                 .               .             .        .
```

Note:

The signal names (e.g., FREE_BUFFER) above the dashed lines refer to parameters (identifying request types) passed in Call invocations of the buffer manager; the additional parameters in parentheses are also passed to BM.

Figure B-5. Sending a Solicited IPM

Figure B-4 on page B-16 and Figure B-5 on page B-17 show receiving and sending a solicited IPM. In this example, BM has been informed previously that HS in node A requested a larger send pacing window (see Figure B-3 on page B-13).

The comments below correspond to the numbers in Figure B-4 on page B-16 and Figure B-5 on page B-17.

1. In node B, BM sends a BUFFERS_RESERVED signal to HS to indicate more buffers are reserved for the buffer pool used to receive paced data from node A. In node A, the IPM received from its partner in node B is stored in a link buffer by DLC.

2. In node B, HS reuses the demand buffer that the BUFFERS_RESERVED signal was in and builds a solicited IPM. HS sends the IPM to node A through PC and DLC. The IPM contains a next-window size (NWS). In node A, the solicited IPM is sent to HS via PC.

3. In node B, the demand buffer is returned to BM after the IPM is sent to node A. In node A, upon receiving the solicited IPM, HS adjusts the limited pool limit count to the new send residual pacing count (via ADJUST_BUF_POOL) to reflect that a grant for a new send window has been received.

4. In node A, the link buffer is returned to BM after the IPM is processed. Now node A can use the new window grant to send more normal-flow requests to node B.

APPENDIX N. FSM NOTATION

A finite-state machine (FSM) is a combination of processing and memory, where the memory consists of the state of the FSM. The state can take one of a small number of named values (the state names). An FSM is defined by a matrix that lists the states and specifies the processing to be performed when the FSM is called. This processing typically depends on the current state of the FSM and on the input passed to the FSM, and may change the FSM state (resulting in a state transition) and produce output. Within this matrix definition, each state is given a number as well as its name, for notational convenience.

A number of alternative FSM definitions may be grouped together as a generic FSM, the definition to be used being assigned dynamically. The assignment of a particular definition to be used at a given time is called the binding of the generic FSM. A generic FSM can also be assigned to be a "no operation." A generic FSM is identified by the pound sign (#) prefix, e.g., #FSM_BIS.

The following operations are performed on an FSM:

- Call. Processing is performed as defined in the FSM definition for the existing combination of current state and input. This may involve a state transition.
- State check. Validity checking is performed for the existing combination of current state and input.
- State test. The current state of the FSM is tested for equality or inequality with a specified value.

An FSM is represented by a state-transition matrix.

The syntax of the state-transition matrix FSM definition is shown in Figure N-1 on page N-2. The column headings give the FSM state names, while the row headings name the inputs to the FSMs. The matrix elements—(row,column) intersections—define the state transitions and output actions.

Horizontal lines are used to group input lines together to improve readability. Their location has no bearing on the FSM function. For compactness, mnemonic abbreviations are used in the matrices.

The input lines within the matrix are scanned from top to bottom at execution time. The first input line found with all its conditions true is used to address the matrix for the next state and the output code. No more than one input line in a matrix has all its conditions true during a scan.

An FSM comes into existence initialized to state 1. If another state is to be the initial state, the FSM is initialized explicitly by calling the FSM with an appropriate signal.

Calling an FSM executes the FSM; i.e., an FSM action code is selected based on the current state of the FSM and the input line that is true. The input line evaluation uses the parameters or signal passed to the FSM. The FSM is scanned for a true input line from top to bottom of the matrix.

If the next-state indicator is a number n, the FSM enters state n. If the next-state indicator is a state-check indicator (>), the call of the FSM would act as if a no-state-change indicator (-) were encountered. (In practice, the formal description checks for such conditions prior to calling an FSM in order to perform special error handling.) If the next-state indicator is a cannot-occur indicator (/), this is an execution-time error; calls of the FSM cannot encounter this indicator because previous logic has filtered out the input for that state of the FSM.

If no input line is true, the call acts as if a no-state-change indicator (-) were encountered.

fname:

```
  +--------------------------------------+-----------+--------+
  |                 STATE NAMES---->     |   snam    |        |
  |                                      |  [. . .]  |  . . . |
  | INPUTS          STATE NUMBERS-->     |   snum    |        |
  +--------------------------------------+-----------+--------+
[ | ic [ ,ic ] . . .                     |    ac     |        | ]
[ | ic [ ,ic ] . . .                     |    ac     |        | ]
[ +--------------------------------------+-----------+--------+ ]
[ | ic [ ,ic ] . . .                     |    ac     |        | ]
[ |         . . .                        |           |        | ]
[ | ic [ ,ic ] . . .                     |    ac     |        | ]
  +--------------------------------------+-----------+--------+
[ |                                                           | ]
[ +--------+--------------------------------------------------+ ]
[ | OUTPUT | FUNCTION                                         | ]
[ |  CODE  |                                                  | ]
[ +--------+--------------------------------------------------+ ]
[ | oc-1   | Output logic statements                          | ]
[[ +--------+--------------------------------------------------+ ]]
[[ |        |  . . .                                           | ]]
[[ +--------+--------------------------------------------------+ ]]
[[ | oc-n   | Output logic statements                          | ]]
  +--------+--------------------------------------------------+
```

Legend:

[ ] = optional parameter
fname = FSM name
snam = state name component
snum = state number
ic = input condition name
ac = action code

An action code (ac) has the syntax: ns[(oc)], where:

ns = next-state indicator
oc = output code (The parentheses around the oc are sometimes omitted to save space.)

Possible next-state indicators and associated action code formats are:

n[(oc)] normal state transition to state n (corresponding to some snum)
-[(oc)] same-state transition—remain in the same state
>[(oc)] error condition, no state change
/ "cannot occur" condition, no state change

Figure N-1. Syntax of an FSM State-Transition Matrix

APPENDIX T. TERMINOLOGY: ACRONYMS AND ABBREVIATIONS

ACT	activate
API	application programming interface
ASCII	American Standard Code for Information Interchange
ASM	address space manager
BB	Begin Bracket
BBI	Begin Bracket indicator
BBIU	Begin Basic Information Unit
BBIUI	BBIU indicator
BC	Begin Chain
BCI	Begin Chain indicator
BETB	between brackets
BIND	BIND SESSION
BIS	BRACKET INITIATION STOPPED
BIU	basic information unit
BM	buffer manager
CD	Change Direction
CDI	Change Direction indicator
CEB	Conditional End Bracket
CEBI	Conditional End Bracket indicator
CINIT	CONTROL INITIATE
CNOS	change number of sessions
CONLOSER	contention-loser
CONWINNER	contention-winner
COPR	control operator services
COS	class of service
CP	control point
CRV	CRYPTOGRAPHY VERIFICATION
CT	correlation table
DAF	Destination Address field
DES	Data Encryption Standard
DFC	data flow control
DIA	Document Interchange Architecture
DLC	data link control
DR1	Definite Response 1
DR1I	Definite Response 1 indicator
DR2	Definite Response 2
DR2I	Definite Response 2 indicator
DSU	distribution service unit
EB	End Bracket
EBI	End Bracket Indicator
EBIU	End Basic Information Unit
EBIUI	EBIU indicator

EC	End Chain
ECI	End Chain indicator
ED	Enciphered Data
EDI	Enciphered Data indicator
EFI	Expedited Flow indicator
ERI	Exception Response indicator
ERP	error recovery procedures
ESD	extended sense data
EXP	expedited
EXR	EXCEPTION REQUEST
FDX	full-duplex
FF	flip-flop
FI	Format indicator
FIC	first in chain (BC, ¬EC)
FM	function management
FMD	function management data
FMH	FM header
FMP	FM profile
FQPCID	Fully Qualified Procedure Correlation Identifier
FSM	finite-state machine
FSP	first speaker
GDS	general data stream
HDX	half-duplex
HDX-FF	HDX flip-flop
HS	half-session
HSID	half-session identification
ID	identifier, identification
INB	In Bracket
INIT-SELF	INITIATE-SELF
IPM	ISOLATED PACING MESSAGE
IPR	ISOLATED PACING RESPONSE
LFSID	local-form session identifier
LIC	last in chain (¬BC, EC)
LL	logical record length (prefix)
LLID	logical record length and GDS ID (prefix)
LU	logical unit
LUCB	LU control block
LUSTAT	LOGICAL UNIT STATUS
LUW	logical unit of work
MC	mapped conversation
MCR	mapped conversation record
MGR	manager
MSG	message
MU	message unit
NAU	network addressable unit
NC	network control

NEG	negative
NG	no good
NOF	node operator facility
NTWK	network
NWS	next-window size
OAF	Origin Address field
ODAI	OAF-DAF Assignor indicator
OIC	only in chain (BC, EC)
PAC	Pacing Request, Pacing Response (value of PI in RH)
PB	protocol boundary
PC	path control
PCID	procedure correlation identifier
PD	Padded Data
PDI	Padded Data indicator
PI	Pacing indicator
PIP	program initialization parameters
PIU	path information unit
PLU	primary LU
POS	positive
PRI	primary
PS	presentation services
PS.CONV	presentation services for (basic) conversation
PS.COPR	presentation services for the control operator
PS.CRPM	presentation services--conversation services resource manager
PS.SPS	presentation services--sync point services
PS.MC	presentation services for mapped conversations
PTR	pointer
PU	physical unit
Q	queue
QR	Queued Response
QRI	Queued Response indicator
R	receive, receiving
RC	return code
RCB	resource control block
RCV	receive
RESYNC	sync point resynchronization service TP
RH	request/response header
RLWI	Request Larger Window indicator
RM	resources manager
RPC	residual pacing count
RQ	request
RQD	RQ indicating definite-response required
RQE	RQ indicating exception-response requested
RQN	RQ indicating no-response required
RRI	Request/Response indicator

RSP	response
RTI	Response Type indicator
RTR	READY TO RECEIVE
RU	request/response unit
SC	session control
SCB	session control block
SD	Sense Data Included
SDI	Sense Data Included indicator
SEC	secondary
SESS	session
SESSEND	SESSION ENDED
SESSST	SESSION STARTED
SIG	SIGNAL
SLDLM	session-limit data-lock manager
SLU	secondary LU
SM	session manager
SNA	Systems Network Architecture
SNA/DS	SNA Distribution Services
SNASVCMG	SNA services manager (LU-LU session mode name)
SNF	Sequence Number field
SON	session-outage notification
SQN	sequence number
SS	session services
SSCP	system services control point
SSLS	source-LU session-limit services
SVC	service; services
SYNCPT	synchronization point
TC	transmission control
TCB	transaction control block
TCCB	transmission control control block
TERM	terminate, terminating, termination, terminal
TH	transmission header
TP	transaction program
TPF	transmission priority
TPN	transaction program name
TS	transmission services
TSLS	target-LU session-limit services
TSP	TS profile
UNBIND	UNBIND SESSION
UPM	undefined protocol machine
URC	user request correlation
VR	virtual route

SPECIAL CHARACTERS

A

B

C

D

I

M

N

O

P

S

T

U

Reader's Comment Form

Systems Network Architecture
LU 6.2 Reference
Peer Protocols

Publication No. SC31-6808-0

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: Copies of IBM Publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Possible topics for comment are: clarity, accuracy, completeness, organization, coding, retrieval, and legibility.

Comments: ____

What is your occupation? ______________________________

If you wish a reply, give your name, company, mailing address, and date:

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC31-6808-0

Reader's Comment Form

Fold and tape | Please Do Not Staple | Fold and tape

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

International Business Machines Corporation
Networking Architecture
Dept. E96
P.O. Box 12195
Research Triangle Park, N.C. 27709-9990

Fold and tape | Please Do Not Staple | Fold and tape

IBM ®

Reader's Comment Form

Systems Network Architecture
LU 6.2 Reference
Peer Protocols

Publication No. SC31-6808-0

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: Copies of IBM Publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Possible topics for comment are: clarity, accuracy, completeness, organization, coding, retrieval, and legibility.

Comments: ______________________________

____

____

____

____

____

____

What is your occupation? ______________________________

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC31-6808-0

Reader's Comment Form

Fold and tape | Please Do Not Staple | Fold and tape

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

International Business Machines Corporation
Networking Architecture
Dept. E96
P.O. Box 12195
Research Triangle Park, N.C. 27709-9990

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

Publication Number
SC31-6808-0

Printed in USA

SC31-6808-00