# Systems Network Architecture

# Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2

SC30-3269-2

IBM

Systems
Network
Architecture

# Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2

SC30-3269-2

Third Edition (December 1984)

This publication obsoletes document SC30-3269-1.

## PREFACE

This book is intended for product developers, system programmers, and others who need detailed information about Systems Network Architecture (SNA) logical unit (LU) type 6.2 in order to develop or adapt a product or program to function within an SNA network. The book describes the formats and protocols for LU type 6.2 from a design viewpoint.

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.

## 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.
- 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.

## RELATED PUBLICATIONS

- SNA Format and Protocol Reference Manual: Architectural Logic, SC30-3112—comprehensive information on the formats and protocols of SNA nodes.
- SNA Reference Summary, GA27-3136—summary information on SNA formats and sequences.
- SNA—Sessions Between Logical Units, GC20-1868—reference information on SNA formats and protocols for LU types other than type 6.2.
- IBM SDLC General Information, GA27-3093—supplementary details of Synchronous Data Link Control.

# CONTENTS

# LIST OF ILLUSTRATIONS

# 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 <u>protocol machines</u>. 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:

- <u>Finite-state machines</u>: 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.

- <u>Routing and checking logic</u>: 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.

- <u>Block diagrams</u>: 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.

- <u>Protocol boundaries</u>: A protocol boundary is a specification of the format and content requirements imposed on the signals exchanged between protocol machines.

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.

- Appendixes D through I provide details of the general data stream and various headers, request-response units, profiles, and sense data used in SNA.

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

- Appendix T (included as foldout pages at the back of the book) provides a comprehensive list of abbreviations and acronyms used in the book.

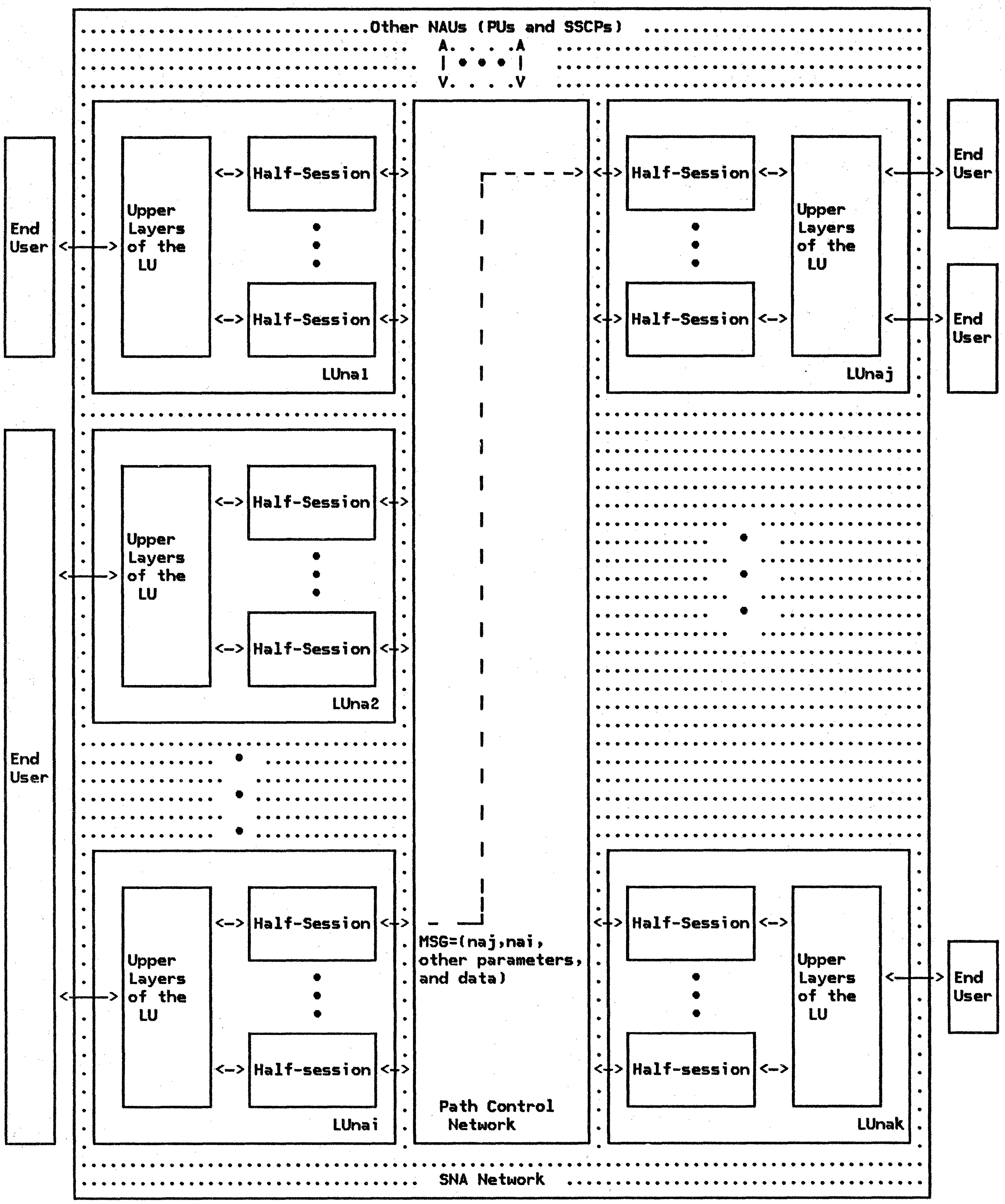


Figure 1-1. Overview of the SNA Network

## 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, each of which in a general SNA network is associated with a network address (na). 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 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 = (naj,nai,other parameters, and data),

where naj is an address of the destination NAU, and nai that of the origin NAU. (The pair, naj and nai, together identify a particular session; their form varies depending on the types of nodes involved.) 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

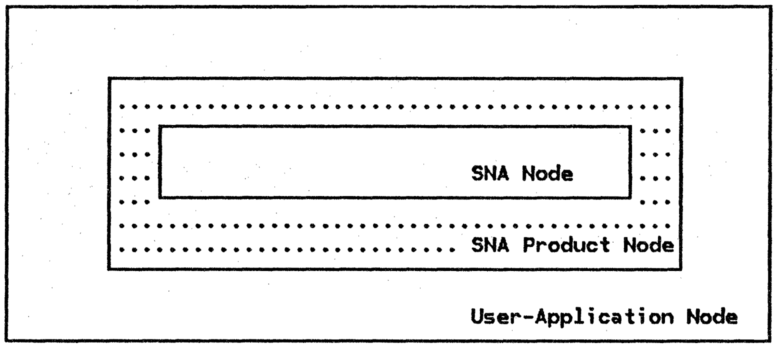
The SNA network physically consists of nodes interconnected via links. An _SNA node_ is a grouping of SNA-defined protocol machines. An _SNA product node_ may consist of additional, product-specific protocol machines that use one or more SNA nodes. A _user-application node_ may consist of additional, 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_.

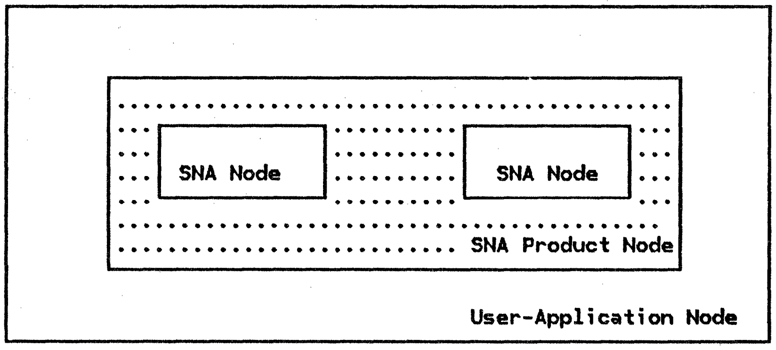
In this book, "node" is synonymous with "SNA node," and the qualifier will generally be omitted. Thus, end users and protocol machines not defined in SNA are external to the node, as that term is used hereafter.

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.

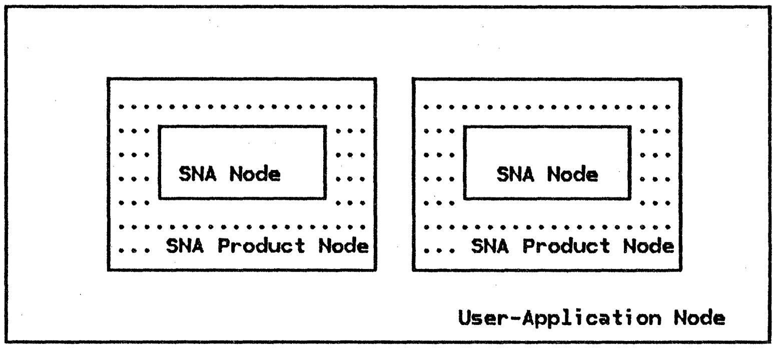
For example, type 2.1 nodes can connect to the general subarea routing network or to other type 2.1 nodes directly. In the former case, subarea nodes (discussed below) provide general intermediate routing within the path control layer, allowing complex network configurations to be fashioned; in the latter



(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

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, 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

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, 2.1, 4, or 5) of node in which it resides.

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 book, node types 2.1 and 5 are of primary interest, as these are the only nodes that include LU 6.2 implementations.

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 nodes each contain a peripheral node control point (PNCP), which provides services on a more local scale than an SSCP provides. In particular, a PNCP can mediate LU-LU session-initiation requests (by doing local address look-up) in the type 2.1 node peer-to-peer context just as an SSCP does in the more general network configuration 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 "Appendix D. RH 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."

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.

Abbreviations commonly used in the text are listed at the back of the book on foldout pages (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, and peripheral node to peripheral 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.

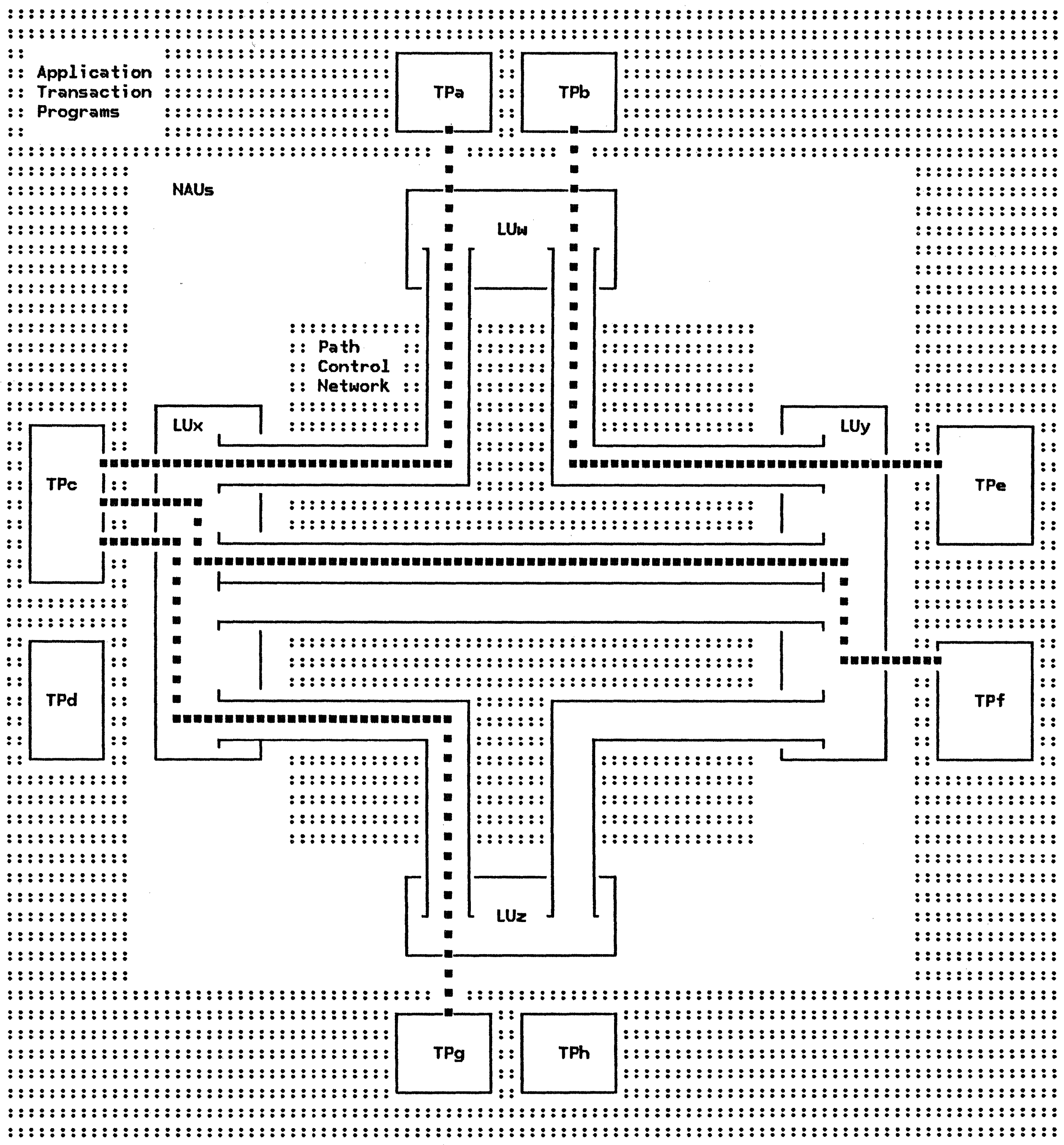
An LU may run many transaction programs successively, concurrently, or both. Each transaction program may be connected to one or more other transaction programs by conversations. Multiple conversations between different pairs of transaction programs can be active concurrently, with each conversation 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: the way it is invoked, and the communication functions it initiates.



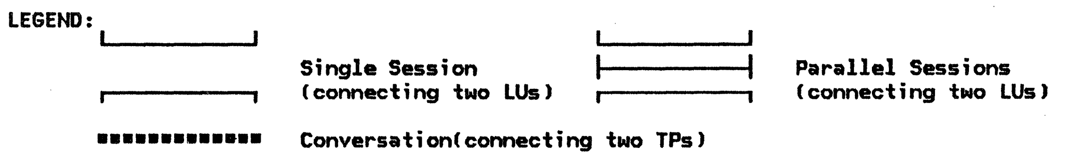

LEGEND:
Single Session (connecting two LUs)
Parallel Sessions (connecting two LUs)
Conversation(connecting two TPs)

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

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, the LU generates an internal

Attach to simulate invocation by another transaction program. It does this in response to some external stimulus, e.g., operator action.)

A transaction program uses the LU to communicate with other transaction programs by issuing transaction program verbs (which are described in the publication 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 (SNADS) 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. They are not explicitly visible to transaction programs; for example, a transaction program cannot explicitly request use of a particular session.

A mode is a set of characteristics that may be associated with a session. These characteristics typically correspond to different requirements for cost, performance, security, 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 characteristics.

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 points that serve 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, however, does provide one or more concrete representations of these functions and parameters. Such a representation of the transaction program protocol boundary is called the an application program interface (API). For example, an API might be statements in a programming language that an implementation supports.

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 SNADS. 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.

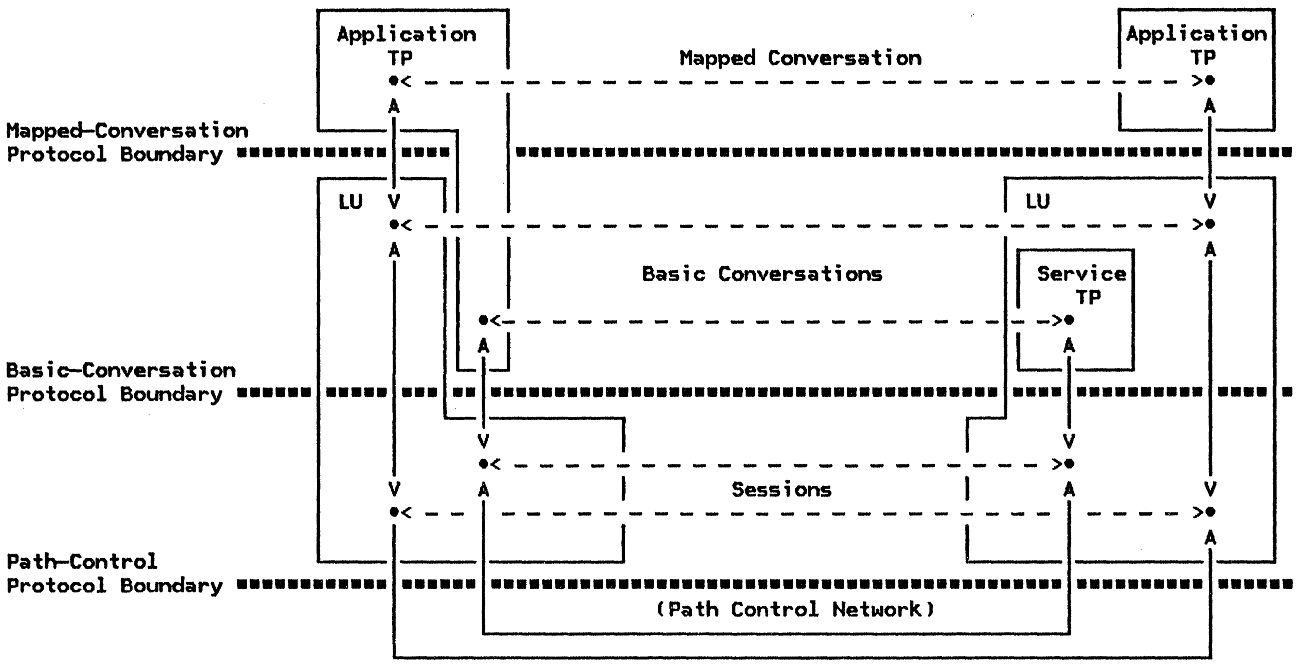
A protocol boundary can be interpreted from two points of view.

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 are called layer protocols.

But from another point of view, a protocol boundary is a boundary between two peer 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. From this viewpoint, the rules of exchange are called peer protocols.

From either viewpoint, the operations and flows across the boundary are the same, e.g., a transaction program uses the same verbs and data formats whether the interaction is thought of as TP-TP or as TP-LU. Specifically, the formats and protocols for peer exchange are the same as those for layer exchange with the next lower layer.

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 interlayer 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).



LEGEND:
<- - -> conceptual flows between peer components (peer exchange)
<———> actual flows across interlayer protocol boundaries (layer exchange)
■■■■■■■ protocol boundary between layers or sublayers

Figure 2-2. Peer and Layer Exchanges

NAMES

The LU allows transaction programs to refer to its resources, such as other TPs and LUs 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 addresses 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 "Appendix H. FM Header and LU Services Commands". (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. The target LU always creates a new transaction program instance for each Attach it receives.

## 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 local LU, but not necessarily outside an LU.

The path control network routes information to an LU by a _network address_ rather than by a name. The correspondence between names and addresses is maintained at the _control point_, which is another NAU that assists the LU during session initiation.

The control point identifies each LU by its _fully qualified LU name_ (also called _network-qualified LU name_). It 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, which may contain multiple domains (for more information on domains, see "Chapter 1. Introduction").

The control point uses the fully qualified LU name of the intended partner LU to determine the corresponding network addresses used for routing within the path control network. The LUs themselves use their fully qualified LU names for certain purposes; for example, LUs resolve some race conditions by exchanging and comparing their fully qualified LU names.

An LU may provide another set of names by which it refers to remote LUs when issuing session-initiation requests to its control point: these names are called _uninterpreted LU names_. Each uninterpreted LU name is unique within a particular initiating LU, and is known to that LU's control point but is not known elsewhere in the network.

The LU name is converted into the network address in stages. If the LU uses an uninterpreted LU name to identify its partner, the control-point translates this into a fully qualified LU name; otherwise, the LU supplies the fully qualified LU name to the control point directly. Then, the the control point provides the network address for that fully qualified LU name.

## Mode Names

A source TP cannot select a particular session for a conversation, but it can specify that the session selected have a particular set of characteristics, or _mode_. It does this by specifying a corresponding _mode name_.

Mode names are unique 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 (SNADS). 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 SNADS, 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.

SNADS 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 SNADS 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 at the same time, even with the same 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, can have only one active LU-LU session at a time (but can have successive sessions with different partner LUs). 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.

An LU that does not support parallel sessions can have only one active LU-LU session at a time. A parallel-session LU can have, concurrently, one or more parallel-sessions with each of one or more parallel-session LUs, and one single session with each of one or more single-session LUs. (No middle capability [multiple-session LU] exists, i.e., any LU that supports multiple concurrent single sessions also supports parallel sessions.)

### 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 of the session. 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 (or first-speaker session). A session that the local LU will surrender to the partner is called a contention-loser session (or the 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 for that particular (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 more detail in "Chapter 5.4. Presentation Services--Control-Operator Verbs".

## 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.[1]

Session shutdown consists of the LU activity to terminate conversation activity on a session prior to deactivating the session.[2]

Session activation and deactivation consists of creating or destroying the end-to-end logical connection between the LUs.[3]

## 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; the LU that receives BIND is called the secondary LU. 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.)

[1] Session initiation and termination protocols use session services RUs, e.g., INIT_SELF, CINIT.
[2] Session shutdown protocols use data flow control RUs, e.g., BIS.
[3] Session activation and deactivation protocols use session control RUs, e.g., BIND, UNBIND.

### 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.

### 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 "Appendix F. Profiles"). The profile used depends on the kind of session and the kind of node:

- On an LU-LU session, all LUs use FM profile 19 and TS profile 7.
- On a CP-LU session, an LU in a subarea node uses FM profile 6 and TS profile 1.
- On a CP-LU session, an LU in a peripheral node uses FM profile 0 and TS profile 1.

## SECURITY

The LU provides two functions to assist the installation in providing security.

To help prevent unauthorized remote programs from accessing local transaction programs, the LU optionally verifies the identity of the remote user by means of a user ID and password supplied in the Attach FM header. (User IDs and passwords are verified and enforced in an implementation-defined way.)

To help prevent data from being interpreted 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 the next verb issued. 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 2-12); 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 2-12), 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.

### 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 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.

The principal base and optional functions are listed below. The complete sets are defined in *SNA Transaction Programmer's Reference Manual for LU Type 6.2*.

### 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.

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.

## MESSAGE UNITS AND THEIR TRANSFORMATIONS

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 Appendix I).

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 "Appendix H. FM Header and LU Services Commands").

## 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 2-5 on page 2-16.

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 two 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.

#### Basic Information Unit

A basic information unit (BIU) is the message unit transferred between two LUs. It consists of a request 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. "Appendix D. RH Formats" gives further details.

RUs carry FM headers, TP-supplied data (formatted by the TP or the LU into logical records), and other protocol information. The LU uses the following RUs on an LU-LU session:

- Category FMD RUs, for transaction-program data

- Category DFC RUs, such as BIS, LUSTAT, RTR, SIG

- EXR, for some path-control-detected errors

(For details, see "Appendix E. Request-Response Unit (RU) Formats" and "Appendix H. FM Header and LU Services Commands".)

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

## SESSION MESSAGE-UNIT SEQUENCES

The following sequences of BIUs are relevant to session protocols:

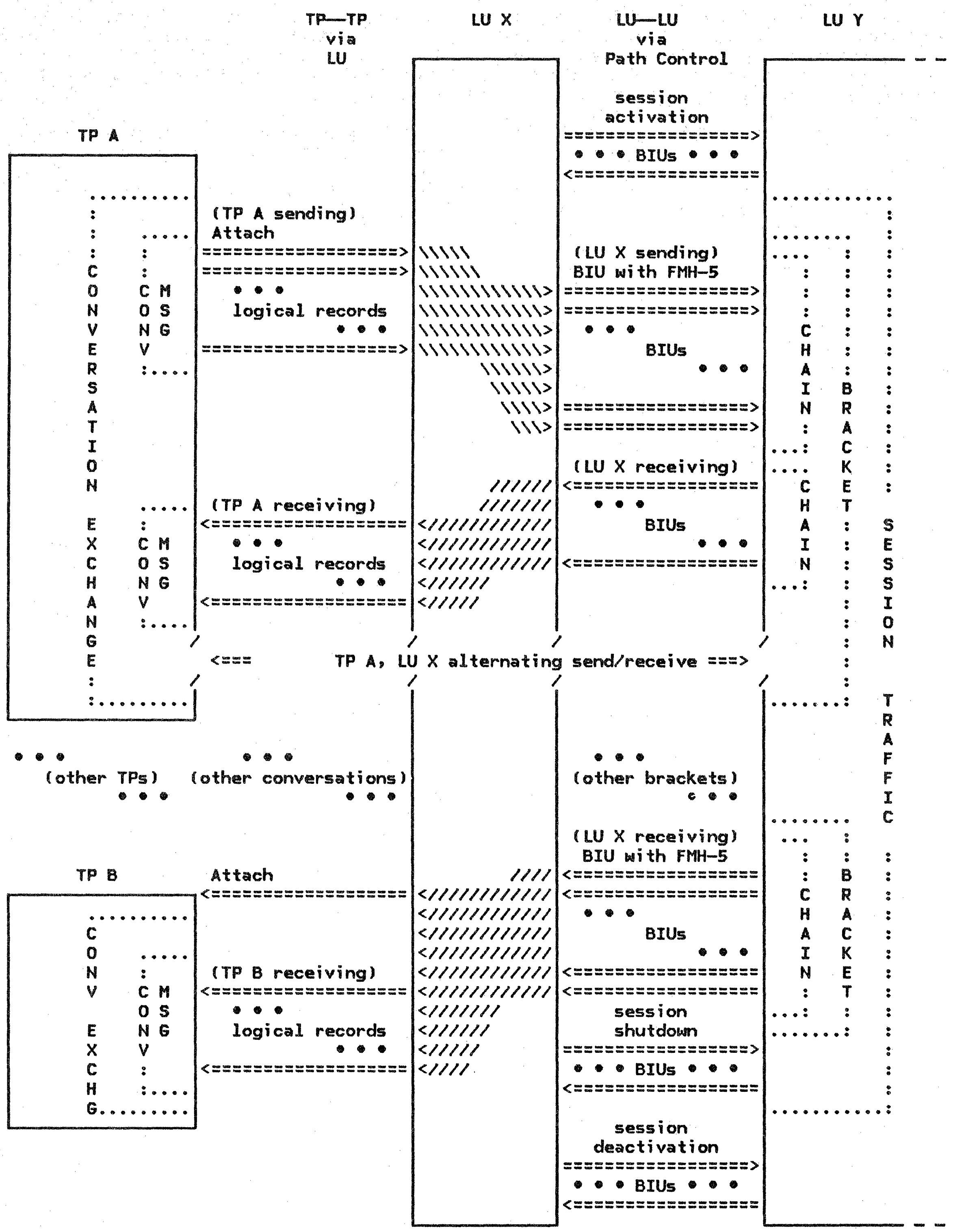
A (BIU) chain is a sequence of BIUs that constitute a single unidirectional transfer. The chain is the most elementary unit that can be independently confirmed or for which errors can be reported using SNA-defined LU protocols. It corresponds to a TP-TP conversation message.

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.

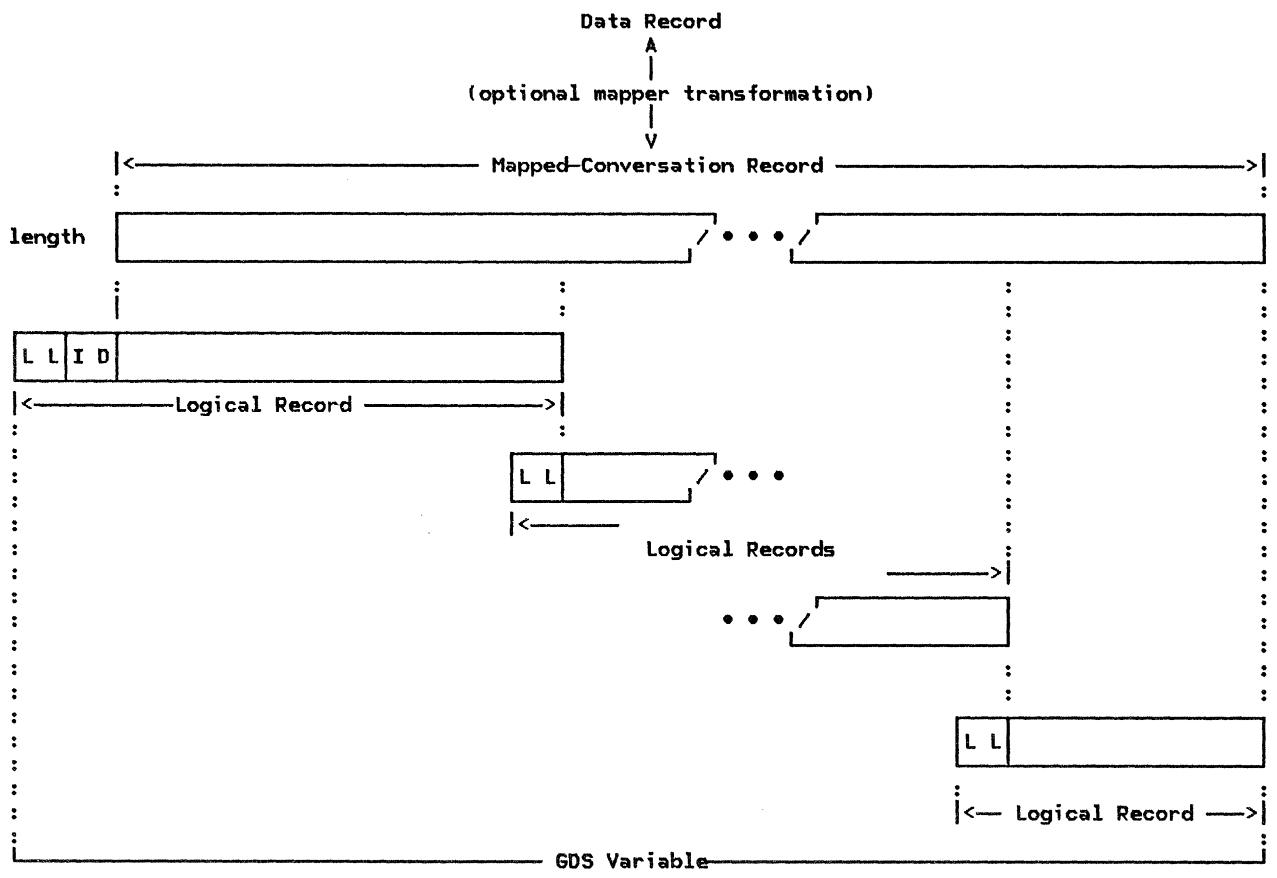
Figure 2-3 on page 2-14 illustrates the correspondence between the conversation message-unit sequences and session message-unit sequences. In the figure:

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



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

Figure 2-3. 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 2-4. Relationship of Data Records to Logical Records (Example)

(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

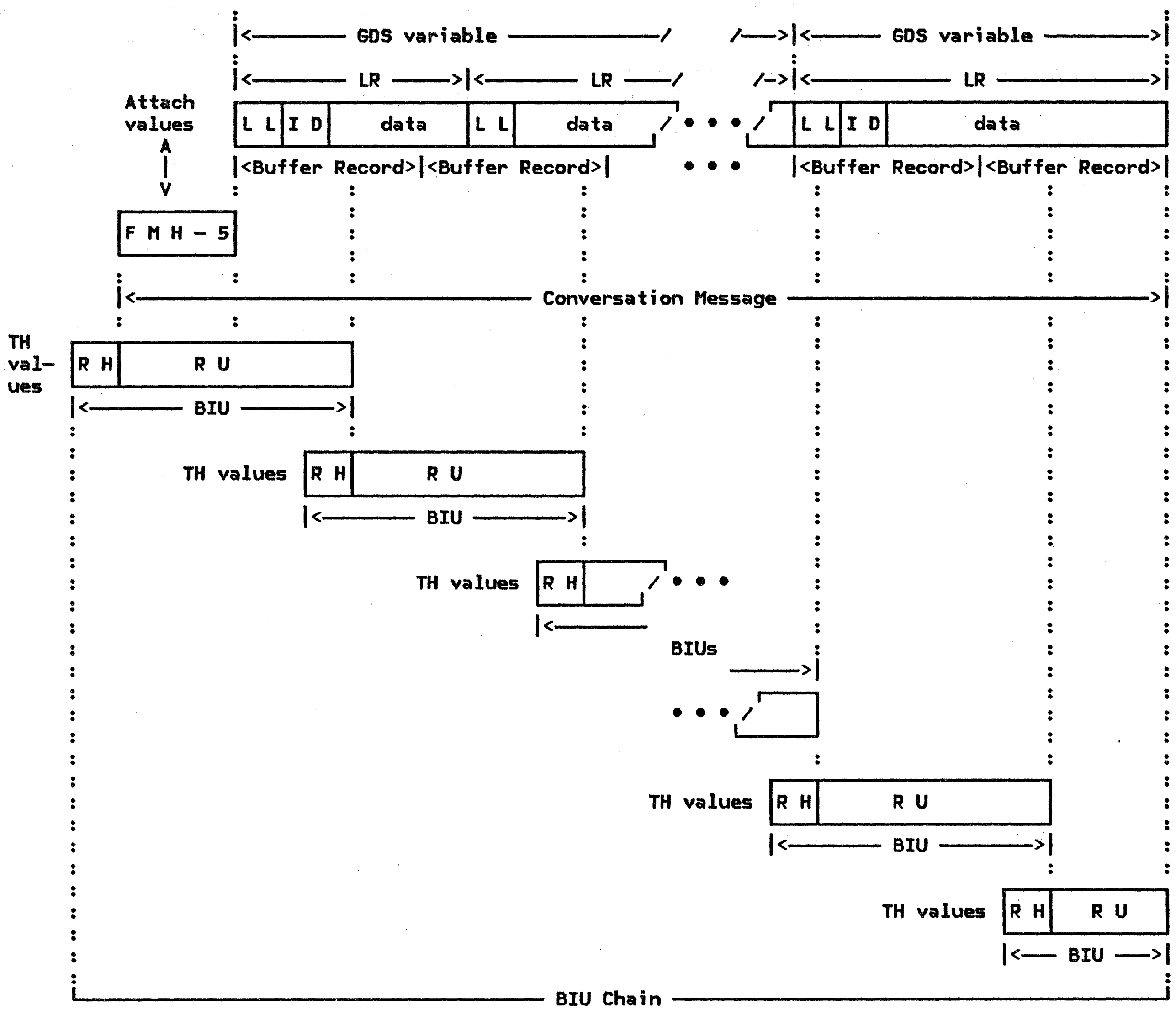
The mapped-conversation support in the LU converts a data record into a GDS variable.

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 2-4.

## 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.



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 2-5. Relationship of Conversation Message to BIU Chain (Example)

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 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 2-5 illustrates the conversion process.

DATA EXCHANGE WITH OTHER NAUS

The LU also exchanges message units with other NAUs, specifically with the CP, via the CP-LU session, and with the PU, directly. These message units are listed in "Chapter 4. LU Network Services" and are described briefly below.

### LU-CP Message Units

The LU sends session services RUs on the CP-LU session. These RUs are used in the session-initiation protocols for LU-LU sessions, e.g., for translating the partner LU name into the network address. In some cases, the choice of RUs depends on the type of node (subarea or peripheral) containing the sending LU.

The LU also uses the CP-LU session to send and receive maintenance services RUs.

### LU-PU Records

The LU has a direct protocol boundary with the PU in its node.

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

Another group of LU-PU internal records is used to connect the LU to other node components or to reset the LU.

## 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 2-6 on page 2-18 through Figure 2-22 on page 2-26. 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 2-21 shows how these flows can be combined, or sequenced, to form complete conversations.

Finally, "Error Flows" on page 2-23 shows asynchronous response cases.

NOTATION

The following notation is used in the figures.

| | |
|---|---|
| ——> | Request RU |
| <----- | Response RU |

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 |

<u>Verbs and Parameters</u>

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.

<u>Data Transfer Description</u>

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 reprsented 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 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 2-6 through Figure 2-13 on page 2-20.

```
SEQUENCE 1

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

Figure 2-6. Start Conversation without Confirmation

```
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 2-7. 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 2-8. Finish Conversation without Confirmation

```
SEQUENCE 4

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

Figure 2-9. Start Conversation with Confirmation

```
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 2-10. 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 2-11. Conversation Turnaround with Confirmation, 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 2-12. Conversation Turnaround with Confirmation, 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 2-13. Finish Conversation with Confirmation

## 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 2-14 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 2-6 on page 2-18 through Figure 2-13 on page 2-20.

  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 2-15 on page 2-22 and Figure 2-16 on page 2-22 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 2-14. Possible Next Sequence in Error-Free Cases

```
ALLOCATE
   SYNC_LEVEL(NONE)   BC,BB,FMH-5
                      ------------------------> (TP started)
SEND_DATA             data                      RECEIVE_AND_WAIT        [NOTE 1--see text]
                      ------------------------>   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 2-15. One-Way Conversation without Confirmation: Combines Sequences 1 and 3

The sequence shown in Figure 2-15 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 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 2-16. Two-Way Conversation with Confirmation: Combines Sequences 4, 6B, and 7.

The sequence shown in Figure 2-15 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 2-17. 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 2-17 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 2-18 on page 2-24 through Figure 2-22 on page 2-26 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 Flow Sequences" on page 2-47.

```
 :                           :                  :
 SEND_DATA
 (TP detects                                RECEIVE_AND_WAIT
  an error)
 SEND_ERROR          data
right 4            ───────────────────────>  WHAT_RECEIVED=DATA_INCOMPLETE
 SEND_DATA           FMH-7(0889),data       RECEIVE_AND_WAIT
                   ───────────────────────>   WHAT_RECEIVED=PROGRAM_ERROR_TRUNC
 :                           :                  :
```

Figure 2-18. 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 2-19. SEND_ERROR Issued by Receiver:

The SEND_ERROR verb causes a negative response to the incoming chain; the sending TP 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 2-20. 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 2-21. 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 2-22. 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 2-23 on page 2-27 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 the path control protocol boundary, below which is the SNA path control network, which the LU uses to communicate with other LUs and with its control point (CP).

The LU also has a protocol boundary with the PU (see "Chapter 4. LU Network Services").

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

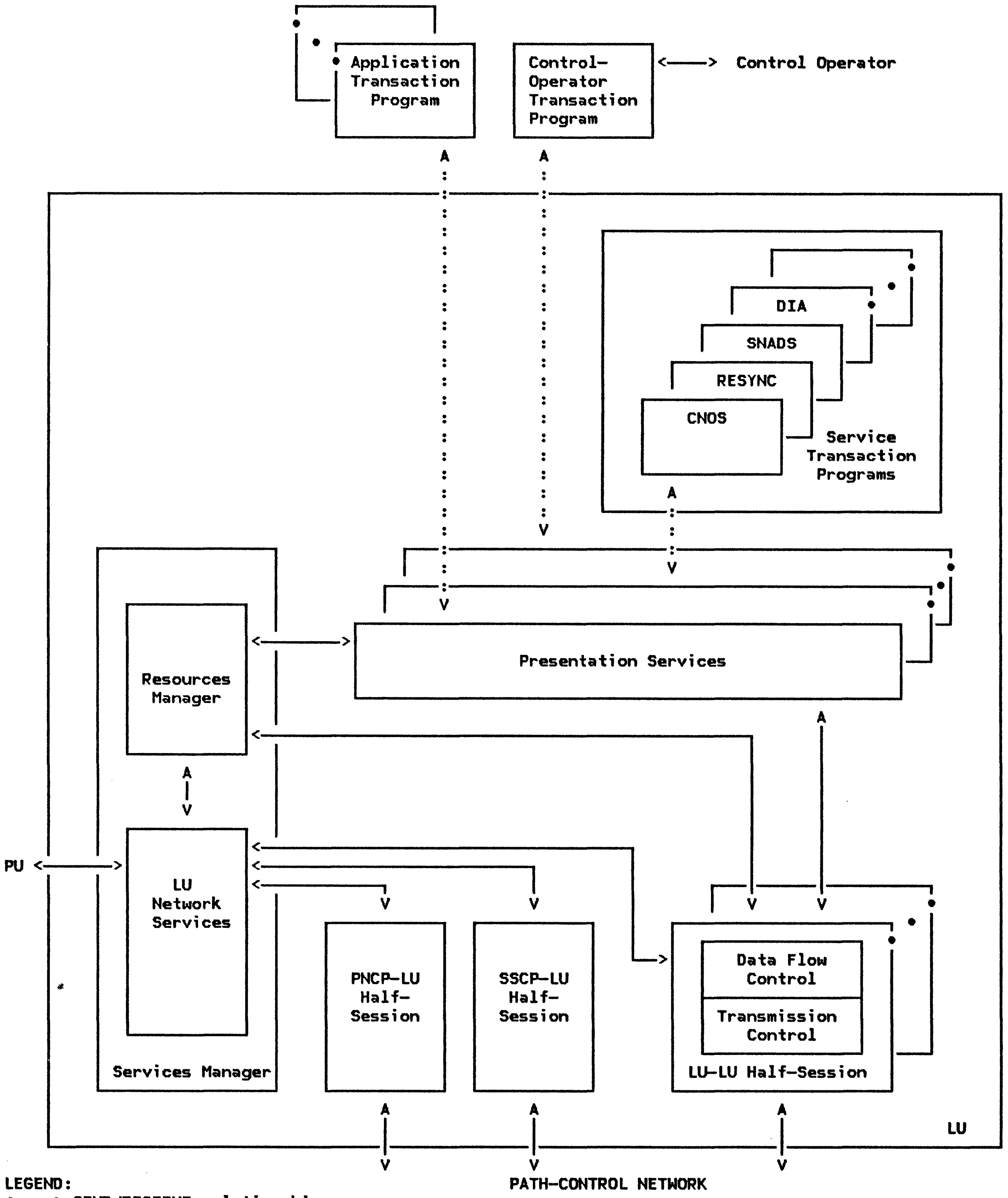
The upper layer contains transaction processes, which contain instances of the following SNA layers:

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 2-24 on page 2-28.)

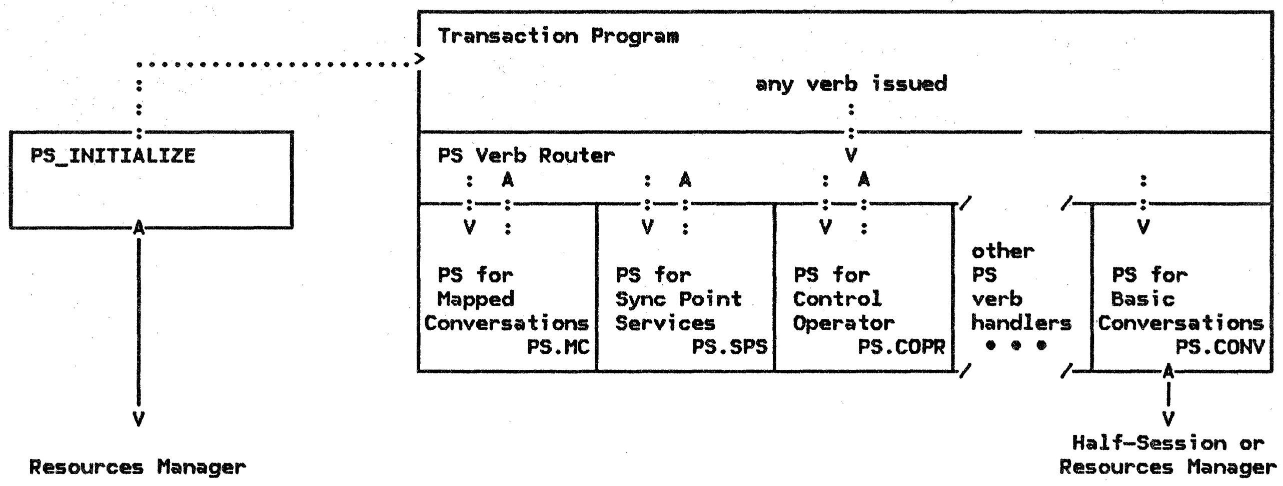
This layer is managed by the resources manager component (RM), which creates transaction processes (in response to Attaches received from remote LUs), destroys them after they have finished executing, and connects them with sessions (thus enabling them to participate in distributed transactions).



LEGEND:
<——> SEND/RECEIVE relationship
<....> CALL/RETURN relationship

| | |
|---|---|
| CNOS: Change Number of Sessions | RESYNC: Sync Point Resynchronization |
| SNADS: SNA Distribution Services | DIA: Document Interchange Architecture Services |

Figure 2-23. Overview of LU 6.2 Components


Transaction Program
any verb issued
PS_INITIALIZE
PS Verb Router
PS for Mapped Conversations PS.MC
PS for Sync Point Services PS.SPS
PS for Control Operator PS.COPR
other PS verb handlers • • •
PS for Basic Conversations PS.CONV
Resources Manager
Half-Session or Resources Manager

LEGEND:
.....> CALL/RETURN relationship (within a process)
<———> SEND/RECEIVE relationship (between processes)
NOTE: PS verb router is called recursively by PS verb handlers.

Figure 2-24. Structure of a Presentation Services Process

The lower layer 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 message 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). While most of these are LU-LU half-sessions for transporting conversation data, one of them must be a CP-LU half-session connecting the LU to its Control Point.

This layer is managed by the LU network services component (LNS), which creates and destroys half-sessions and interacts with SNA components outside the LU (the control point and the nodal NAU manager in the PU).

The resources manager and LU network services components are created by the PU when it activates the LU; they run continuously thereafter.

## 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 Flow Sequences" on page 2-47. In particular, Figure 2-33 on page 2-48 and Figure 2-34 on page 2-49 illustrate the interactions, at the source and target LUs, respectively, for a typical conversation; Figure 2-35 on page 2-50 and Figure 2-36 on page 2-51 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 2-25 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 2-25. 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, PS transforms the mapped-conversation record (MCR) into logical records, determines message-unit sequence boundaries such as the end of a conversation message, and passes the data and control information to HS. HS converts the logical records into one or more RUs, encodes the protocol information into the RH, and passes the resulting BIU and TH information to path control.

When receiving, HS checks incoming BIUs for format and protocol validity and passes the data to PS. When the TP issues a RECEIVE_AND_WAIT verb, PS checks the verb for validity, waits until HS supplies the requested amount of data, and passes the data and protocol information back to the TP.

The following sections discuss these functions in more detail. (Figure 2-3 on page 2-14, Figure 2-4 on page 2-15, and Figure 2-5 on page 2-16 illustrate the message-unit relationships discussed.)

### <u>Sending Data</u>

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 issues SEND_DATA verbs as often as necessary (determined by the buffer-record size used by the PS.MC implementation) to send all the logical records.

PS (in particular, the PS verb router) is recursively callable: it is called by a TP when the TP issues a verb, and it is also called by verb handlers within PS that themselves issue verbs. For example, the mapped-conversation verb handlers in PS typically issue one or more basic-conversation verbs 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 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 an internal buffer of implementation-defined length. When the 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, RECEIVE_AND_WAIT, CONFIRM, SYNCPT, or 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 reblocks it into RU-sized units (the RU size for a session is determined by BIND negotiation when the session is activated). When HS has received more data than necessary to fill an RU, it generates an RH, builds the BIU, and generates 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., PREPARE_TO_RECEIVE verb issued), RQD2 (e.g., CONFIRM issued), RQD1 (DEALLOCATE TYPE(ABEND)) issued), and CEB (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 session-level pacing. The sending HS sends at most one 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. Depending on its ability to receive additional data, the receiver may send a pacing response at any time after receiving the first BIU of a window.

### Receiving Data

The HS process at the receiving LU receives BIUs and TH information from path control. It sends pacing responses when it is able 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 LNS to issue UNBIND indicating a format or protocol error.

If the BIU is satisfactory, HS sends the Attach FM header, if present, to RM, and sends all other RU data 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 repeatedly (and recursively) issues RECEIVE_AND_WAIT verbs until it receives a complete MCR from HS.

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. As PS receives the data, which includes LL fields, PS accumulates the data in an internal buffer, until it reaches the end of a logical record (or buffer record). While accumulating the data, PS keeps track of the LL fields, to verify that the conversation message ends on a logical record boundary.

When the PS verb handler for RECEIVE_AND_WAIT returns (recursively) to the PS verb handler for MC_RECEIVE_AND_WAIT, PS checks the length and continuation fields in the LLs to verify that a complete 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.

### Internal Buffering

Figure 2-26 on page 2-31 illustrates internal buffering that the LU may perform during send and receive operations. The figure has the following meaning.

Column (A)

<u>TP send buffer record</u> is the DATA parameter (LL and data) of the SEND_DATA verb.

Column (B)

<u>PS send buffer</u> is a buffer in the sending PS of implementation-defined length (in this example, 6) for accumulating TP data to be sent to HS.

<u>PS-to-HS record</u> is the data transferred to HS from a full PS send buffer.

Column (C)

<u>HS internal buffer</u> is a buffer in the sending HS of RU size (in this example, 4) that accumulates data from PS until a complete RU can be sent.

|  | Source LU |  |  |  |  | Path Control | Target LU |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|
|  | (A) |  | (B) |  | (C) |  | (D) | (E) |  |  |
|  | TP Send Buffer Record |  | PS Send Buffer (length 6) | PS-to-HS Record (length 6) | HS Internal Buffer (RU size 4) | HS-to-HS via PC (RU size 4) | HS-to-PS Record (RU size 4) | PS Receive Buffer (infinite) | TP Receive Buffer Record (length 8) |  |
|  | Data | LL |  |  |  |  |  |  | Data | (len) |
| (1) | gfedcbA | 7 | g | fedcbA | fe | dcbA | dcbA | dcbA |  |  |
| (2) | ponmlkjiH | 9 | ponm | lkjiHg | lkji | Hgfe | Hgfe | H | gfedbcA | 7 |
| (3) |  |  |  |  | (HS defers sending RU) |  |  |  |  |  |
| (4) | srQ | 3 | s | rQponm | rQponm | lkji | lkji | lkjiH |  |  |
| (5) |  |  |  |  | rQ | ponm | ponm | p | onmlkjiH | 9 |
| (6) | vuT | 3 | vuTs |  | rQ |  |  |  | p |  |
| (7) | zyxW | 4 | zy | xWvuTs | xWvu | TsrQ | TsrQ | T | srQ | 3 |
| (8) | # | 0 |  | #zy | #zy | xWvu | xWvu | xW | vuT | 3 |
| (9) |  |  |  |  |  | #zy | #zy | # | zyxW | 4 |
| (10) |  |  |  |  |  |  |  |  | # |  |

Direction of Flow
————————————>

NOTATION:

Read data strings right to left to correspond with the order of flow on the session.

A capital letter represents the start of a logical record
(i.e., the first byte of the LL field.)

# represents the end-of-conversation-message indication.
(This is actually coded in the RH, which is not shown in this example.)

Parenthesized numbers and letters identify rows and columns for explanations in the text.

Figure 2-26. Internal Buffering in LU Send/Receive Data Operations (Example)

<u>HS-to-HS</u> <u>via</u> <u>PC</u> is an RU transmitted over the path control network.

Column (D)

<u>HS-to-PS</u> <u>record</u> is a received RU sent from HS to PS.

Column (E)

<u>PS</u> <u>Receive</u> <u>Buffer</u> is an unbounded buffer for accumulating received data from HS.

<u>TP</u> <u>Record</u> is the DATA parameter buffer of the RECEIVE verb (of length 8 in this example).

This example assumes that the FILL parameter of the receive verb has the value LL. The buffer and record sizes were selected to simplify the illustration; typical actual sizes would be much larger, e.g., 256 bytes for the RU size, and up to 32,767 bytes for a TP record.

Notes on the figure:

Row (1)

(A) The sending TP sends a 7-byte logical record (Abcdefg) to PS.

(B) PS sends the first 6 bytes (its buffer length) to HS (Abcdef) and retains the 7th (g), awaiting more data.

(C) HS at the sender receives the 6 bytes from PS and sends 1 RU (4 bytes: Abcd) to path control and retains the remaining 2 bytes (ef).

(D) HS at the receiver receives the RU (4 bytes) and sends the data to PS

(E) Meanwhile, the receiving TP issues RECEIVE_AND_WAIT.

PS accumulates the data in its buffer until it has enough to satisfy a TP request, i.e., enough to fill the TP receive buffer or complete a logical record.

Row (2)

(A) The sending TP sends a 9-byte logical record (H...p).

(B) This forces another 6-byte buffer from PS (g...l); PS retains the remaining 4 bytes (m...p).

(C) HS now has 8 bytes; it sends 1 RU (4 bytes: efgH) and retains 4 (ijkl).

(D,E) At the receiving LU, this RU completes the logical record (A...g) at the receiver. PS passes the record to the TP and retains the first byte of the next record (H).

Row (3)

(C) HS at the sender still has exactly enough data accumulated for one more RU (ijkl), but HS does not send this RU until forced by arrival of another byte or an end-of-conversation-message indication. HS always waits with an exactly full RU so it can incorporate any subsequent protocol signals into the RH.

The interpretation of the remaining lines is similar. Highlights are given below.

Row (5)

(E) At the receiver, the second RU received completes the second logical record (H...p) at the receiving PS. But since the receiving TP buffer is only 8 bytes, PS can pass only 8 bytes (H...o) on the current receive verb.

Row (6)

(E) PS at the receiver passes the last byte (p) of the second logical record to the TP on the next receive verb.

Rows (8-9)

(A-C) The end-of-conversation-message indication (#) from the sending TP forces the sending PS and HS to send all residual data in their buffers. This makes one more record available to the receiving TP.

Row (9)

(D,E) When the receiving HS and PS get the end-of-conversation-message indication, they forward all residual data as soon as possible. The TP gets the last logical record.

Row (10)

(E) The receiving TP gets the end-of-conversation-message indication alone on the next receive verb.

## 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 (in some cases, via RM) for transmission to the partner LU. When the target HS receives the Attach FM header, it passes it to its RM. This RM then creates a PS process and passes it the Attach FM header. The new PS analyzes the Attach FM header, 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.

### Initiating the Initial Local Transaction Program

The first TP activated for a distributed transaction is initiated in a way that appears to the TP as though it were invoked as a target TP by another source TP. To do this, the source RM behaves as if it had received an Attach: it creates the PS process and generates an Attach FM header to pass to PS. These RM actions are triggered by implementation-defined means such as issuing a local control-operator verb.

PS then loads and calls the TP, which can then issue verbs by calling PS.

### 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. RM then 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. (In some cases, PS sends the Attach FM header to RM rather than directly to HS; RM then sends it to HS when bidding for the session.)

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 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 LNS activate a new 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. The LU that activated it (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 LNS 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 LNS receives a session activation request from RM, LNS sends an INITIATE session-services RU, containing the partner LU name, to its control point, using the CP-LU session.

When the control point receives the INITIATE, it translates the LU name into a network address.

The CP then sends a CINIT RU, which contains the network address, the cryptographic key if session cryptography is used, and a description of other characteristics for the session, to the LU that is to activate the session. (The LU that activates a session is called the primary LU [PLU]. The PLU is not necessarily the LU that requested session initiation.)

Session Activation: LNS for the PLU receives the CINIT and retains the address. Using information from the CINIT and from the LU's mode table for the requested mode, LNS then generates a BIND session-control RU containing the desired session parameters and sends it to its local PU for routing to the partner LU.

LNS for the LU receiving BIND (the secondary LU or SLU) negotiates the proposed session parameters to acceptable values and sends a positive response to BIND via its local PU.

(If the LUs cannot agree on session parameters, the session activation fails.)

When the positive response to BIND is sent or received, the LNS 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 LNS notifies its RM that a new session is available.

### Session Outage

If session outage occurs, LNS notifies RM. If a conversation was active on the session, RM notifies PS, which notifies the transaction program of conversation failure. RM requests LNS 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 only upon specific request from the control-operator transaction program.

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 LNS to deactivate the session.

Session Deactivation: When LNS receives a session-deactivation request from RM, it sends UNBIND, via the local PU, and awaits a response. When the partner LNS receives an UNBIND, it unconditionally sends a positive response. When the response to UNBIND is sent or received, the corresponding LNS 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
- Buffers conversation-message data from the transaction program into contiguous blocks for efficient subdivision by HS
- Reblocks RU 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:

- Reblocks data from PS into RU-sized units
- 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
- Generates and enforces sequence numbering to detect lost or duplicate BIUs
- Provides session-level pacing
- 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
- 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 LNS to activate and deactivate sessions

### LU Network Services

LNS 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
- Notifies RM of session outage
- Notifies the control point of LU characteristics and conditions during LU initialization (ACTLU exchange)
- Creates and destroys half-session instances and connects them to path control instances

## 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 (SNADS) 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.

SNADS 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):

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

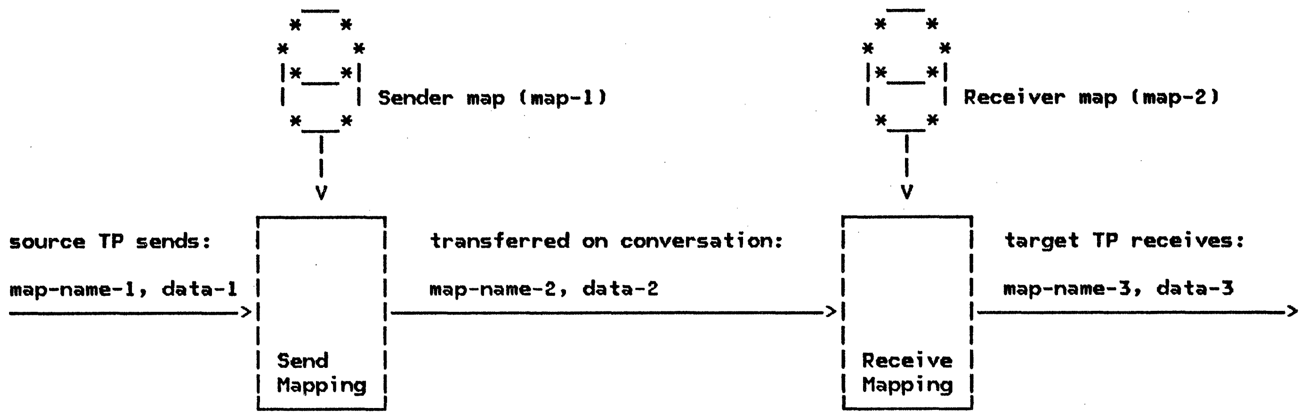


Figure 2-27. Map Name Usage by Mapped Conversations

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 "Appendix H. FM Header and LU Services Commands"), 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.

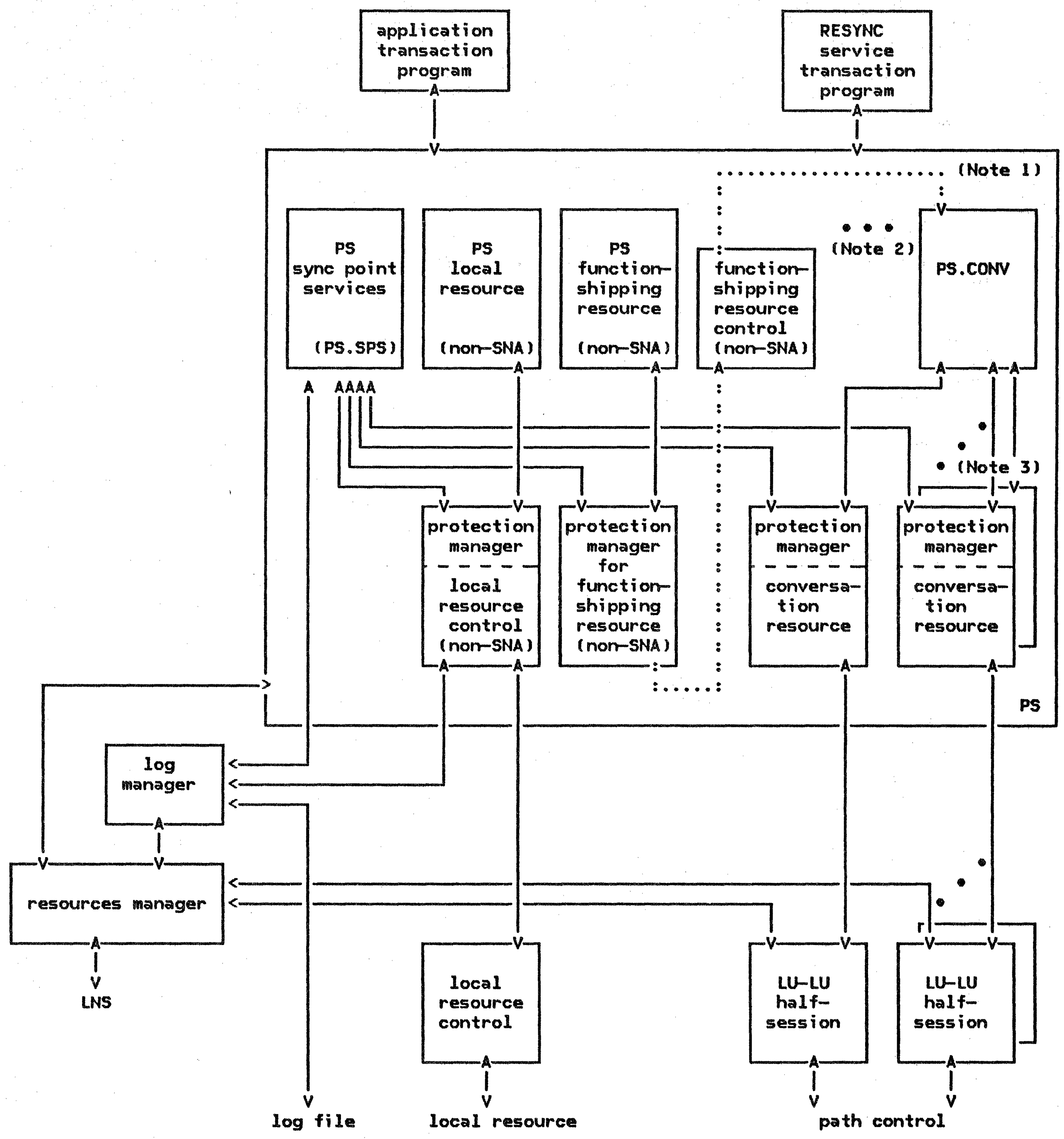
## Sync Point Function

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 2-38.


application transaction program
RESYNC service transaction program
(Note 1)
PS sync point services (PS.SPS)
PS local resource (non-SNA)
PS function-shipping resource (non-SNA)
function-shipping resource control (non-SNA)
(Note 2)
PS.CONV
(Note 3)
protection manager
local resource control (non-SNA)
protection manager for function-shipping resource (non-SNA)
protection manager
conversation resource
protection manager
conversation resource
PS
log manager
resources manager
LNS
local resource control
LU-LU half-session
LU-LU half-session
log file
local resource
path control

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

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. RM also logs conversation allocations, thereby supplementing the work of the conversation protection manager.

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.

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 "Appendix H. FM Header and LU Services Commands" for the SNA-defined format of the EXCHANGE_LOG_NAME and COMPARE_STATES GDS variables.)

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 control-points that serve this LU, the partner LUs with which this 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 and mapping) that it supports.

An CPLU_CAPABILITY structure describes a control point. This information includes the allowed formats of addresses and the set of session-services RUs used on the LU-CP session.

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. The PARTNER_LU structure also contains a list of mode descriptions.

A MODE structure describes a mode. This information includes the mode name 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.

## 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.

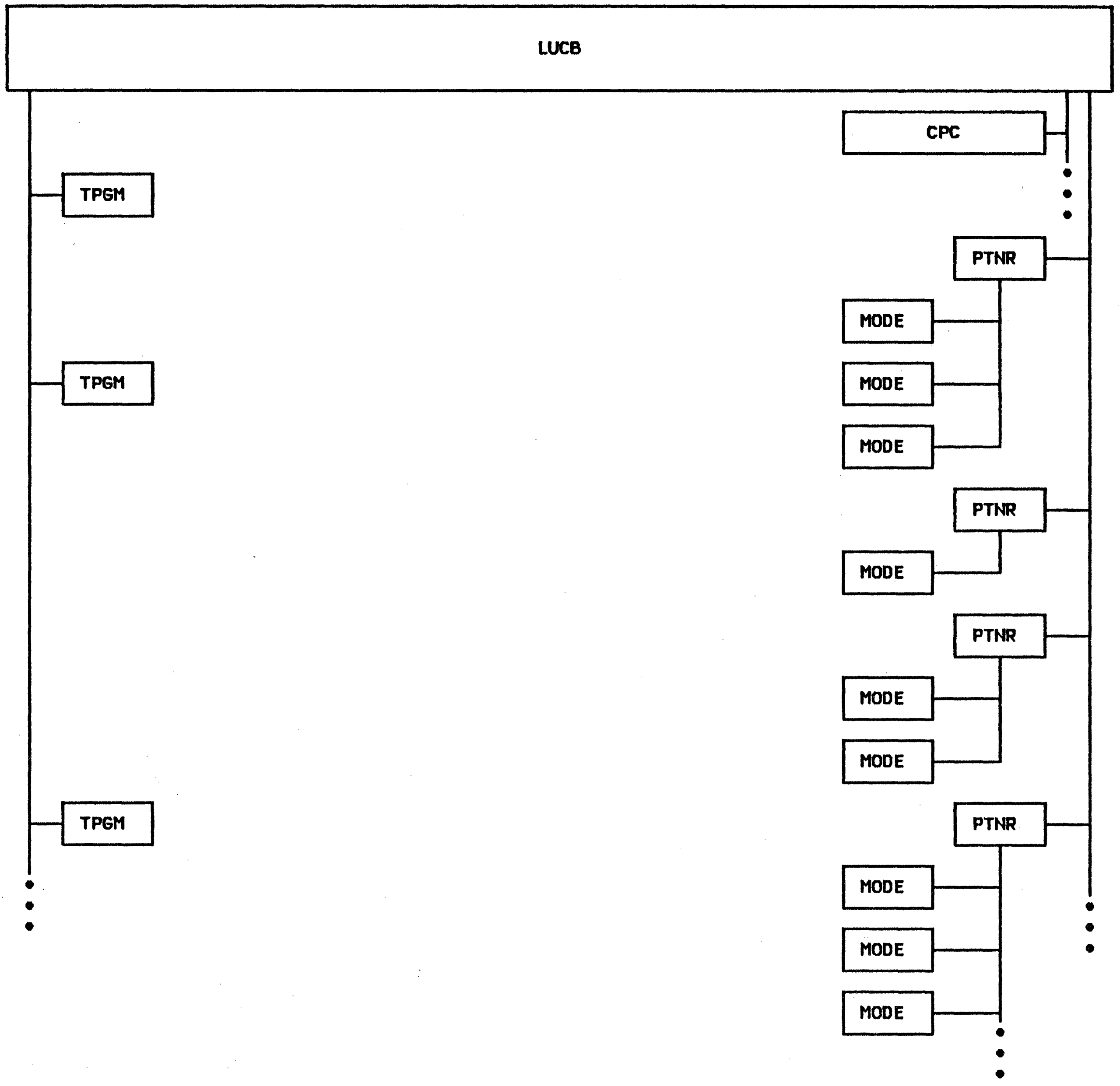
The TP process represents a transaction program instance. It identifies the transaction program code that it is using. There may be multiple transaction program processes executing the same transaction program code.

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 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:

- An active session is associated with the control-point (CPC).



LEGEND:
Vertical lines represent lists of subordinate resources

| Abbr. | Data Structure Name |
|---|---|
| LUCB: Local LU information | (LUCB) |
| TPGM: Transaction Program Code information | (TRANSACTION_PROGRAM) |
| CPC: Control Point information | (CPLU_CAPABILITY) |
| PTNR: Partner LU information | (PARTNER_LU) |
| MODE: Mode information | (MODE) |

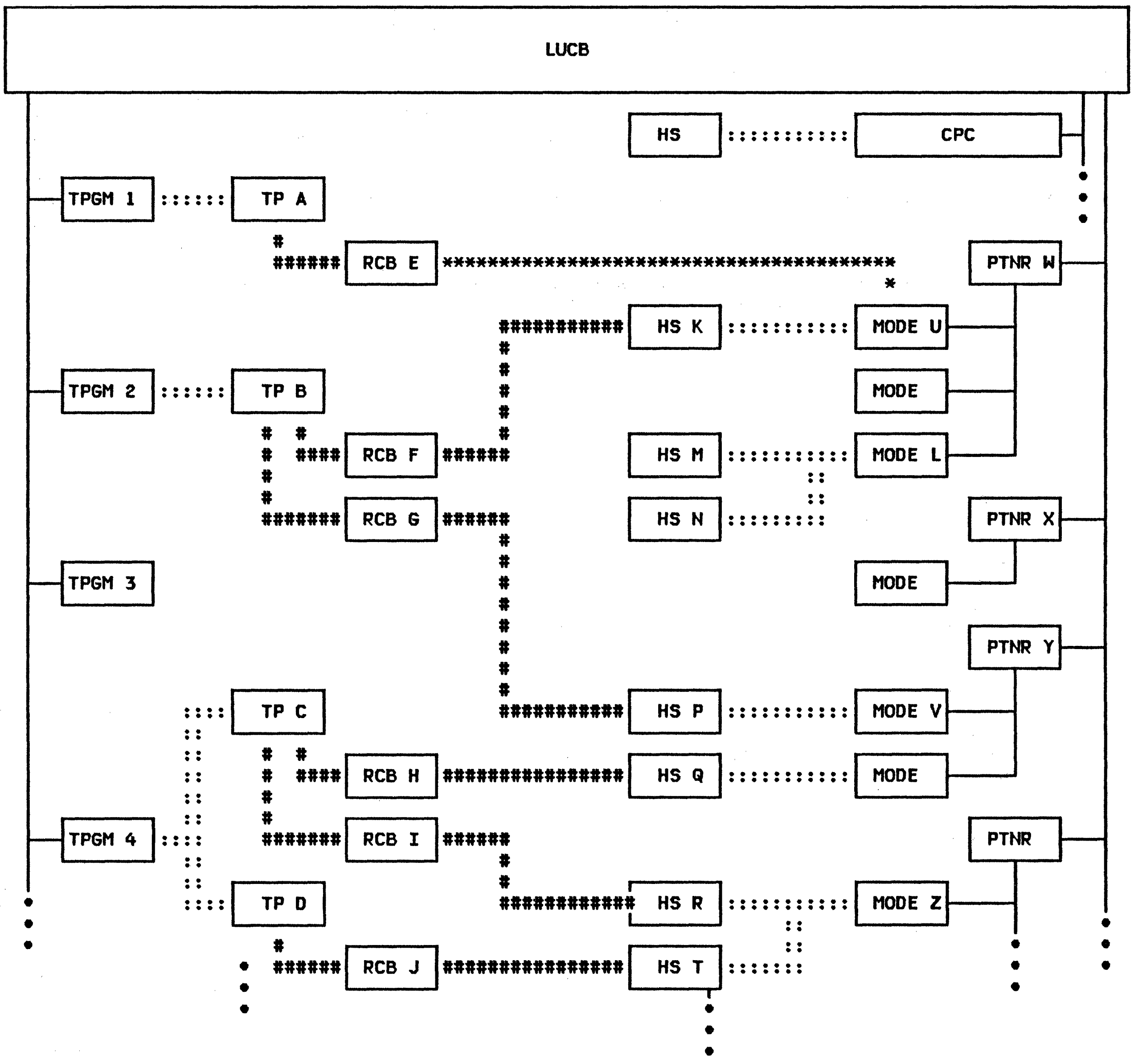
Figure 2-29. LU Static Data Structures (Example)

(This session is used directly by LU internal components, so no relationship to a transaction program is shown.)

- RCB E associates active TP A for transaction program code 1 with mode name U, awaiting a free session with mode name U.

- 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.



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) |
| CPC: | Control Point information | (CPLU_CAPABILITY) |
| 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)

- 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 four phases: creating the LU processes, activating the CP-LU session, initiating the control operator transaction program, and setting the LU parameters 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 steps to terminate communications include resetting the session limits and deactivating the CP-LU session.

#### LU PROCESS CREATION AND TERMINATION

Figure 2-32 on page 2-45 shows the process creation and termination hierarchy for the LU.

First, the PU in the node creates two dynamic processes, RM and LNS. These processes continue running thereafter.

The PU creates the CP-LU half-session when it receives ACTLU session-control RU from the CP (see "CP-LU Session Activation").

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

#### CP-LU SESSION ACTIVATION

The CP in the network (the PNCP or the SSCP) activates the CP-LU session for the LU by sending ACTLU, to which LNS responds, if ready, with +RSP(ACTLU). This session activation is required prior to any LU-LU session initiation or termination.

When the CP determines that no further session initiation or termination activity is required, it deactivates the CP-LU session by sending DACTLU to the LU.

If the CP-LU session is interrupted because of session outage, the CP attempts to reactivate it. This need not interrupt normal LU-LU session traffic.

#### CONTROL-OPERATOR TRANSACTION PROGRAM INITIATION

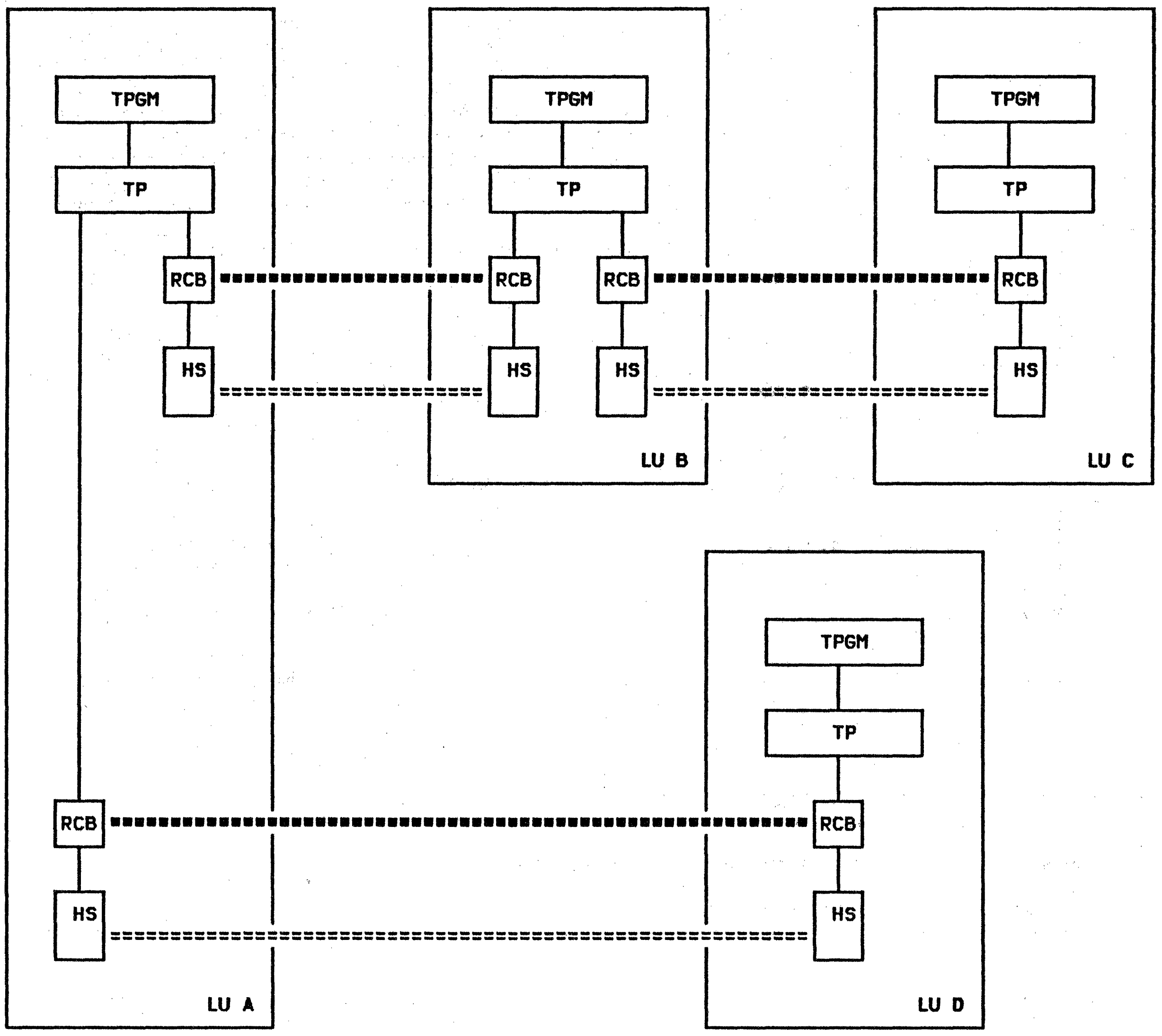
RM creates a PS process and initiates the control-operator TP.

#### CONTROL-OPERATOR ACTIONS

The control operator specifies the LU parameters describing the LU-accessed network resources: the control points, transaction programs, partner LUs, and modes. (An implementation might provide this function without requiring explicit operator interaction, e.g., the LU parameters might be defined 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 LNS deactivate them.



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)

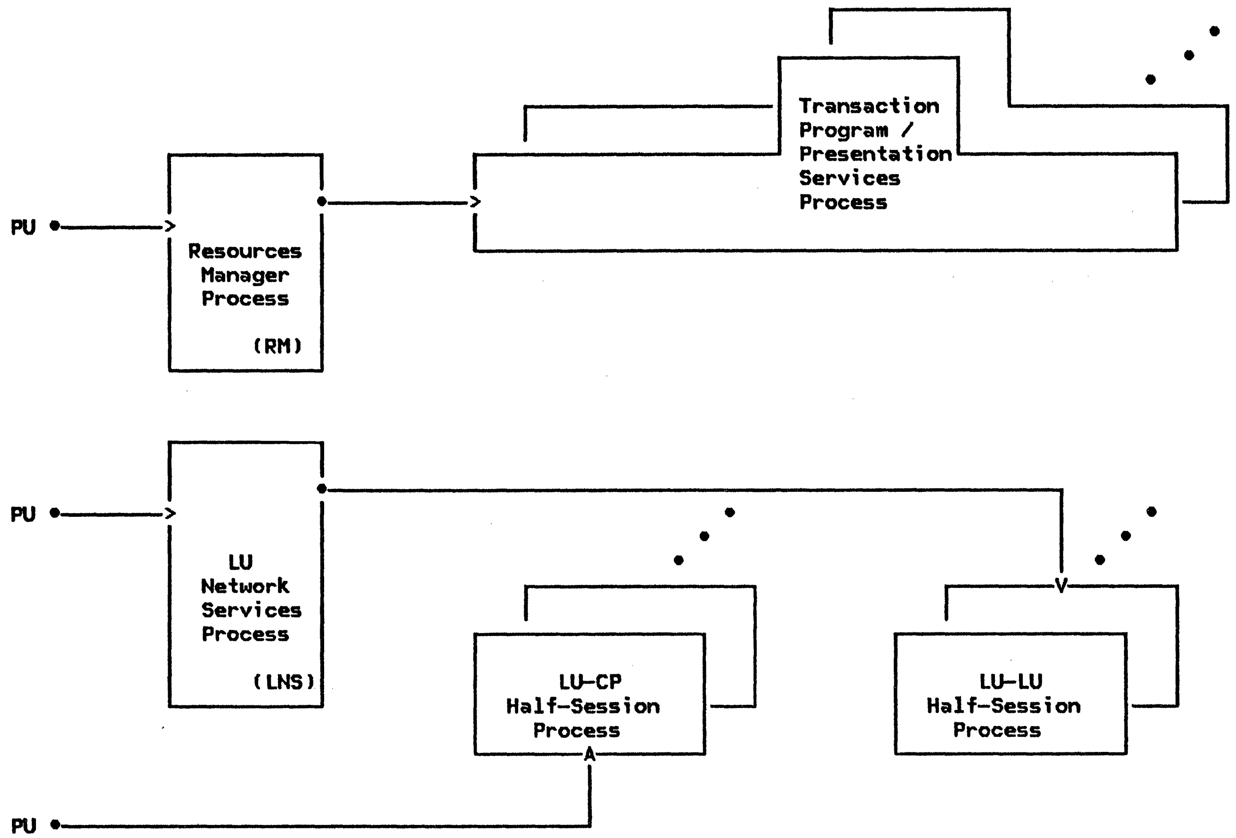
The LU initiates no further actions to shut down the LU. Any further actions are at the initiative of the CP or the PU.

RUNNING STATE

Once the CP-LU session has been activated and 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; 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, LNS performs the relevant processing. LNS creates an HS process for an LU-LU session and connects it to a path control instance whenever it sends or receives BIND. LNS destroys that process when it has sent or received a positive response to UNBIND, has disconnected the half-session from path control (by sending PS_HS_DISCONNECT), and has notified the CP


PU
Resources Manager Process
(RM)
Transaction Program / Presentation Services Process
PU
LU Network Services Process
(LNS)
LU-CP Half-Session Process
LU-LU Half-Session Process
PU

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

Figure 2-32. LU Process Creation and Termination Hierarchy

that the session is ended (by sending SESSEND).

EXAMPLE

Figure 2-35 on page 2-50 and Figure 2-36 on page 2-51 illustrate typical interactions at the local and remote LUs, respectively, for an LU shutdown sequence. LU startup and shutdown are described in more detail in "Chapter 5.4. Presentation Services--Control-Operator Verbs" .

## PROTOCOL BOUNDARY SUMMARY

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

### EXTERNAL PROTOCOL BOUNDARY VERBS AND MESSAGE UNITS

#### PS-TP Protocol Boundary: Transaction Program Verbs

TRANSACTION_PGM_VERB

Basic-Conversation Verb Variants

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

Mapped-Conversation Verb Variants

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_REQUEST_TO_SEND
MC_SEND_DATA
MC_SEND_ERROR
MC_TEST

Control-Operator Verb Variants

ACTIVATE_SESSION
CHANGE_SESSION_LIMIT
DEACTIVATE_SESSION
INITIALIZE_SESSION_LIMIT
PROCESS_SESSION_LIMIT
RESET_SESSION_LIMIT

#### LNS-PU Protocol Boundary

LNS_TO_NNM_RECORD
ACTLU_RSP_SEND_RECORD
BIND_RQ_SEND_RECORD
BIND_RSP_SEND_RECORD
DACTLU_RSP_SEND_RECORD
HIERARCHICAL_RESET_RSP
PC_CONNECT
PC_HS_CONNECT
PC_HS_DISCONNECT
SESSION_ROUTE_INOP_RSP
UNBIND_RQ_SEND_RECORD
UNBIND_RSP_SEND_RECORD

NNM_TO_LNS_RECORD
ACTLU_RQ_RCV_RECORD
BIND_RQ_RCV_RECORD
BIND_RSP_RCV_RECORD
DACTLU_RQ_RCV_RECORD
HIERARCHICAL_RESET
PC_CONNECT_RSP
SESSION_ROUTE_INOP
UNBIND_RQ_RCV_RECORD
UNBIND_RSP_RCV_RECORD

#### HS-PC Protocol Boundary

PC_TO_HS_RECORD
HS_TO_PC_RECORD

### INTER-COMPONENT STRUCTURES

#### PS-HS Protocol Boundary

PS_TO_HS_RECORD

Variants
CONFIRMED
REQUEST_TO_SEND
SEND_DATA_RECORD
SEND_ERROR

HS_TO_PS_RECORD
CONFIRMED
RECEIVE_DATA
RECEIVE_ERROR
REQUEST_TO_SEND
RSP_TO_REQUEST_TO_SEND

#### PS-RM Protocol Boundary

PS_TO_RM_RECORD
ALLOCATE_RCB
CHANGE_SESSIONS
DEALLOCATE_RCB
GET_SESSION
RM_ACTIVATE_SESSION
RM_DEACTIVATE_SESSION
TERMINATE_PS
UNBIND_PROTOCOL_ERROR

RM_TO_PS_RECORD
ATTACH_RECEIVED
CONVERSATION_FAILURE
RCB_ALLOCATED
RCB_DEALLOCATED
RM_SESSION_ACTIVATED
SESSION_ALLOCATED

RM-HS Protocol Boundary

```
RM_TO_HS_RECORD
  BID_RSP
  BID_WITH_ATTACH
  BID_WITHOUT_ATTACH
  BIS_REPLY
  BIS_RQ
  HS_PS_CONNECTED
  RTR_RQ
  RTR_RSP
  YIELD_SESSION

HS_TO_RM_RECORD
  ATTACH_HEADER
  BID
  BID_RSP
  BIS_RQ
  BIS_REPLY
  FREE_SESSION
  RTR_RQ
  RTR_RSP
```

RM-LNS Protocol Boundary

```
RM_TO_LNS_RECORD
  ACTIVATE_SESSION
  DEACTIVATE_SESSION

LNS_TO_RM_RECORD
  ACTIVATE_SESSION_RSP
  CTERM_DEACTIVATE_SESSION
  SESSION_ACTIVATED
  SESSION_DEACTIVATED
```

LNS-HS Protocol Boundary

```
LNS_TO_HS_RECORD
  HS_SEND_RECORD
  INIT_HS

HS_TO_LNS_RECORD
  ABORT_HS
  HS_RCV_RECORD
  INIT_HS_RSP
```

## COMPONENT INTERACTIONS AND FLOW SEQUENCES

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

Each sequence 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 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 correlate corresponding flows in the facing figures. When flows cross in the path-control network, the crossing is illustrated on the sending side of the delayed flow.

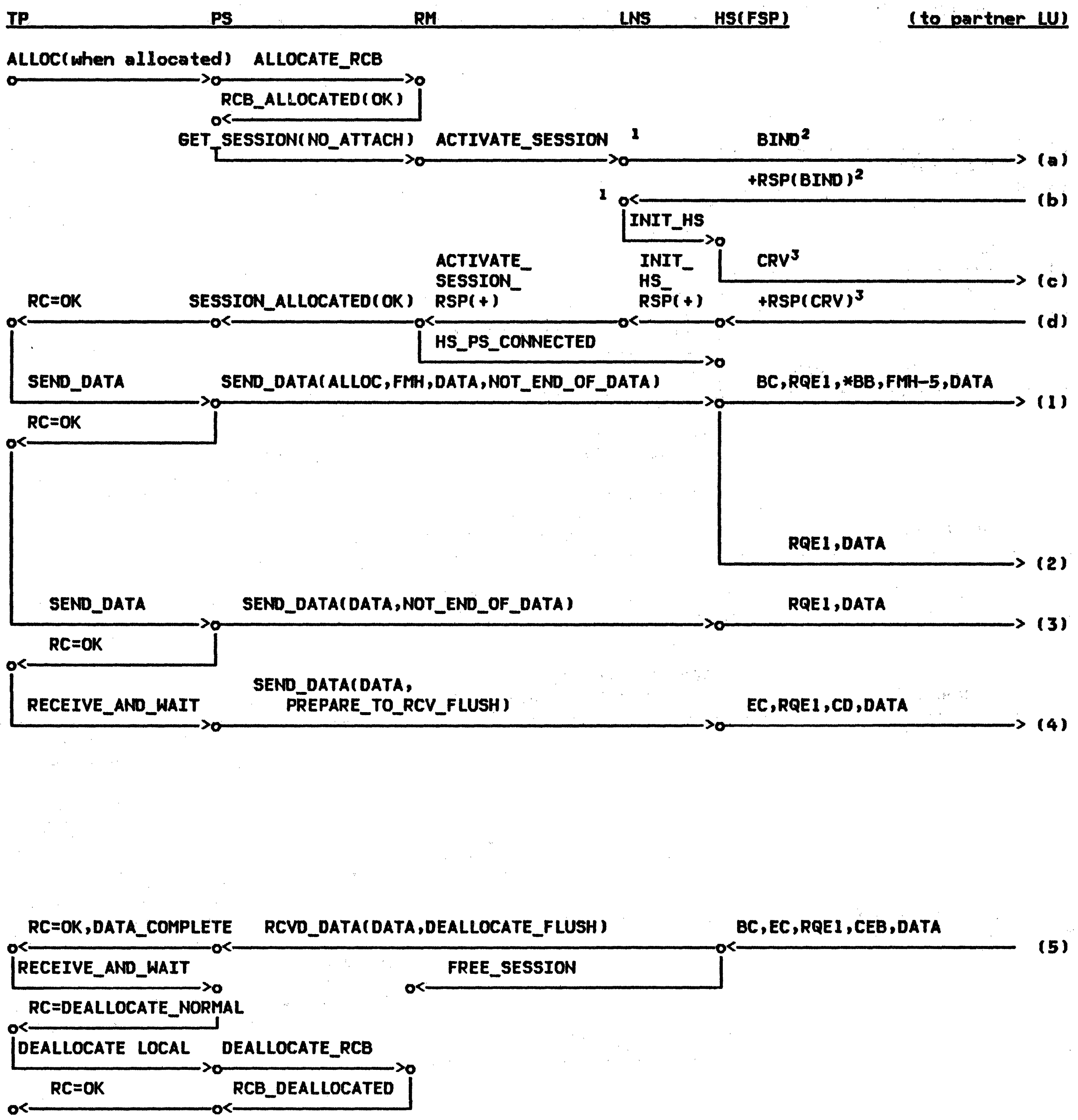
### NOTATION

For the interpretation of labels on the arrows, see the following. (In some cases, these names have been abbreviated.)

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

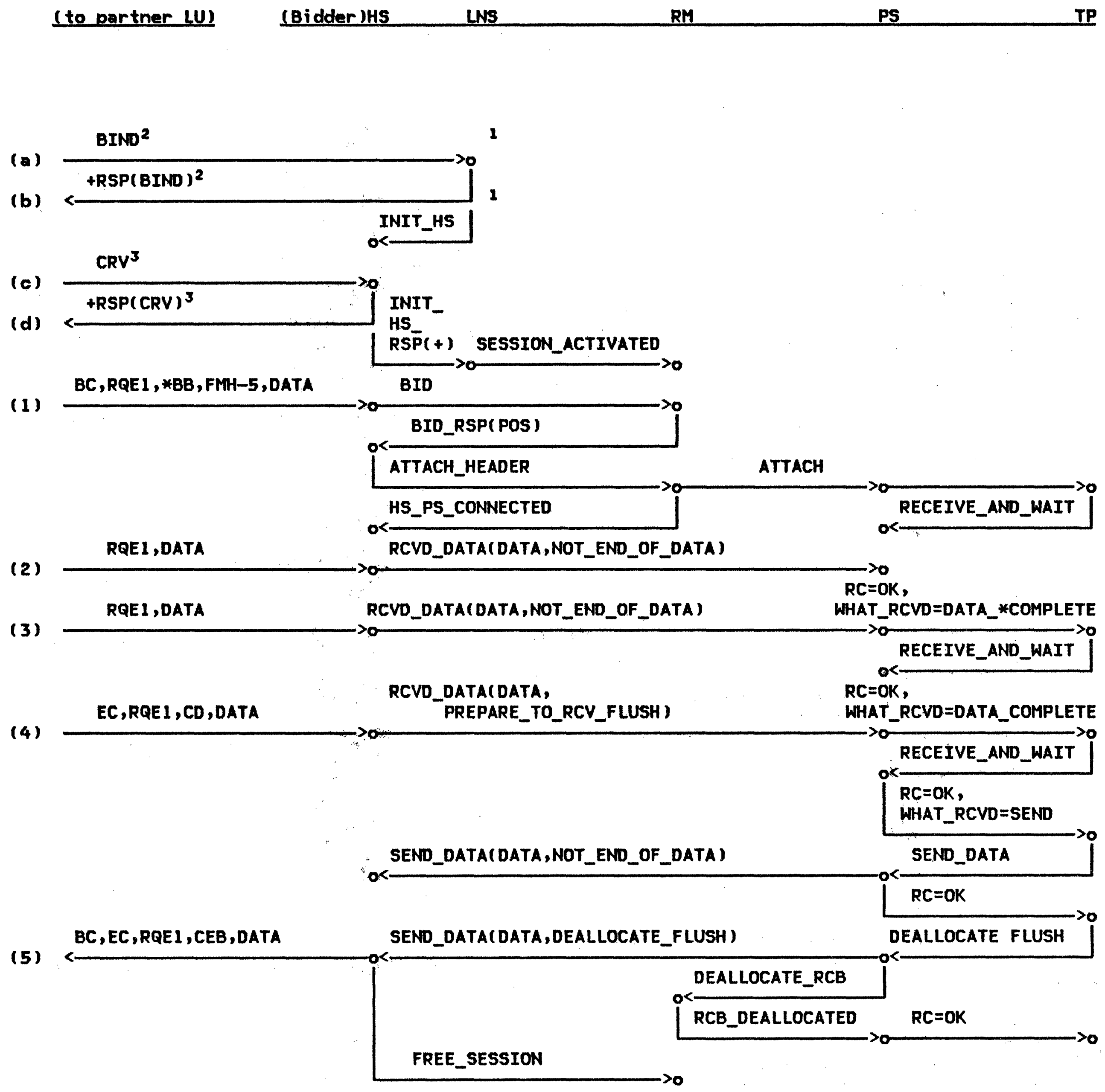
  The following abbreviations for chaining indicators are also used:

  - 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 (LNS-LNS, HS-HS), see "Appendix H. FM Header and LU Services Commands"



NOTES:
[1] Session-activation flows to PU, CP, and path control have been omitted; see "Chapter 4. LU Network Services" for details.
[2] BIND/RSP(BIND) flows through the PU (not shown).
[3] CRV/RSP(CRV) flows only when session-level cryptography is being used.

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



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

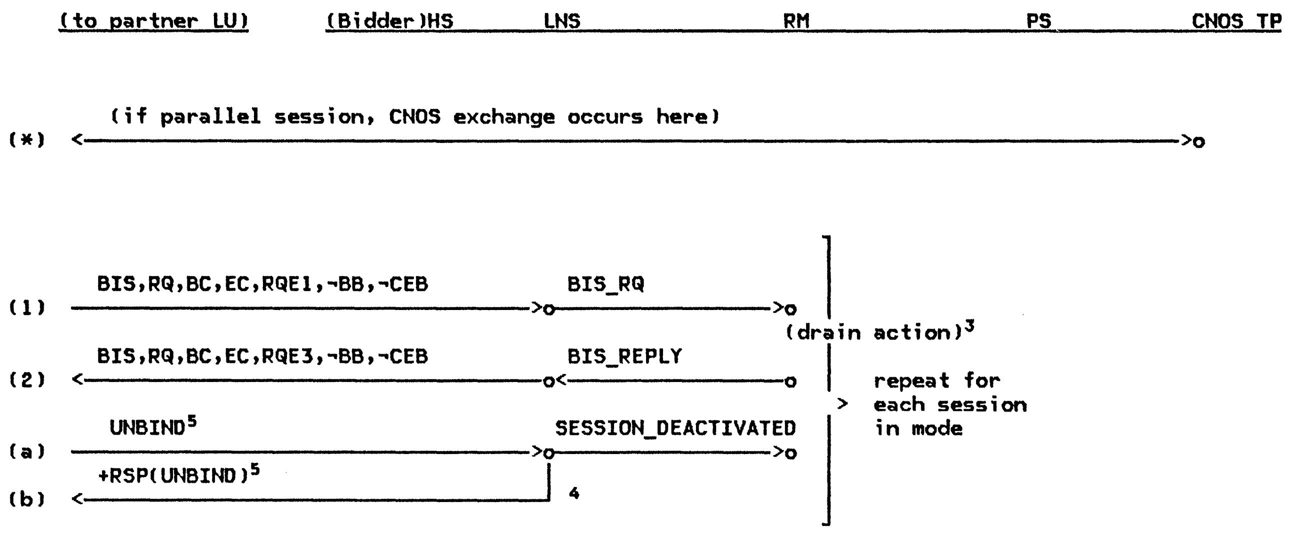
Figure 2-34. Complete Conversation Example--Remote LU

```
COPR TP            PS                   RM                  LNS      HS(FSP)          (to partner LU)

RESET_SESSION_LIMIT¹
o------------------>o
                      (if parallel session, CNOS exchange occurs here)
                    o<-------------------------------------------------------------------------> (*)

                      CHANGE_SESSIONS²
                    o--------------------->o
                                           r
                           (drain action)³
                                           |               BIS_RQ          BIS,RQ,BC,EC,RQE1,¬BB,¬CEB
                                           |  o---------------------------->o------------------------> (1)
                    Repeat for             |
                    each session      <    |               BIS_REPLY       BIS,RQ,BC,EC,RQE3,¬BB,¬CEB
                    for the                |  o<----------------------------o<------------------------ (2)
                    specified              |  |
                    mode name.             |  |DEACTIVATE_SESSION  ⁴          UNBIND⁵
                                           |  L---------------->o-------------------------------------> (a)
                                           |                                 +RSP(UNBIND)⁵
                                           |                  ⁴ o<------------------------------------ (b)
                                           L
```

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 PU and CP have been omitted.
[5] UNBIND/RSP(UNBIND) flows through the PU (not shown)

Figure 2-35. Session Deactivation--Local LU


(to partner LU)
(Bidder)HS
LNS
RM
PS
CNOS TP
(if parallel session, CNOS exchange occurs here)
(*)
BIS,RQ,BC,EC,RQE1,¬BB,¬CEB
BIS_RQ
(1)
(drain action)3
BIS,RQ,BC,EC,RQE3,¬BB,¬CEB
BIS_REPLY
(2)
repeat for each session in mode
UNBIND5
SESSION_DEACTIVATED
(a)
+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 PU (not shown).

Figure 2-36. Session Deactivation--Remote LU

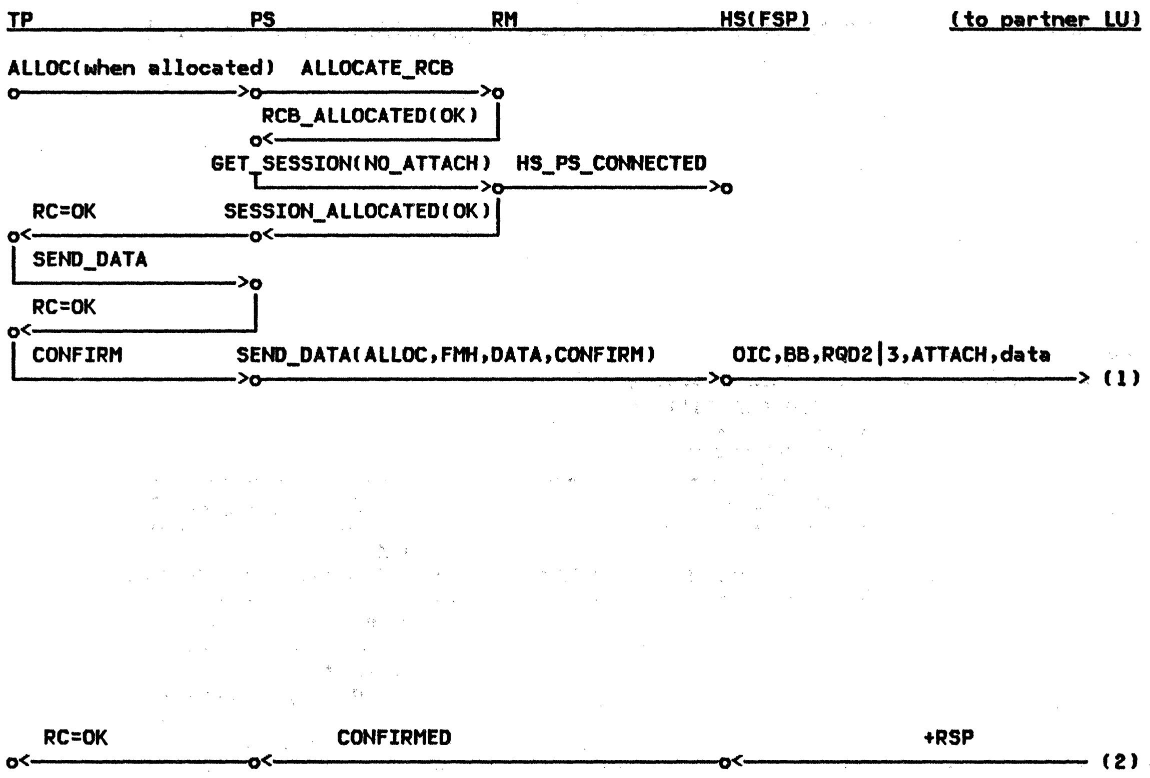


Figure 2-37. ALLOCATE (when allocated), CONFIRM (by First Speaker) --Local LU

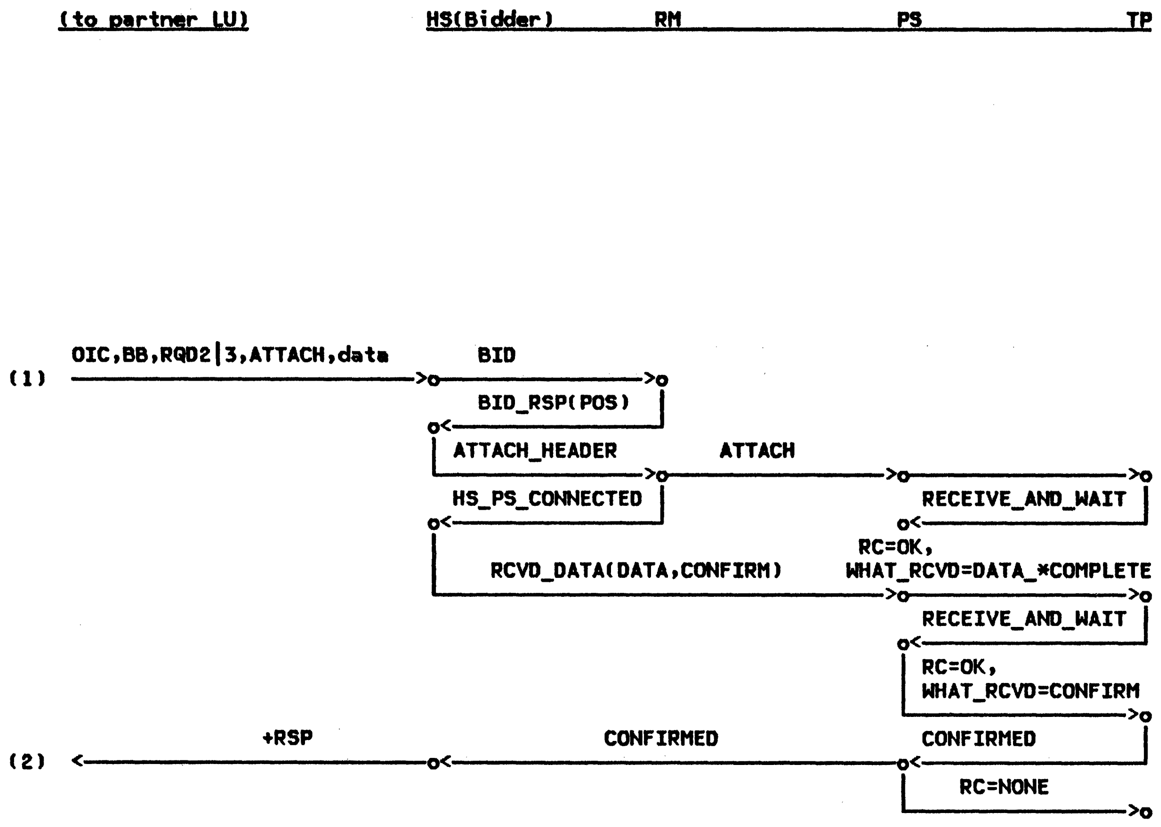


Figure 2-38. ALLOCATE (when allocated), CONFIRM (by First Speaker) --Remote LU

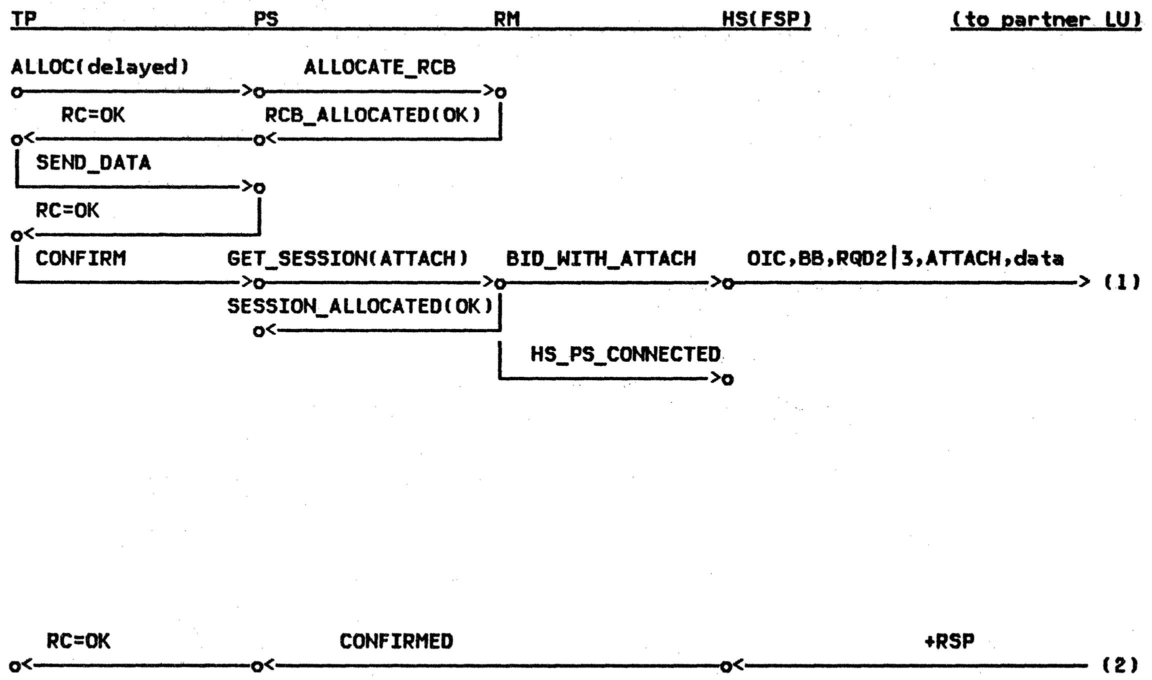


Figure 2-39. ALLOCATE (delayed), CONFIRM (by First Speaker) --Local LU

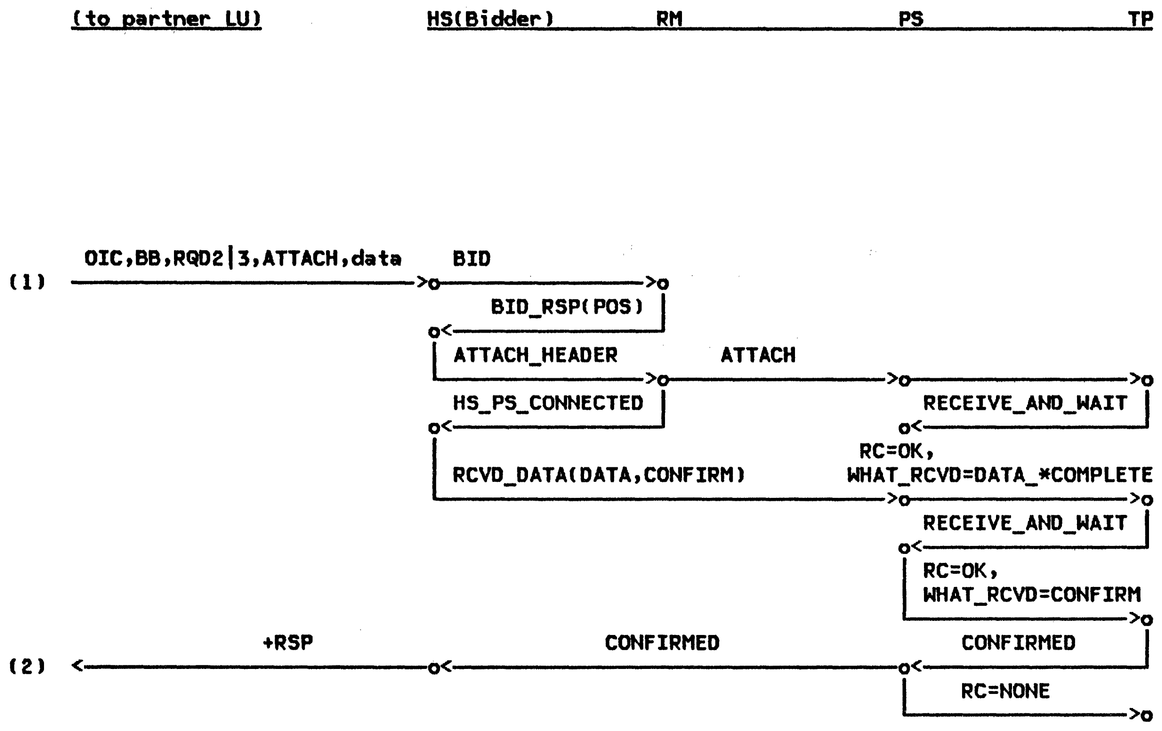


Figure 2-40. ALLOCATE (delayed), CONFIRM (by First Speaker) --Remote LU

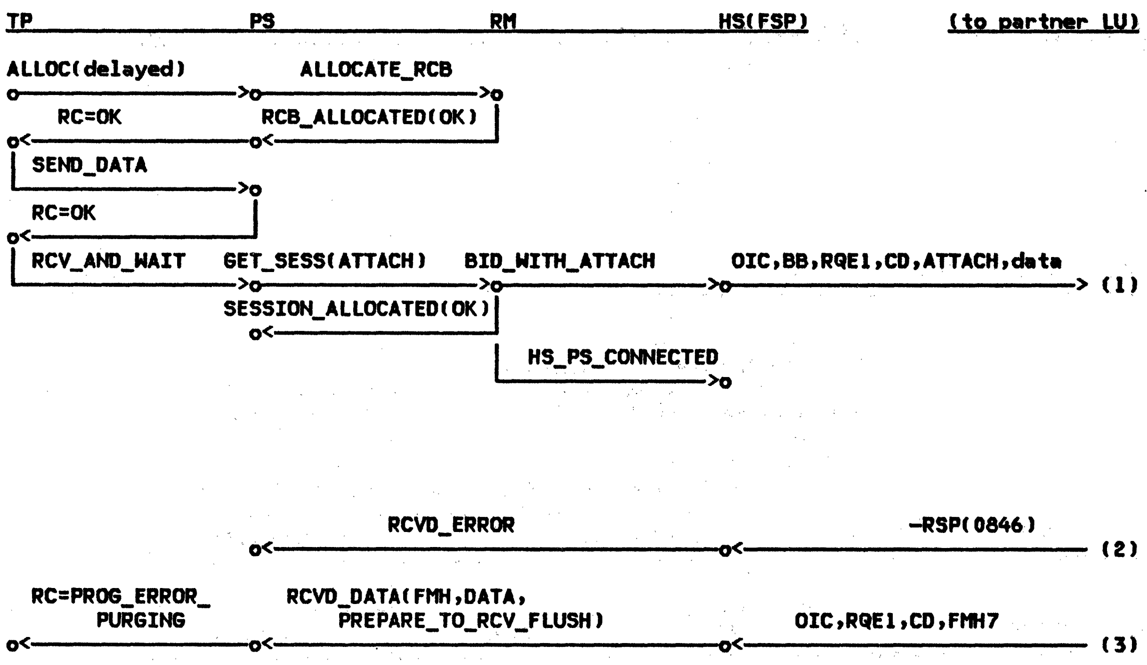


Figure 2-41. ALLOCATE (delayed), RECEIVE_AND_WAIT (by First Speaker) --Local LU

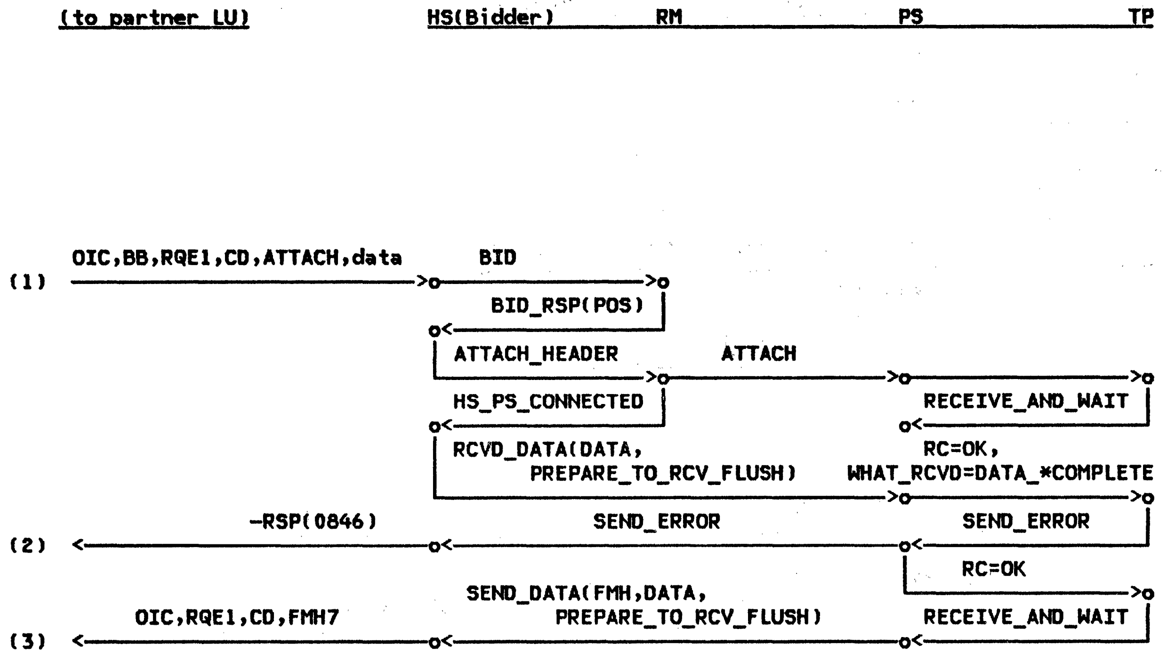


Figure 2-42. ALLOCATE (delayed), RECEIVE_AND_WAIT (by First Speaker) --Remote LU

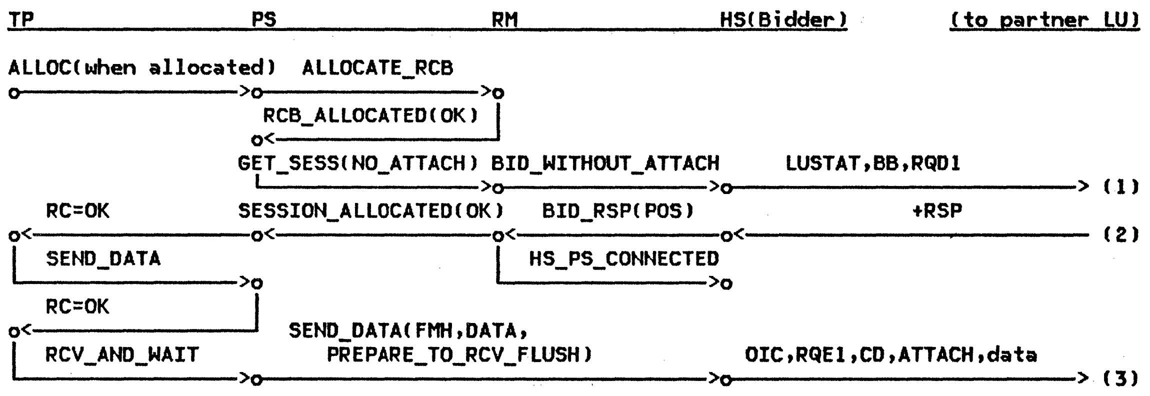


Figure 2-43. ALLOCATE (when allocated), RECEIVE_AND_WAIT (by Bidder) --Local LU

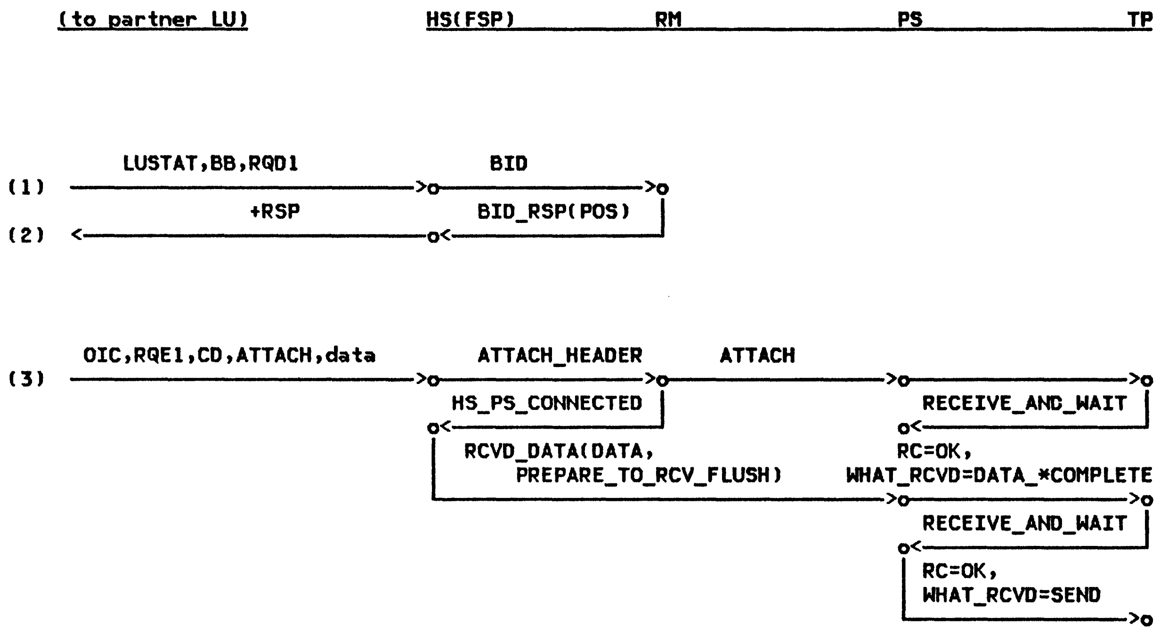


Figure 2-44. ALLOCATE (when allocated), RECEIVE_AND_WAIT (by Bidder) --Remote LU

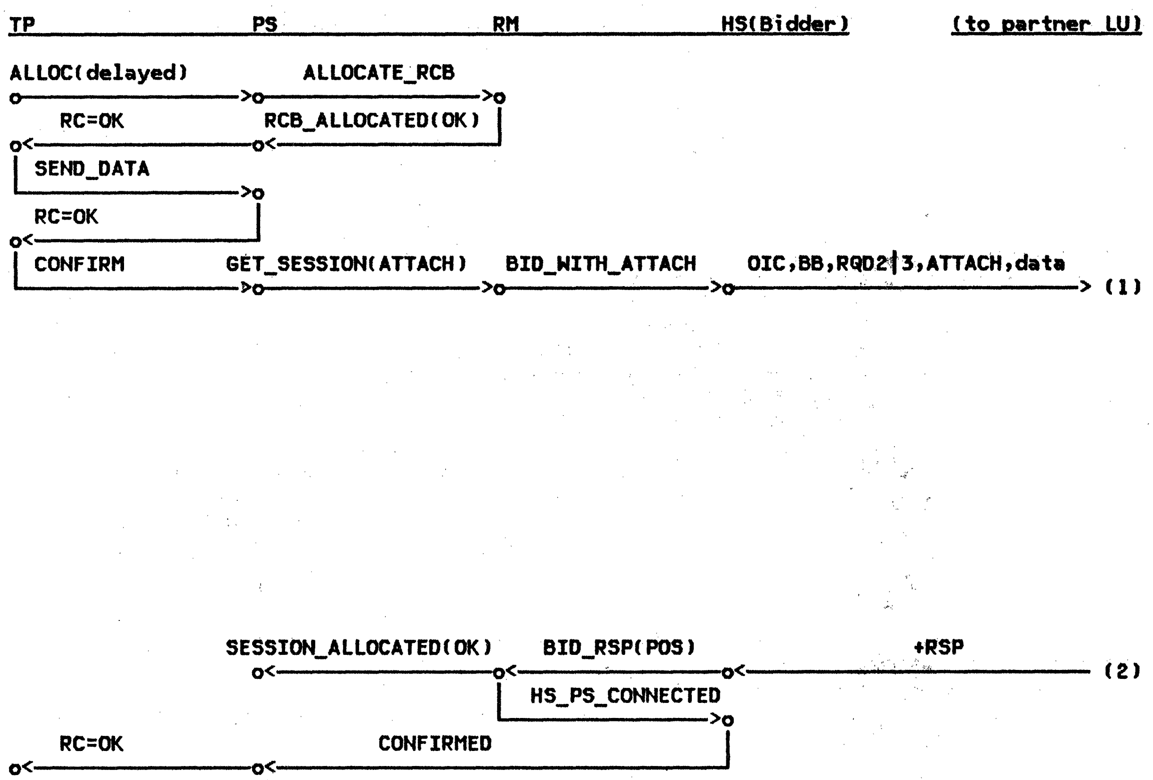


Figure 2-45. ALLOCATE (delayed), CONFIRM (by Bidder) --Local LU

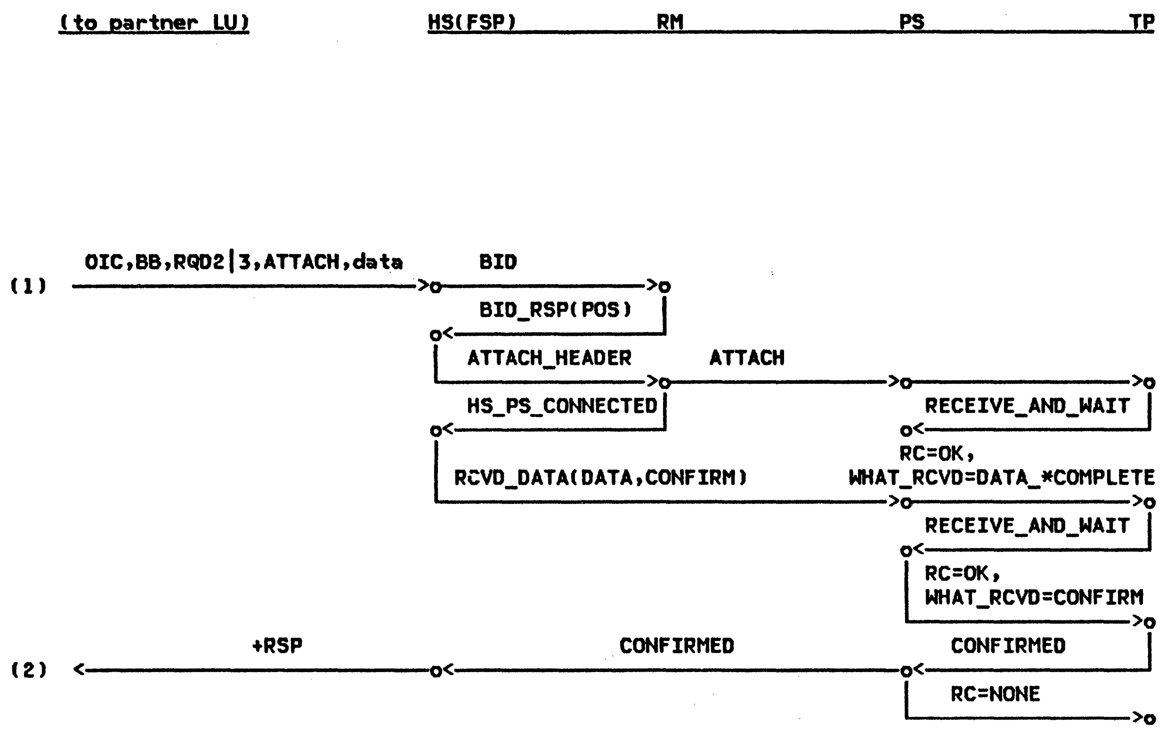


Figure 2-46. ALLOCATE (delayed), CONFIRM (by Bidder) --Remote LU

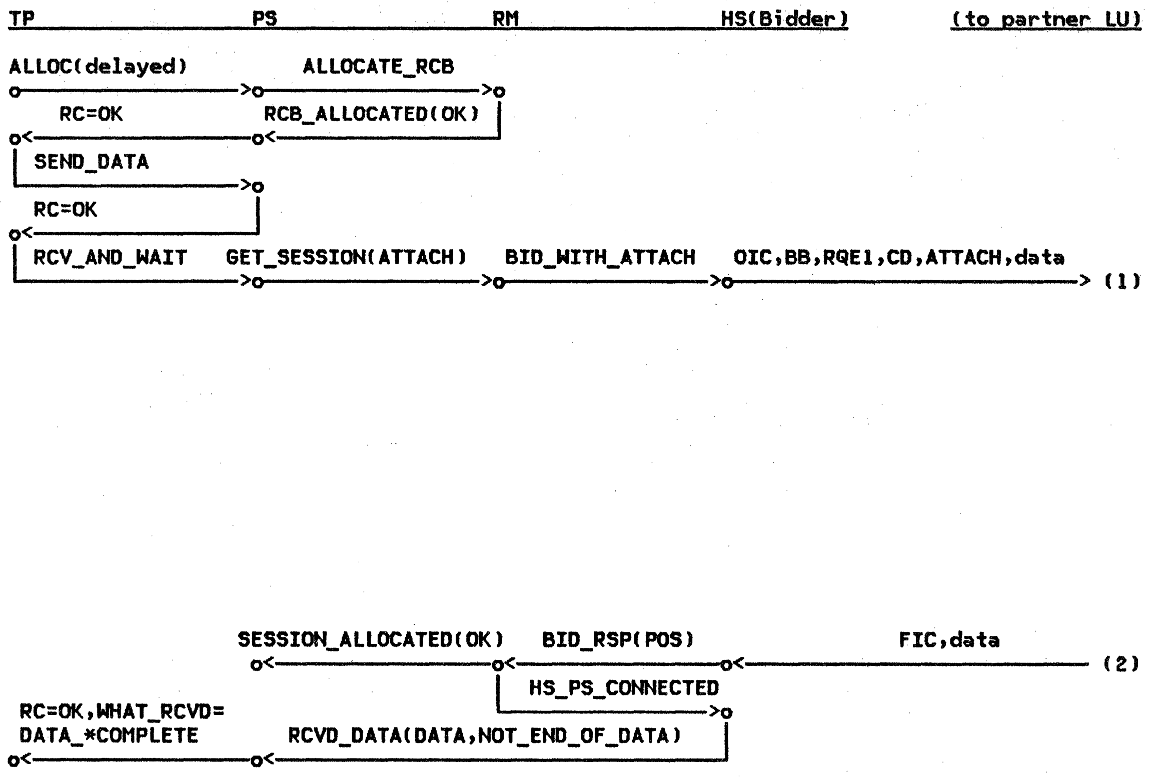


Figure 2-47. ALLOCATE (delayed), RECEIVE_AND_WAIT (by Bidder) --Local LU

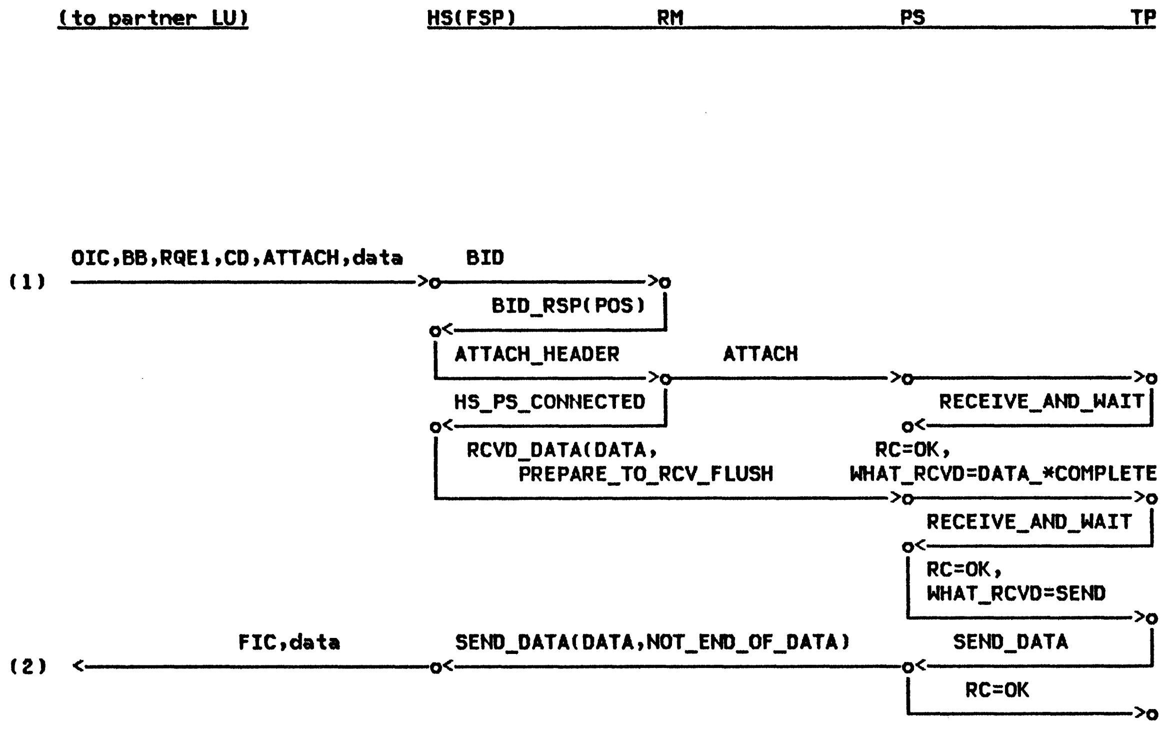


Figure 2-48. ALLOCATE (delayed), RECEIVE_AND_WAIT (by Bidder) --Remote LU

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

ALLOC(delayed)         ALLOCATE_RCB
o----------------->o------------------>o
    RC=OK           RCB_ALLOCATED(OK) |
o<-----------------o<------------------+
| SEND_DATA
+----------------->o
  RC=OK            |
o<-----------------+
| CONFIRM        GET_SESSION(ATTACH)  BID_WITH_ATTACH   OIC,BB,RQD2|3,ATTACH,data
+----------------->o------------------>o----------------->o----------------------------> (1)




                   SESSION_ALLOCATED(OK)  BID_RSP(POS)            -RSP(0846)
                   o<-----------------o<----------------o<---------------------------- (2)
                                      | HS_PS_CONNECTED
                                      +---------------->o
                             RCVD_ERROR                 |
                   o<-----------------------------------+

                       RCVD_DATA(FMH,DATA,
RC=ALLOCATION_ERROR         DEALLOCATE_FLUSH)                OIC,CEB,RQE1,FMH7
o<-----------------o<-----------------------------------o<---------------------------- (3)
|DEALLOCATE_LOCAL   DEALLOCATE_RCB       FREE_SESSION   |
+----------------->o------------------>o<---------------+
   RC=OK             RCB_DEALLOCATED  |
o<-----------------o<------------------+
```

Figure 2-49. ALLOCATE (delayed), CONFIRM (by Bidder), Attach Error --Local LU

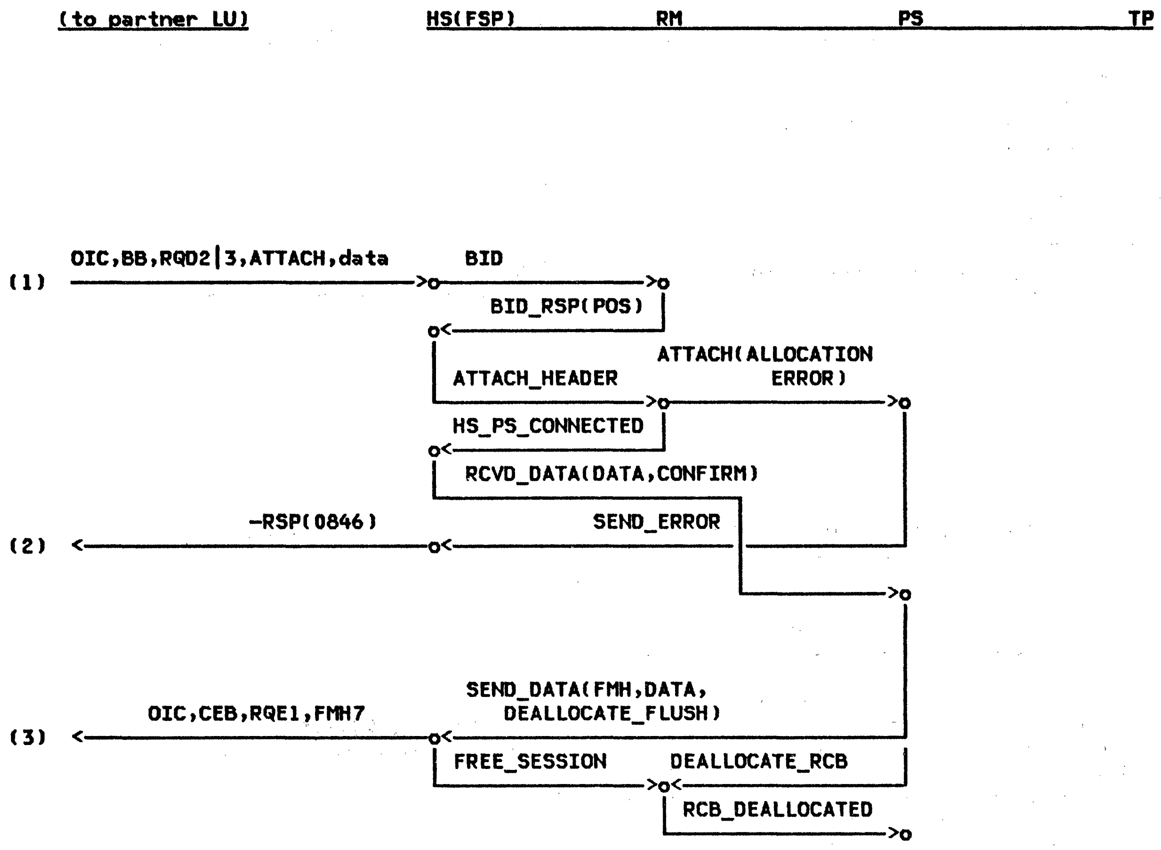


Figure 2-50. ALLOCATE (delayed), CONFIRM (by Bidder), Attach Error --Remote LU

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

ALLOCATE(immediate)  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(when allocated) case.]

Figure 2-51. ALLOCATE (immediate), Successful --Local LU

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

                              (no activity at remote LU)



                   from here on just like ALLOCATE(when allocated)
```

Figure 2-52. ALLOCATE (immediate), Successful --Remote LU

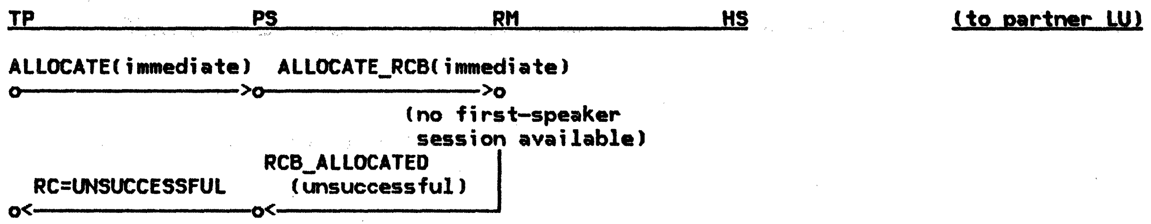


Figure 2-53. ALLOCATE (immediate), Unsuccessful --Local LU

```
(to partner LU)          HS__________RM__________PS__________TP


                              (no activity at remote LU)
```

Figure 2-54. ALLOCATE (immediate), Unsuccessful --Remote LU

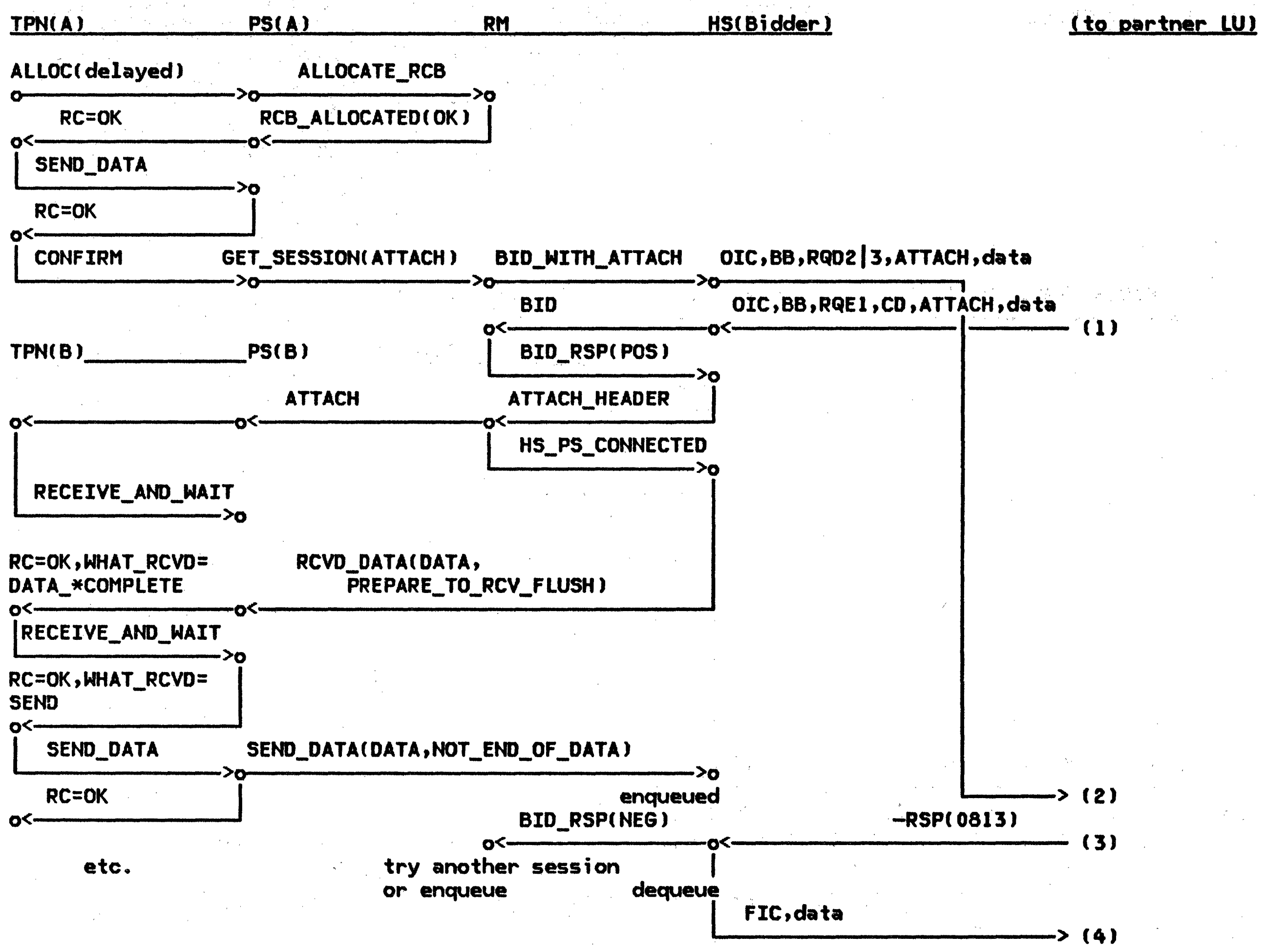


Figure 2-55. ALLOCATE (delayed) Race, Bracket Rejected --Bidder LU

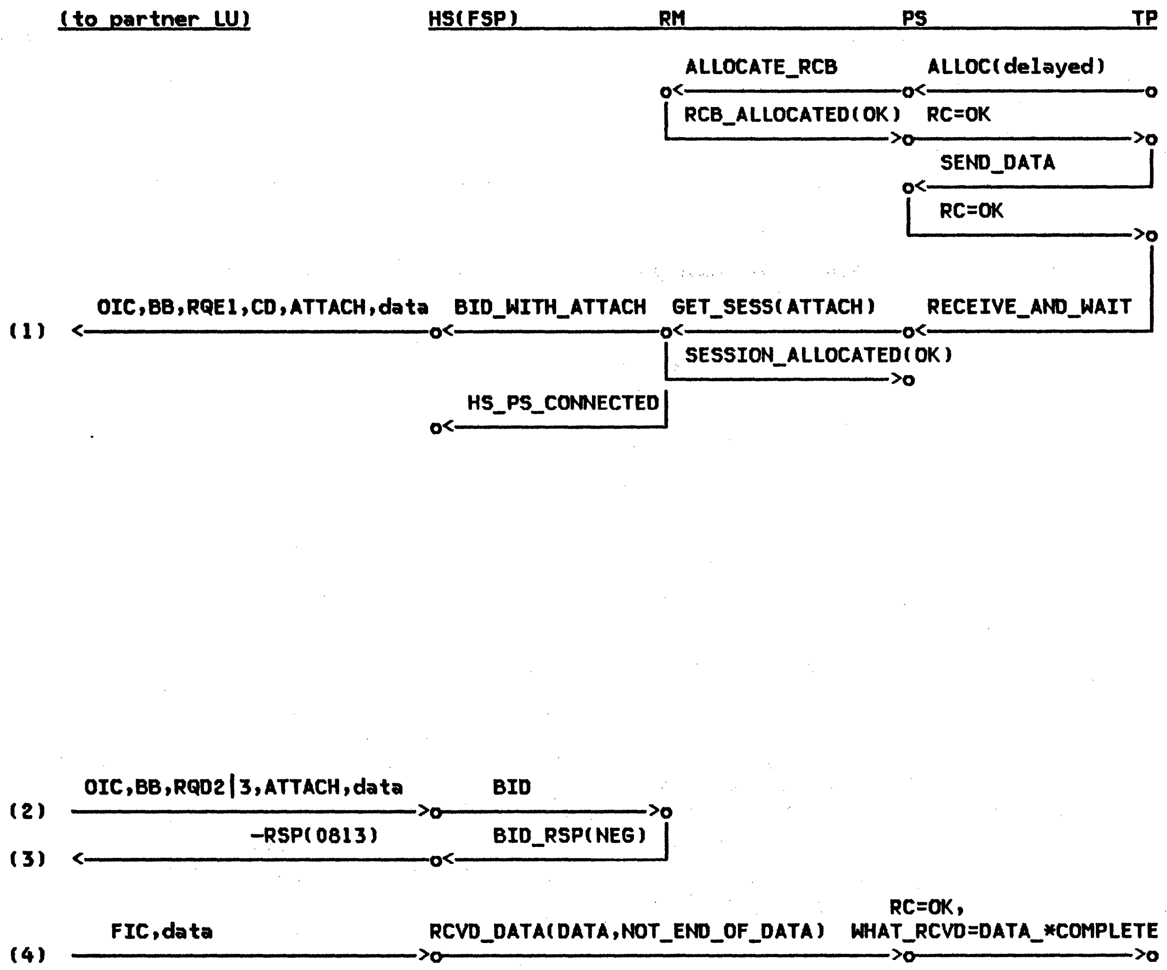


Figure 2-56. ALLOCATE (delayed) Race, Bracket Rejected --First Speaker LU

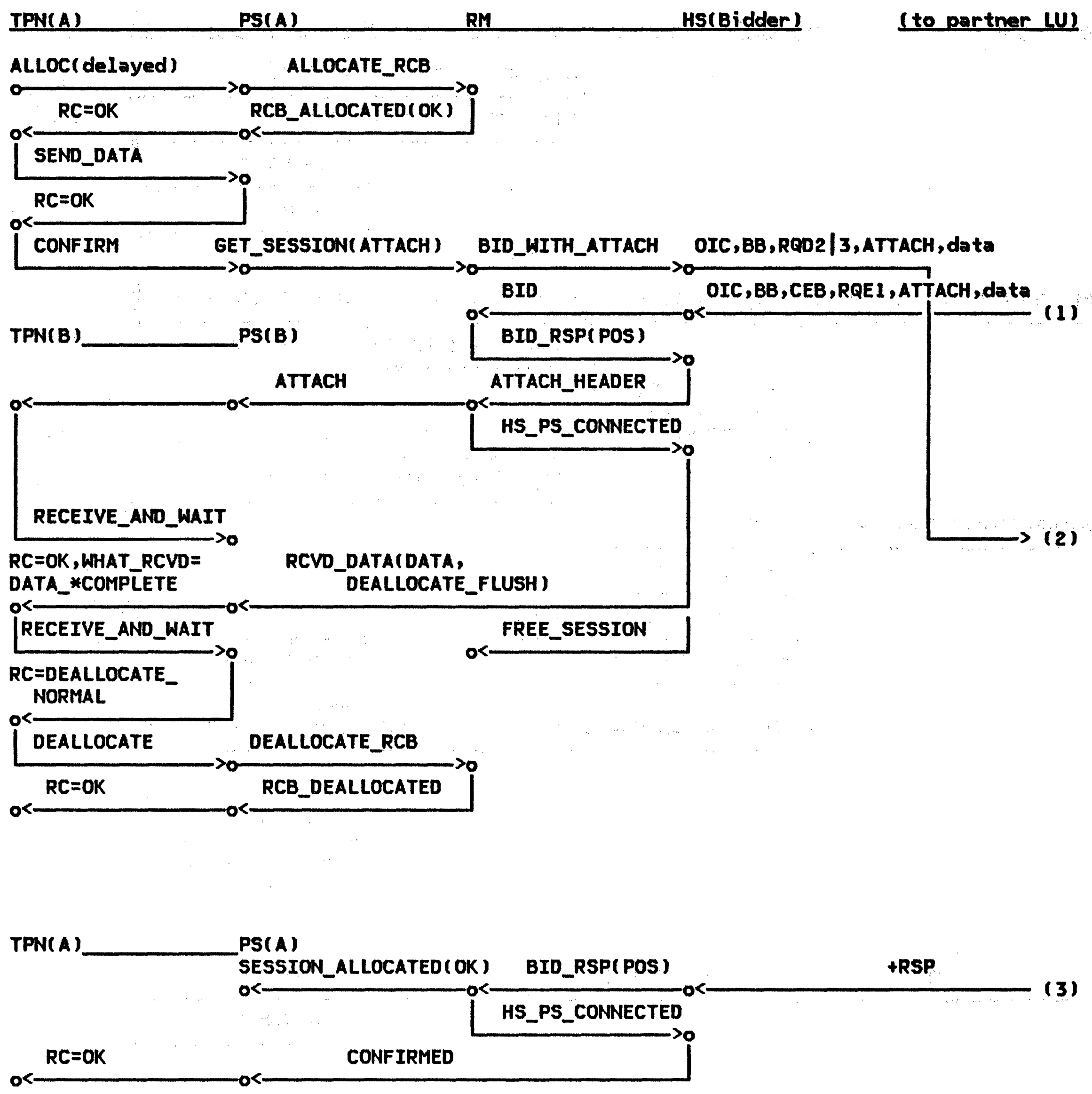


Figure 2-57. ALLOCATE (delayed) Race, Bracket Accepted --Bidder LU

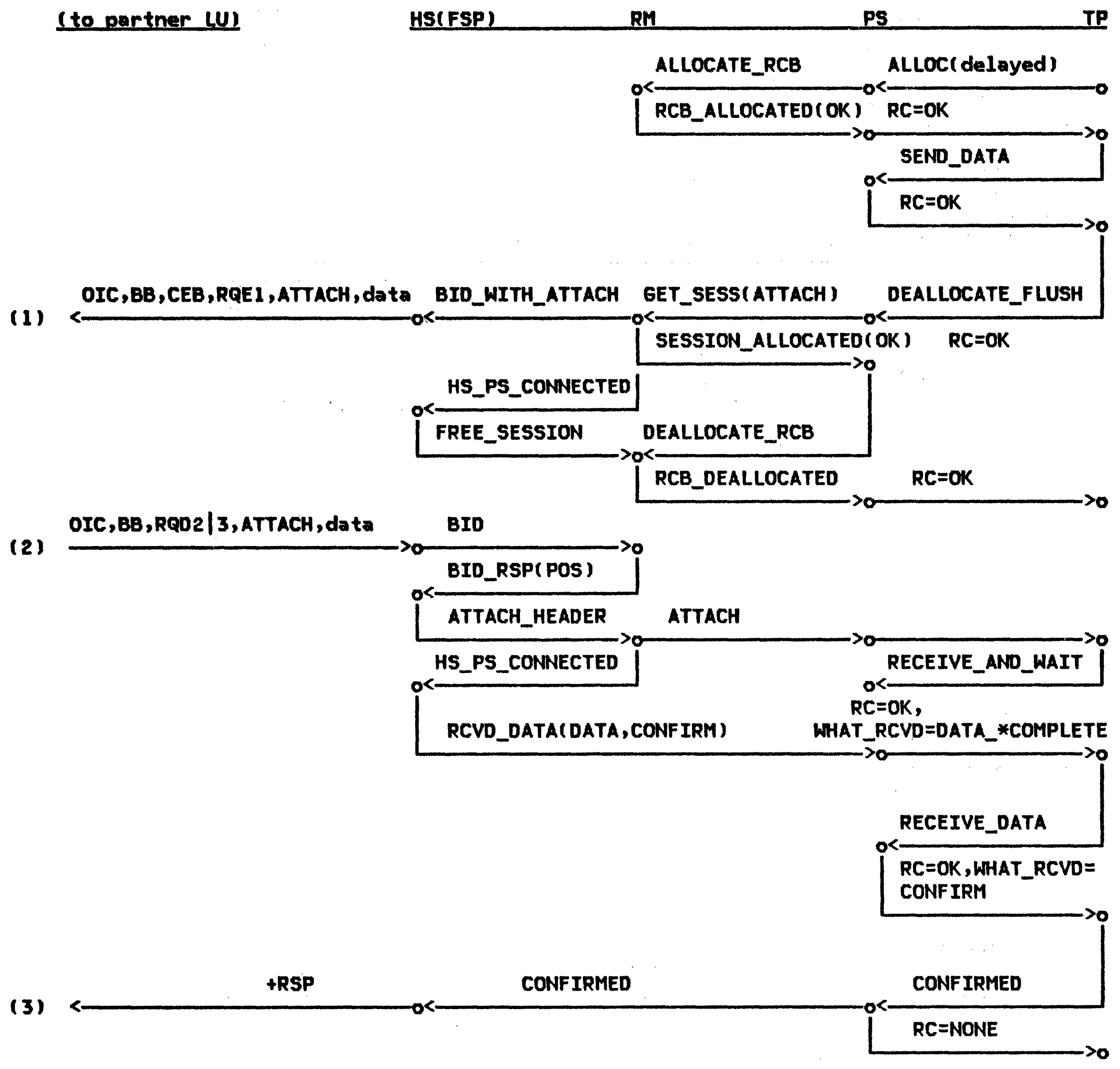


Figure 2-58. ALLOCATE (delayed) Race, Bracket Accepted --First Speaker LU

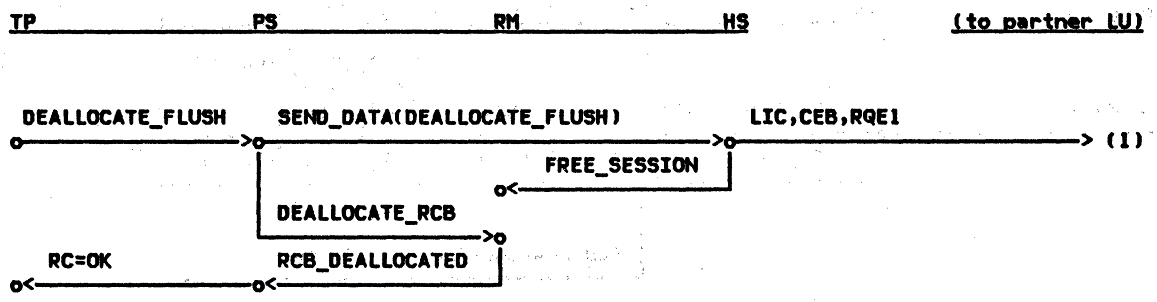


Figure 2-59. DEALLOCATE FLUSH (RQE1) --Local LU

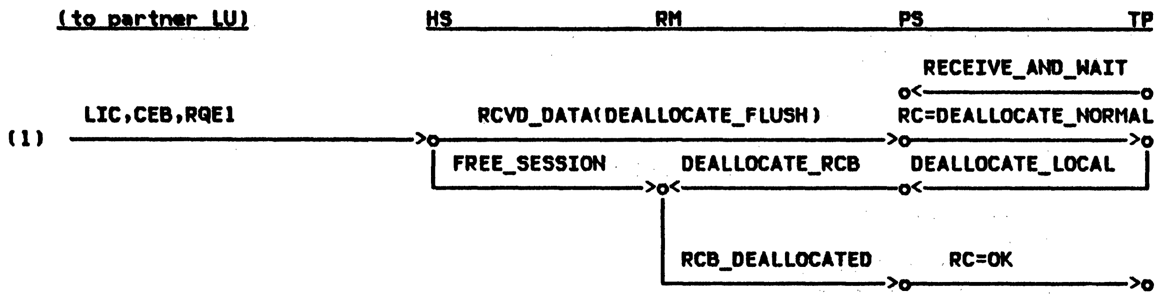
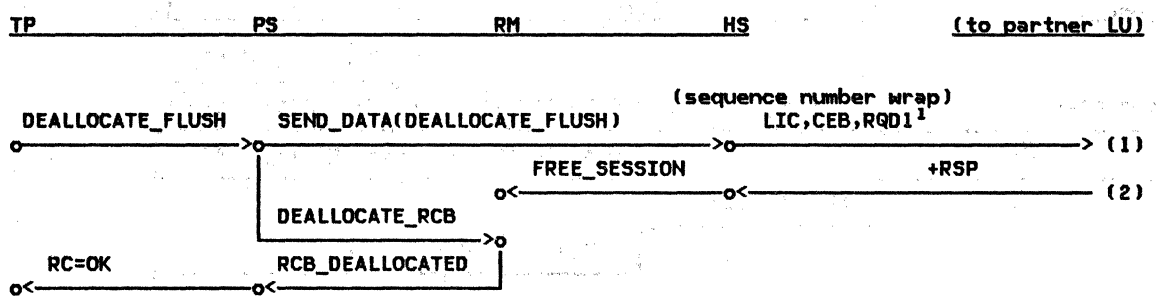


Figure 2-60. DEALLOCATE FLUSH (RQE1) --Remote LU



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

Figure 2-61. DEALLOCATE FLUSH (RQD1) --Local LU

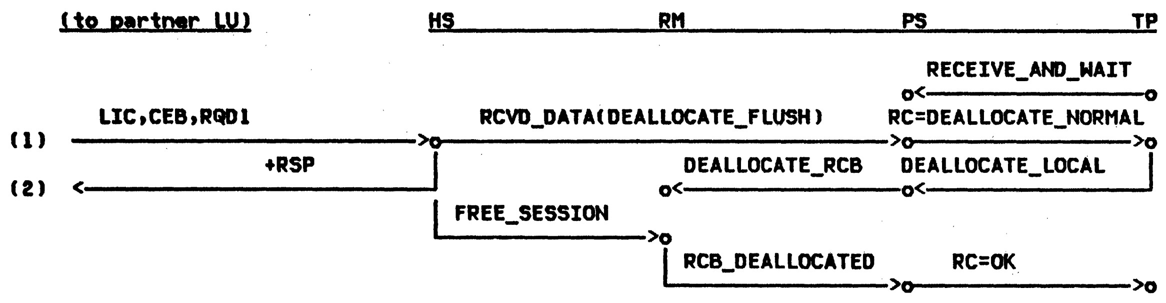


Figure 2-62. DEALLOCATE FLUSH (RQD1) --Remote LU

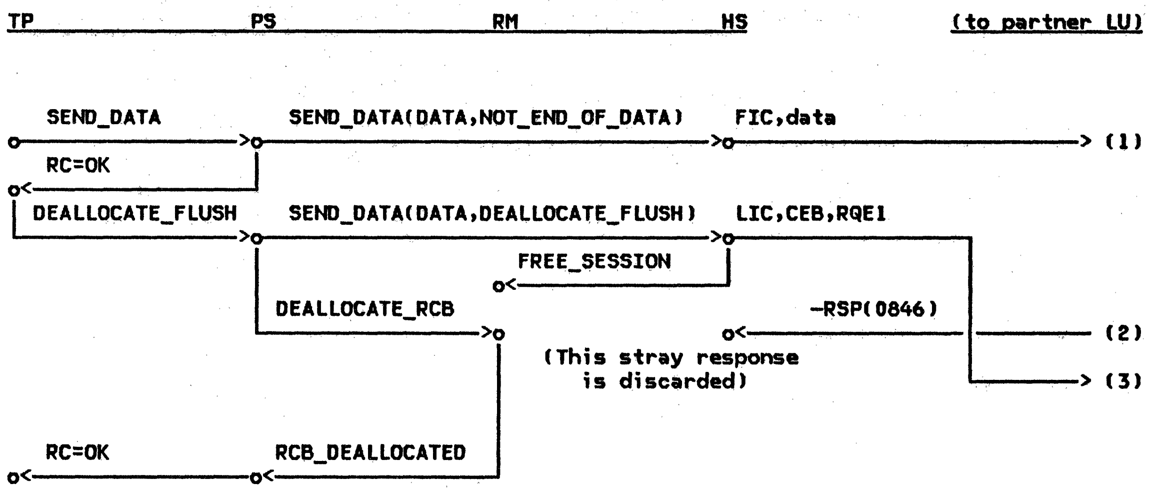


Figure 2-63. DEALLOCATE FLUSH (RQE1), SEND_ERROR, -RSP Sent --Local LU

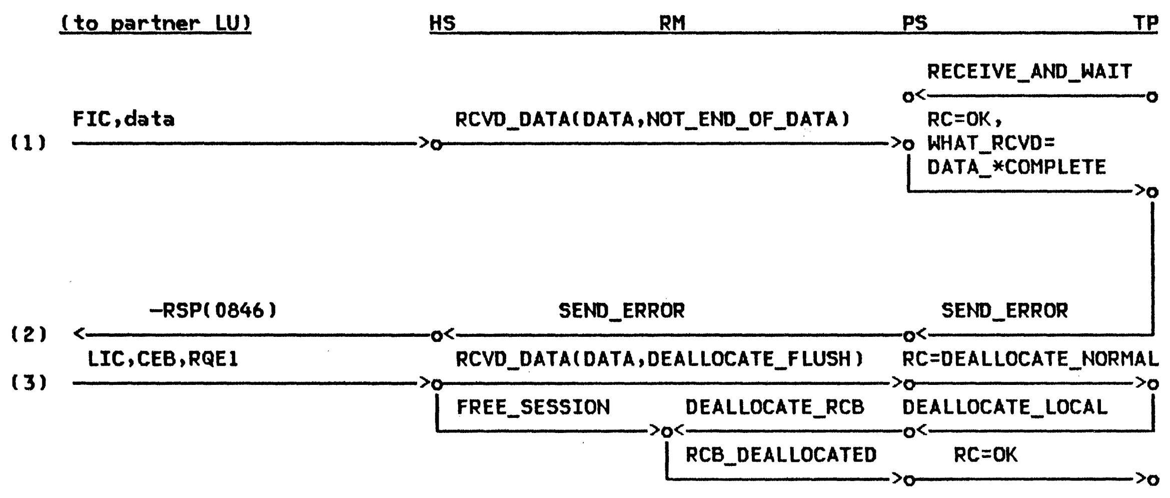


Figure 2-64. DEALLOCATE FLUSH (RQE1), SEND_ERROR, -RSP Sent --Remote LU

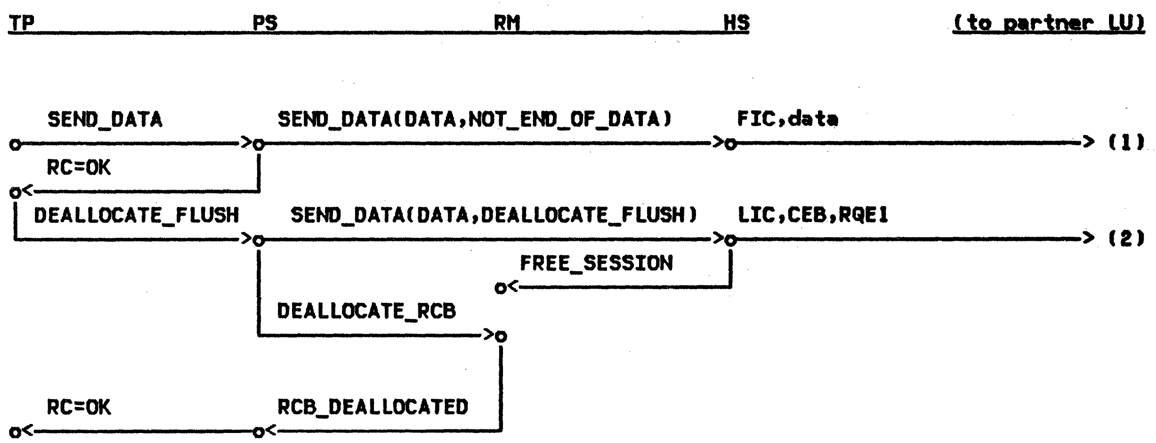


Figure 2-65. DEALLOCATE FLUSH (RQE1), SEND_ERROR, -RSP not Sent --Local LU

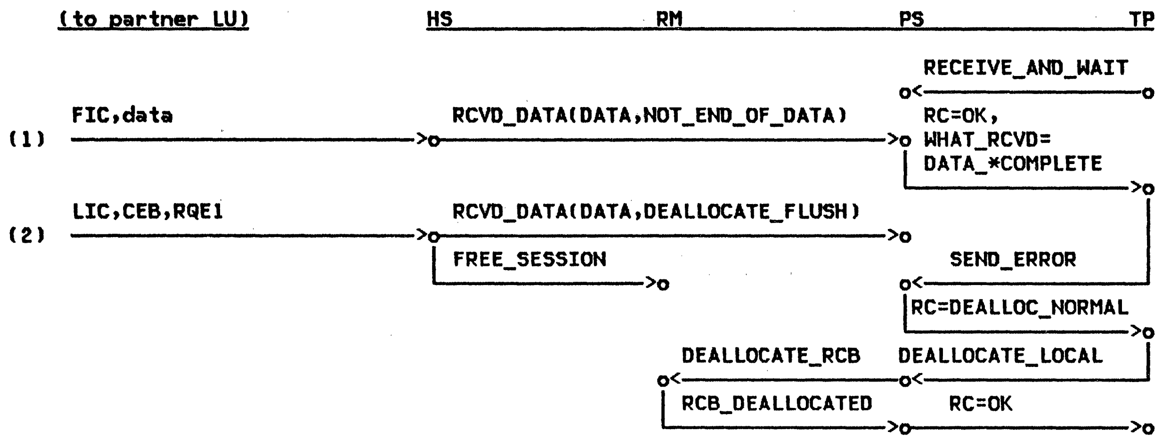


Figure 2-66. DEALLOCATE FLUSH (RQE1), SEND_ERROR, -RSP not Sent --Remote LU

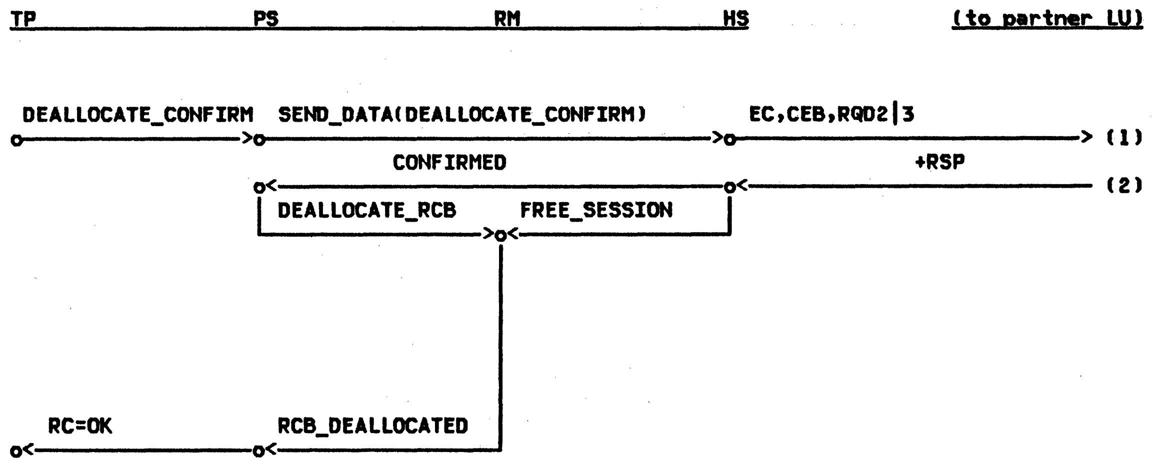


Figure 2-67. DEALLOCATE CONFIRM (RQD2|3) --Local LU

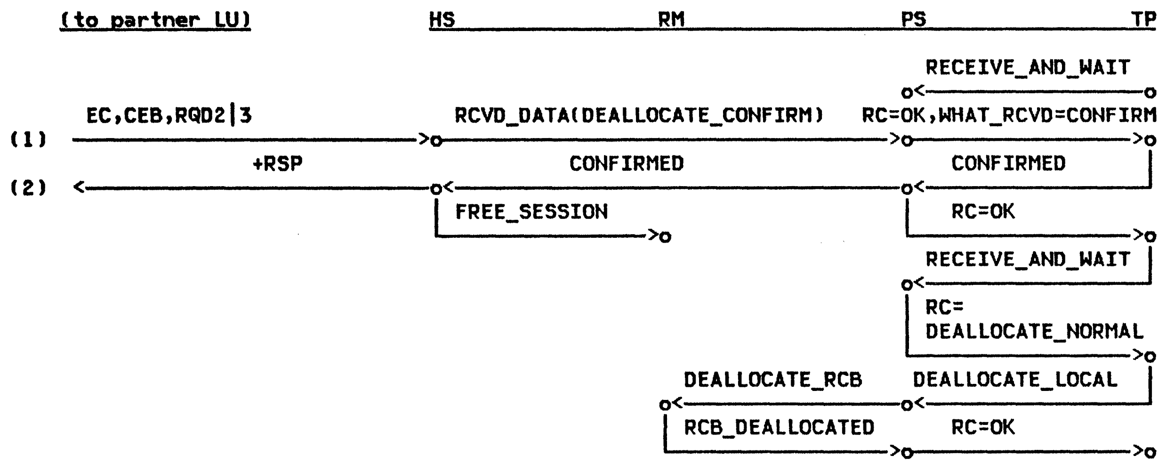


**Figure 2-68. DEALLOCATE CONFIRM (RQD2|3) --Remote LU**

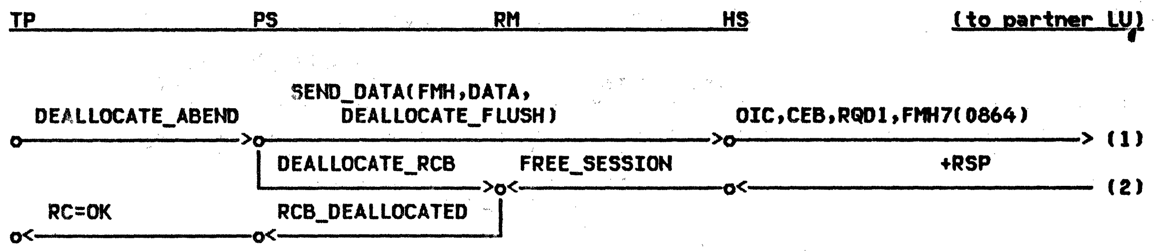


Figure 2-69. DEALLOCATE ABEND Issued in SEND, Between-Chain State --Local LU

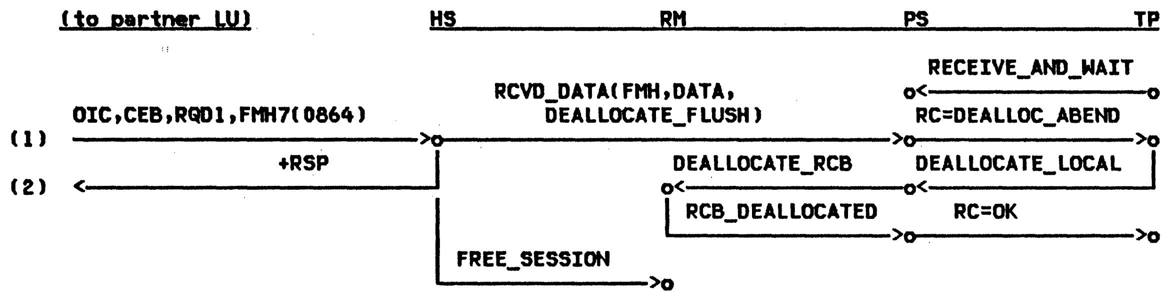


Figure 2-70. DEALLOCATE ABEND Issued in SEND, Between-Chain State --Remote LU

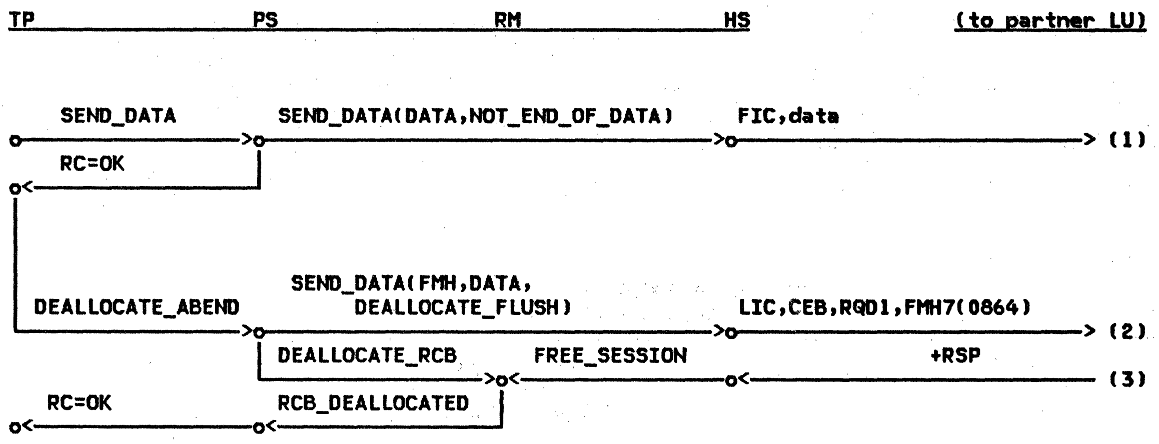


Figure 2-71. DEALLOCATE ABEND Issued in SEND, In-Chain State --Local LU

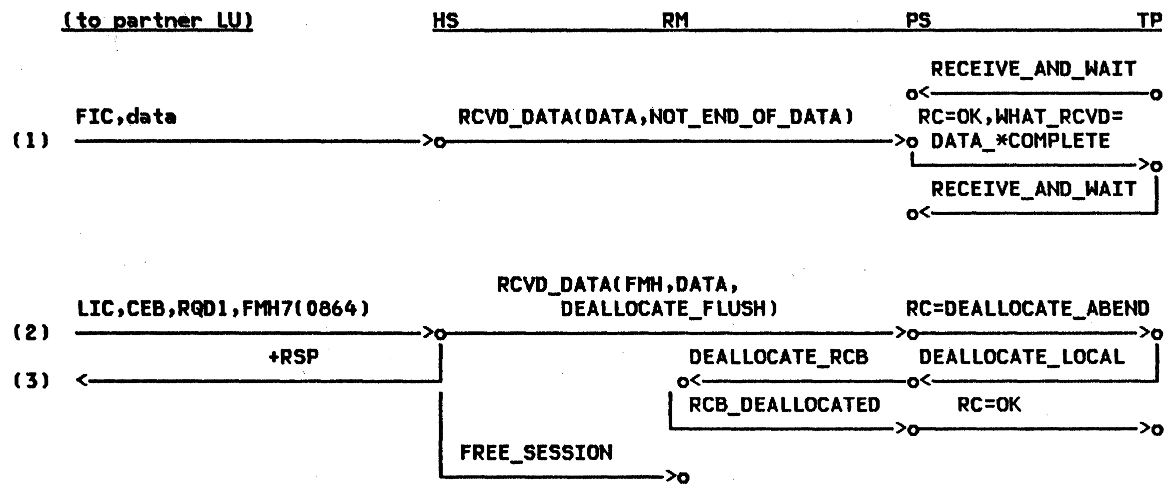


Figure 2-72. DEALLOCATE ABEND Issued in SEND, In-Chain State --Remote LU

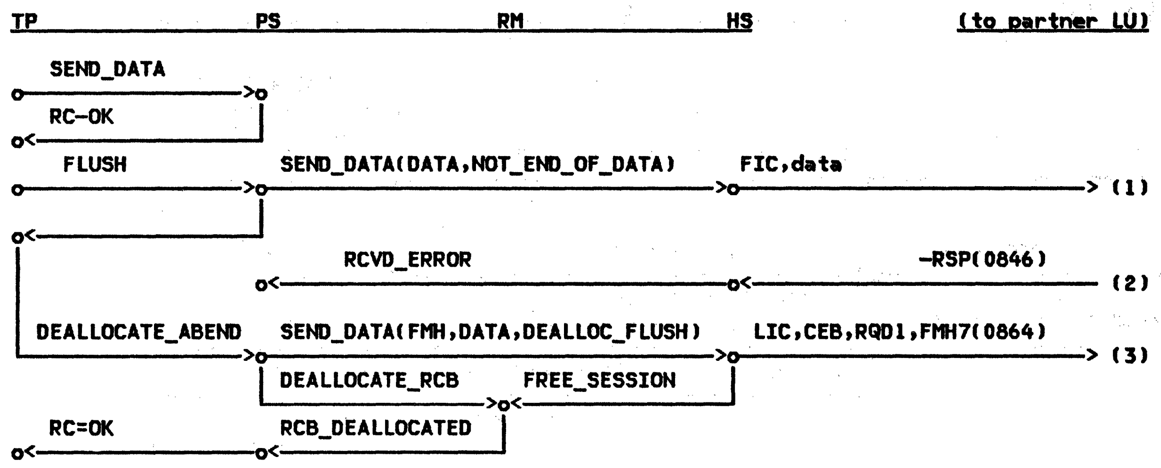
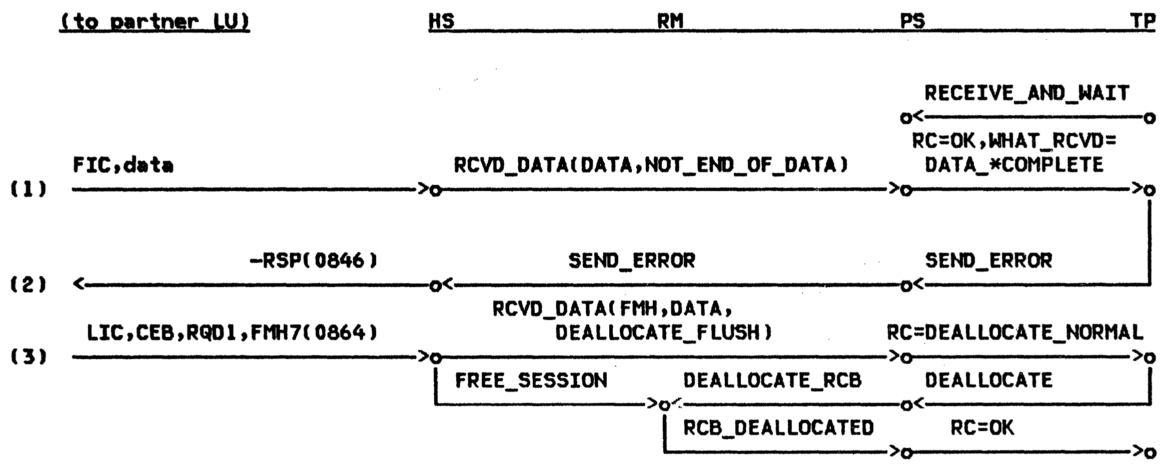


Figure 2-73. DEALLOCATE ABEND Issued in SEND, -RSP Received State --Local LU



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

Figure 2-74. DEALLOCATE ABEND Issued in SEND, -RSP Received State --Remote LU

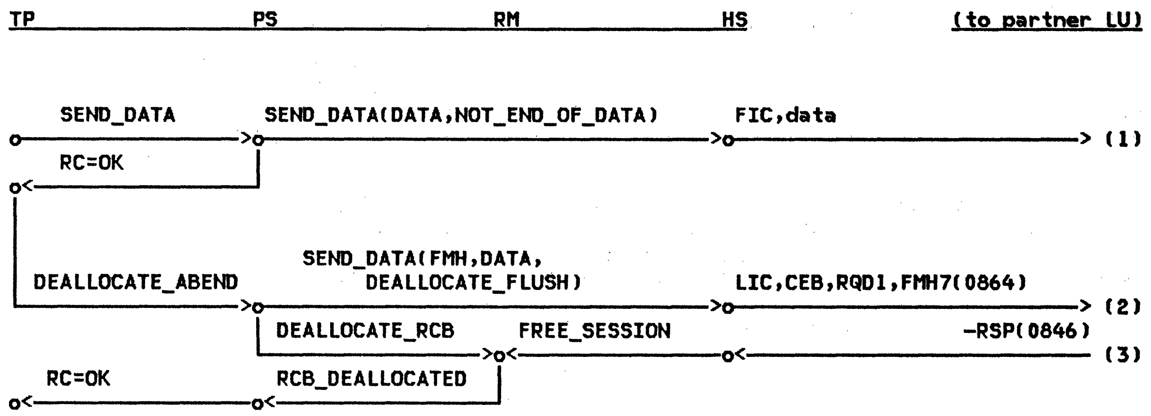
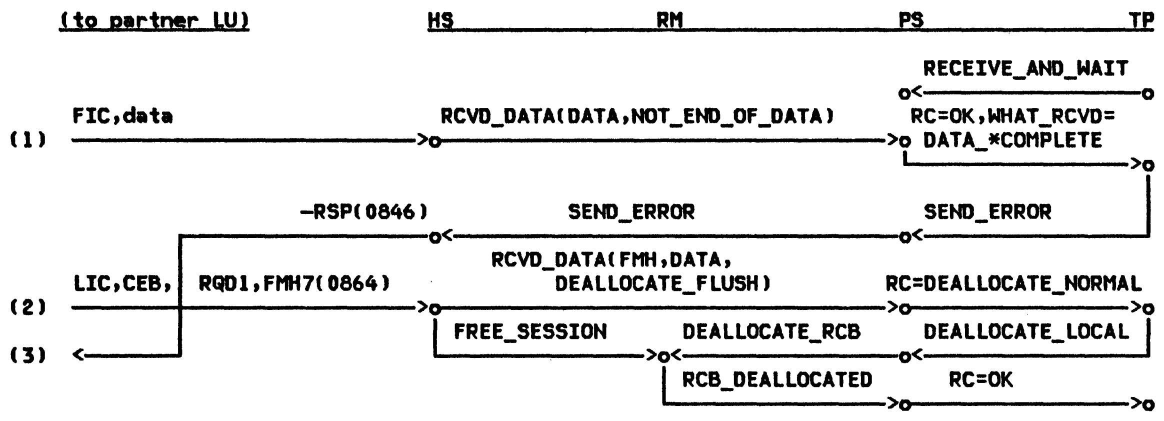


Figure 2-75. DEALLOCATE ABEND Issued in SEND State --Local LU



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

Figure 2-76. DEALLOCATE ABEND Issued in SEND State --Remote LU

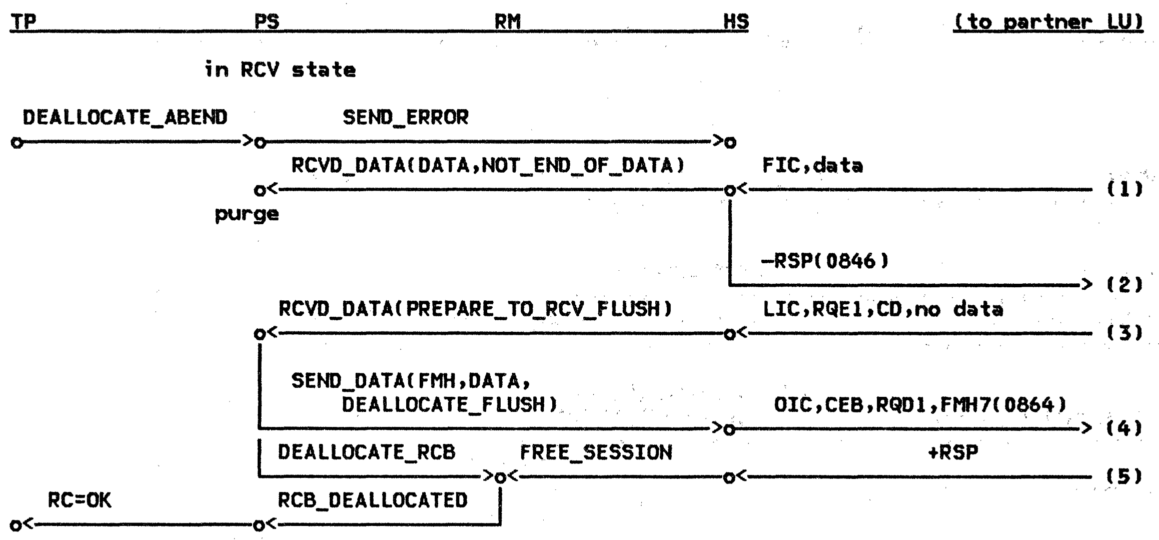


Figure 2-77. DEALLOCATE ABEND Issued in RCV, Between-Chain State --Local LU

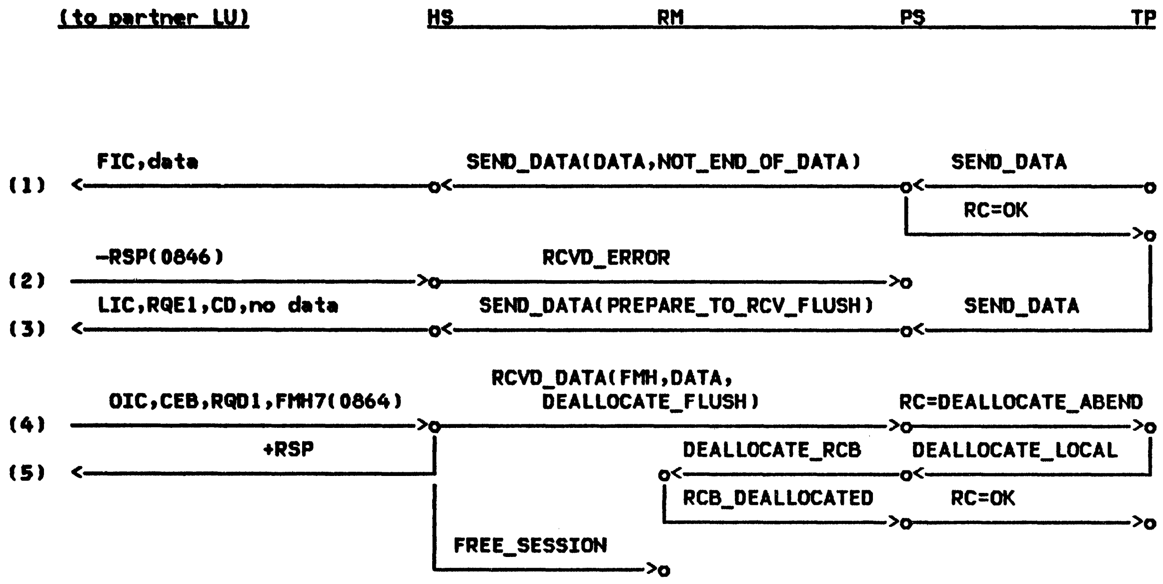


Figure 2-78. DEALLOCATE ABEND Issued in RCV, Between-Chain State --Remote LU

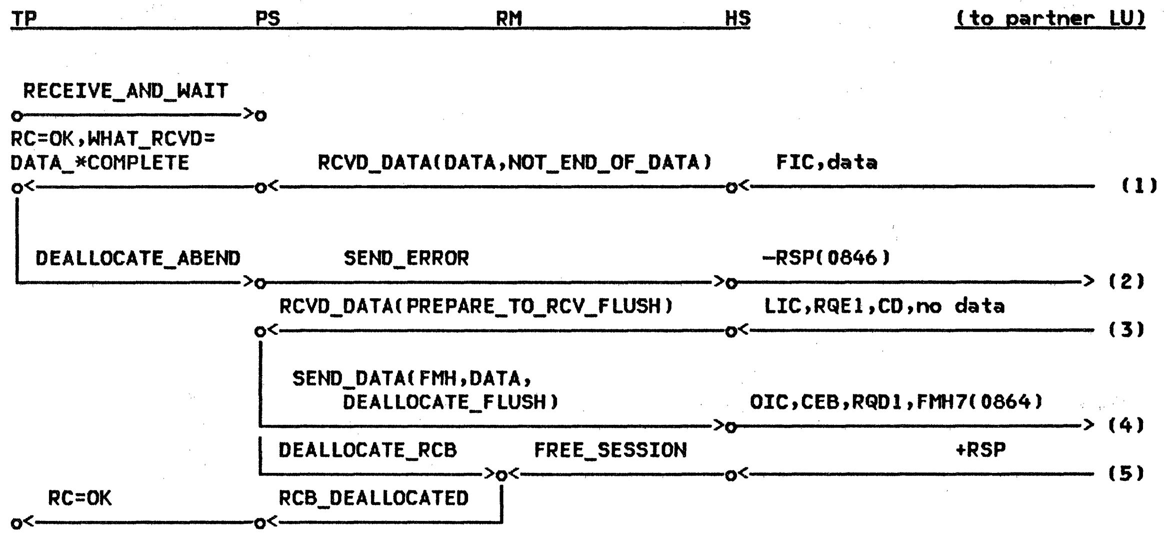


Figure 2-79. DEALLOCATE ABEND Issued in RCV, In-Chain State --Local LU

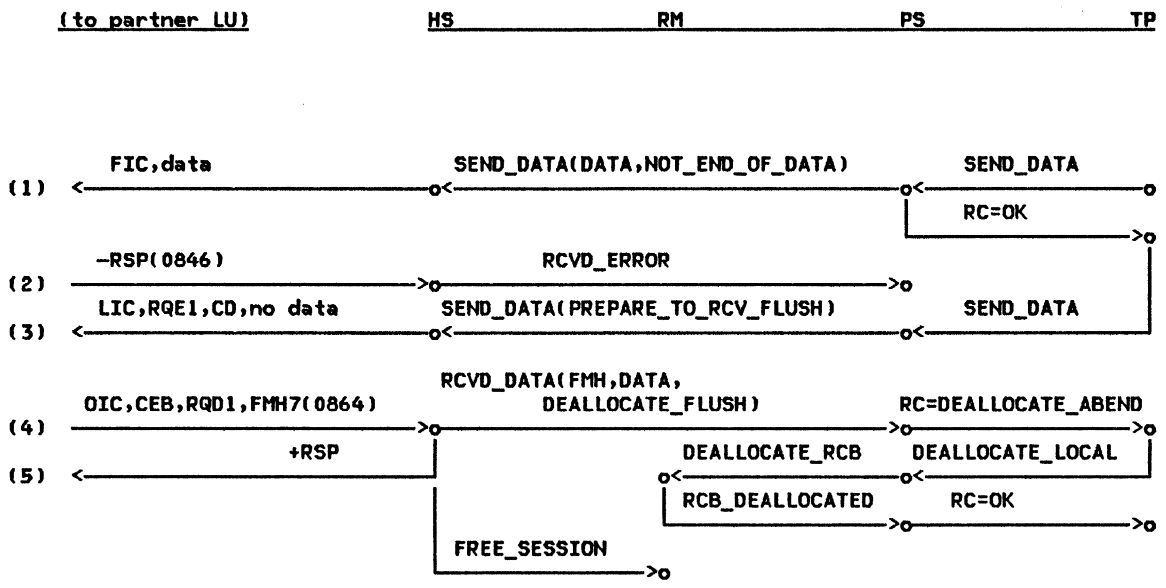


Figure 2-80. DEALLOCATE ABEND Issued in RCV, In-Chain State --Remote LU

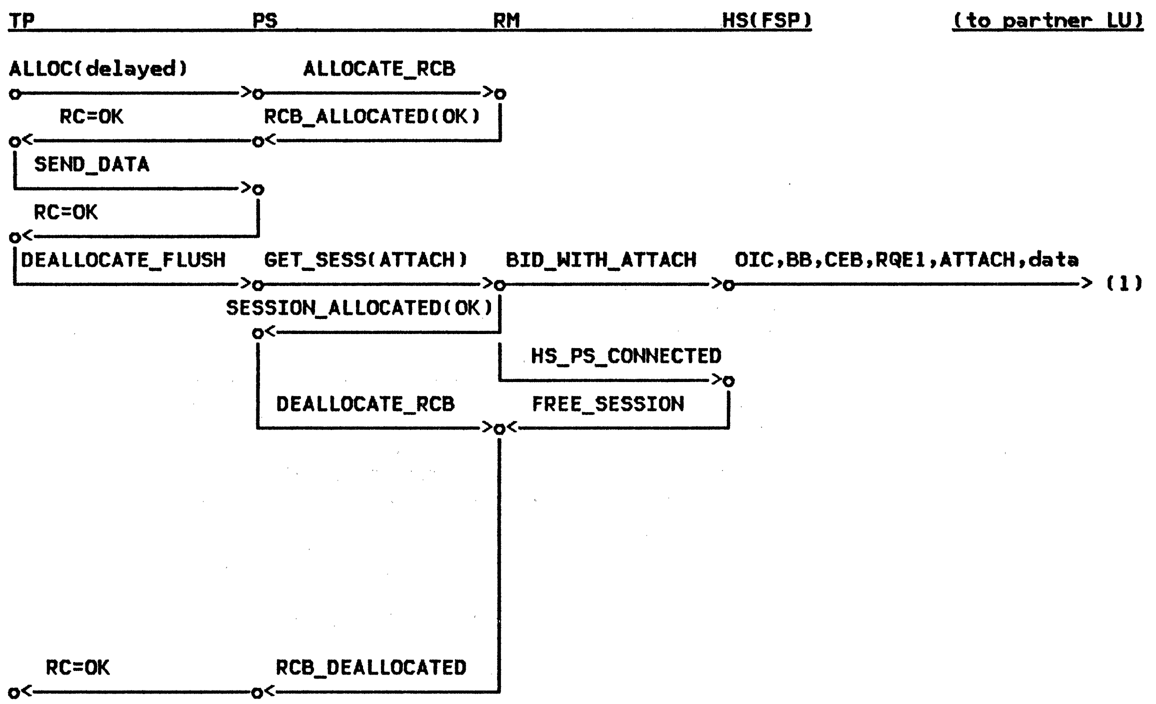


Figure 2-81. ALLOCATE (delayed), DEALLOCATE FLUSH (by First Speaker) --Local LU

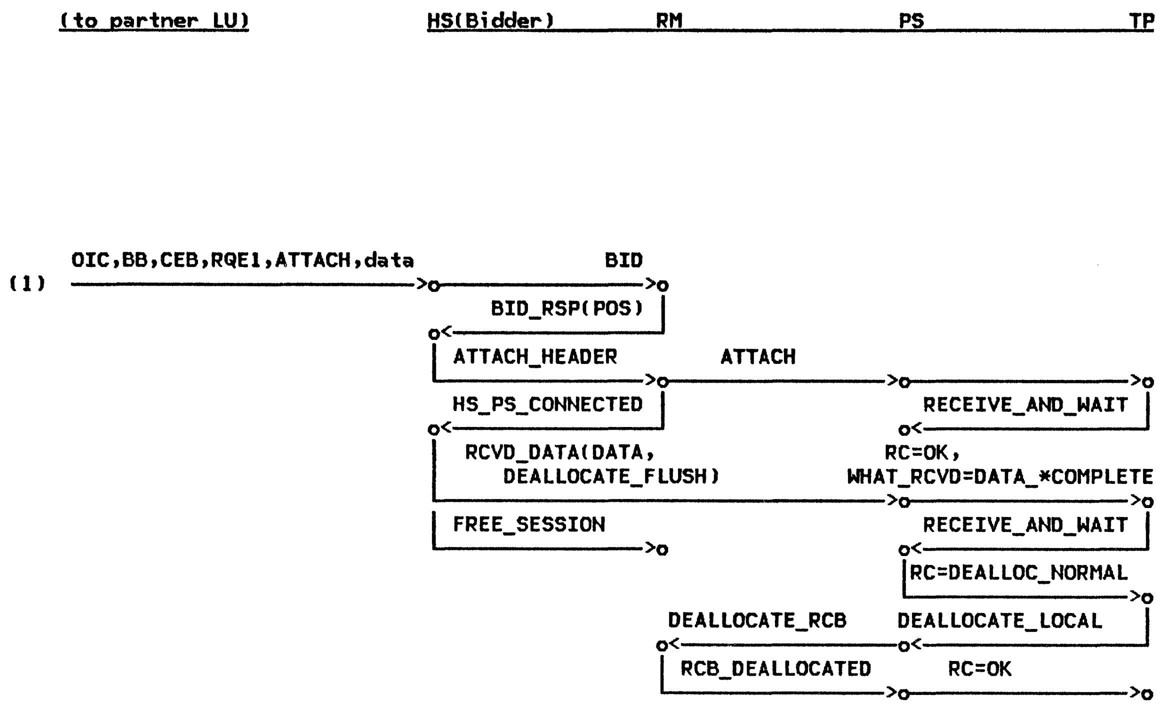


Figure 2-82. ALLOCATE (delayed), DEALLOCATE FLUSH (by First Speaker) --Remote LU

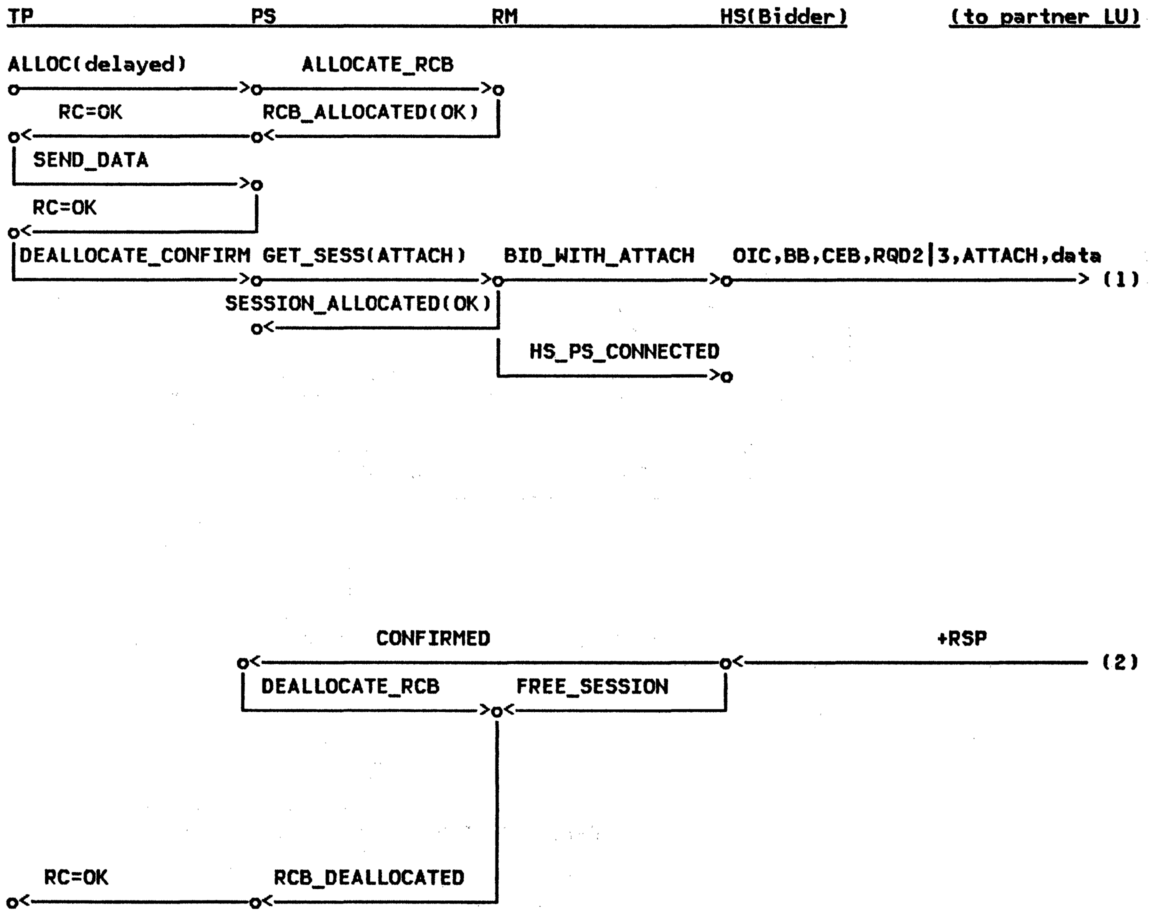


Figure 2-83. ALLOCATE (delayed), DEALLOCATE CONFIRM (BY First Speaker) --Local LU

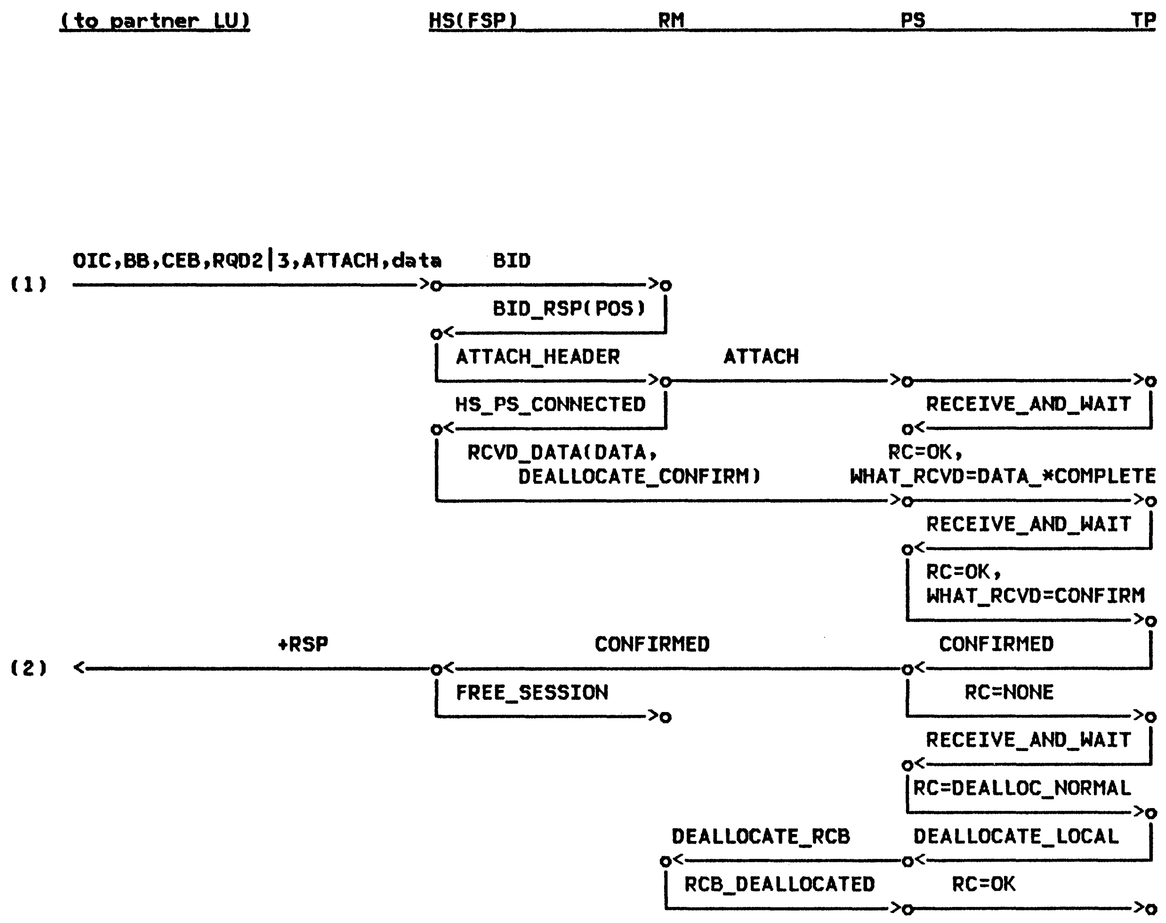


Figure 2-84. ALLOCATE (delayed), DEALLOCATE CONFIRM (BY First Speaker) --Remote LU

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

ALLOC(delayed)          ALLOCATE_RCB
o------------------->o------------------->o
    RC=OK            RCB_ALLOCATED(OK) |
o<-------------------o<-----------------+
| SEND_DATA
+------------------->o
  RC=OK              |
o<-------------------+
|DEALLOCATE_FLUSH   GET_SESS(ATTACH)   BID_WITH_ATTACH   OIC,BB,CEB,RQD1,ATTACH,data
+------------------->o------------------->o------------------>o----------------------------> (1)




                  SESSION_ALLOCATED(OK)   BID_RSP(POS)                   +RSP
                     o<-------------------o<------------------o<---------------------------- (2)
                     |                    | HS_PS_CONNECTED
                     |                    +------------------>o
                     |  DEALLOCATE_RCB       FREE_SESSION     |
                     +------------------->o<------------------+
   RC=OK                RCB_DEALLOCATED   |
o<-------------------o<-------------------+
```

Figure 2-85. ALLOCATE (delayed), DEALLOCATE FLUSH (by Bidder) to RECEIVE_AND_WAIT --Local LU

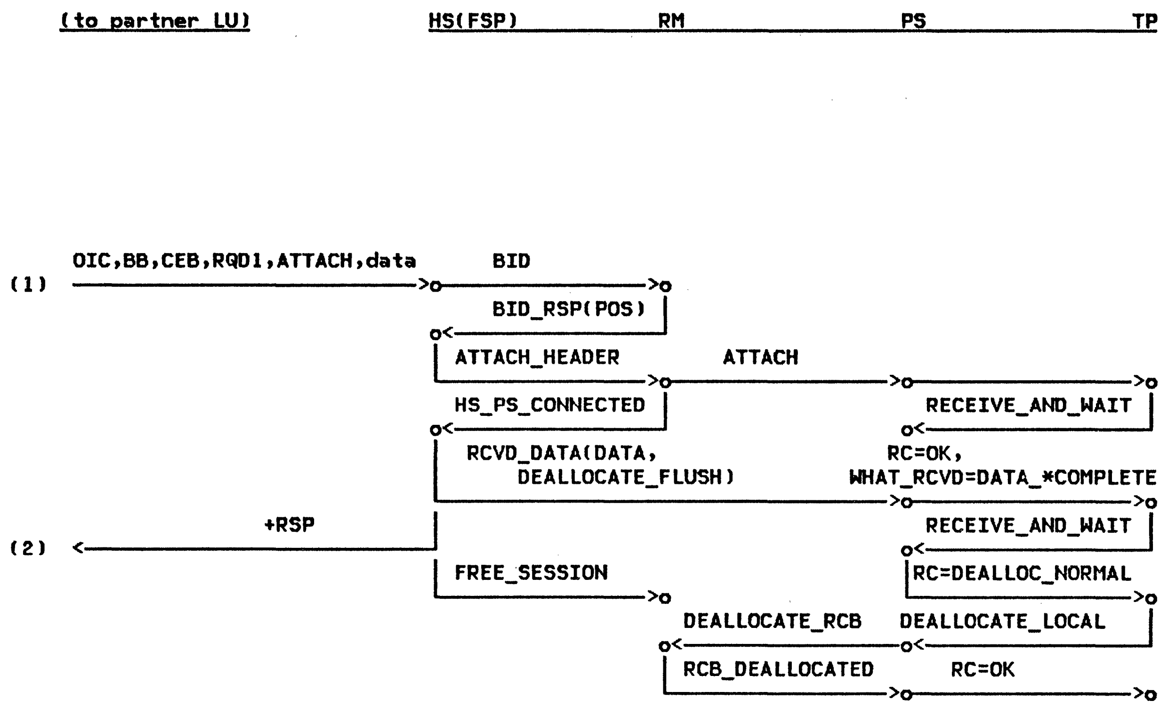


Figure 2-86. ALLOCATE (delayed), DEALLOCATE FLUSH (by Bidder) to RECEIVE_AND_WAIT --Remote LU

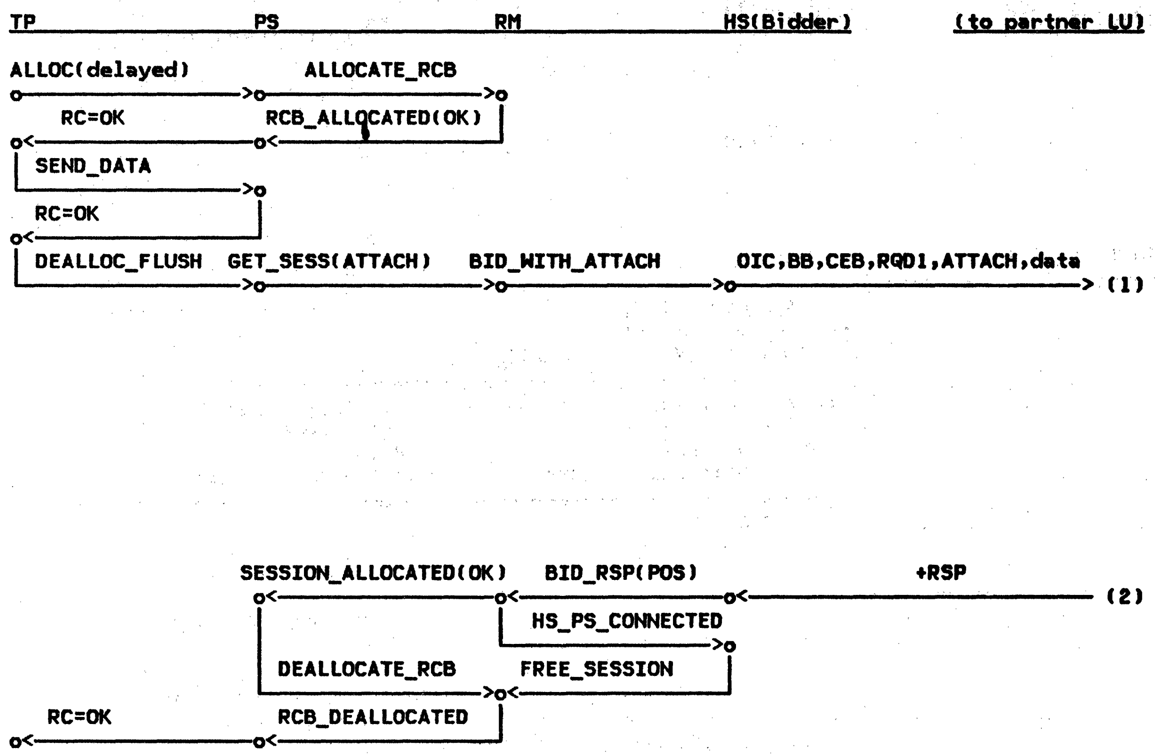


Figure 2-87. ALLOCATE (delayed), DEALLOCATE FLUSH (by Bidder) to SEND_ERROR --Local LU

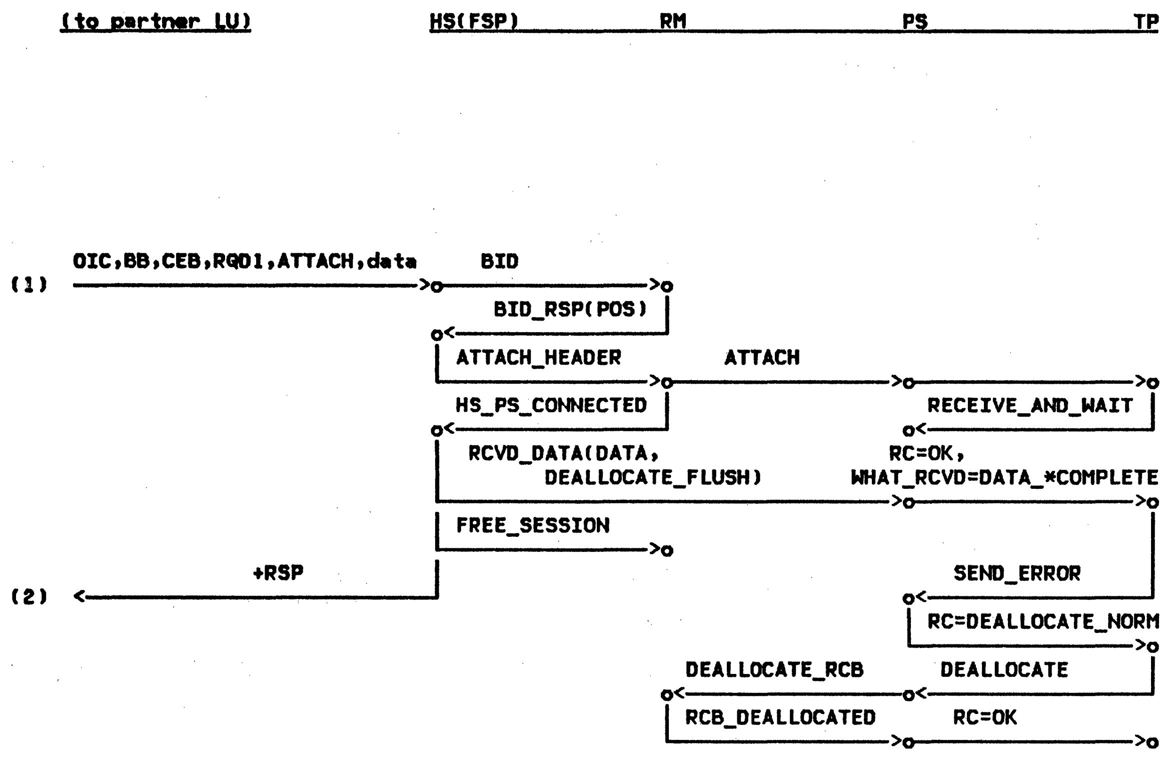


**Figure 2-88. ALLOCATE (delayed), DEALLOCATE FLUSH (by Bidder) to SEND_ERROR --Remote LU**

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

ALLOC(delayed)          ALLOCATE_RCB
o------------------->o------------------->o
    RC=OK               RCB_ALLOCATED(OK) |
o<-------------------o<-------------------+
| SEND_DATA
+------------------->o
  RC=OK              |
o<-------------------+
|DEALLOCATE_CONFIRM GET_SESS(ATTACH)   BID_WITH_ATTACH   OIC,BB,CEB,RQD2|3,ATTACH,data
+------------------->o------------------->o------------------->o----------------------------> (1)




                  SESSION_ALLOCATED(OK)  BID_RSP(POS)                 +RSP
                     o<-------------------o<-------------------o<---------------------------- (2)
                                          | HS_PS_CONNECTED
                                          +------------------->o
                             CONFIRMED                         |
                     o<----------------------------------------+
                     | DEALLOCATE_RCB        FREE_SESSION      |
                     +------------------->o<-------------------+
                                          |
    RC=OK               RCB_DEALLOCATED   |
o<-------------------o<-------------------+
```

Figure 2-89. ALLOCATE (delayed), DEALLOCATE CONFIRM (by Bidder) --Local LU

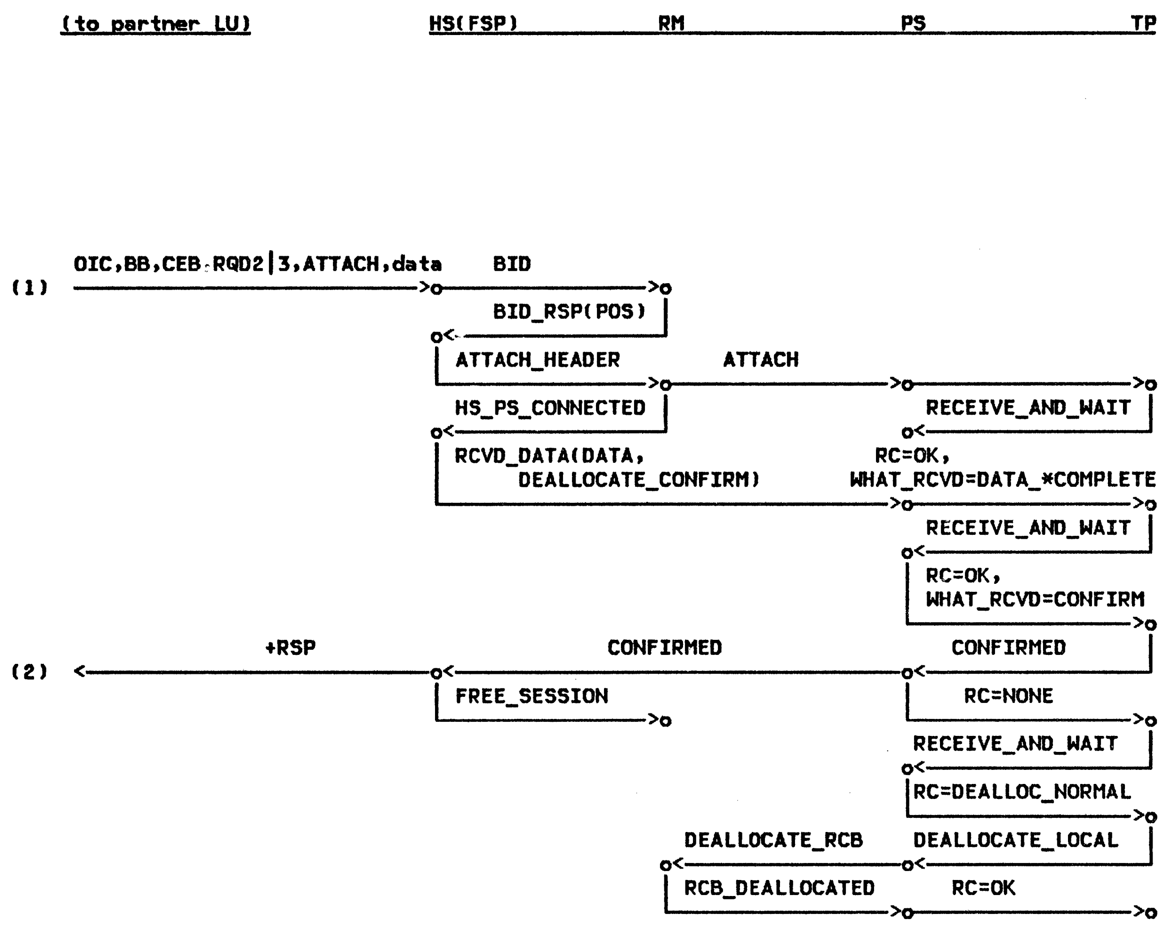


Figure 2-90. ALLOCATE (delayed), DEALLOCATE CONFIRM (by Bidder) --Remote LU

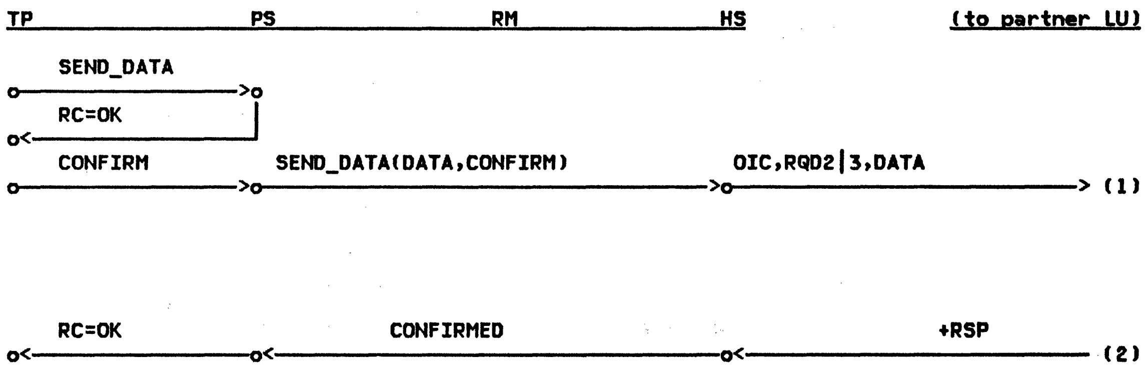


Figure 2-91. CONFIRM (RQD2|3) --Local LU

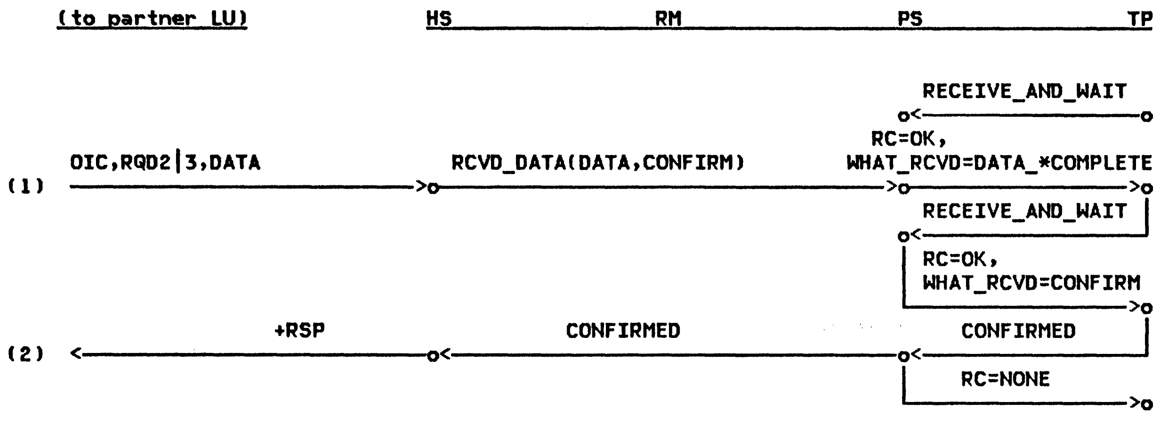


Figure 2-92. CONFIRM (RQD2|3) --Remote LU

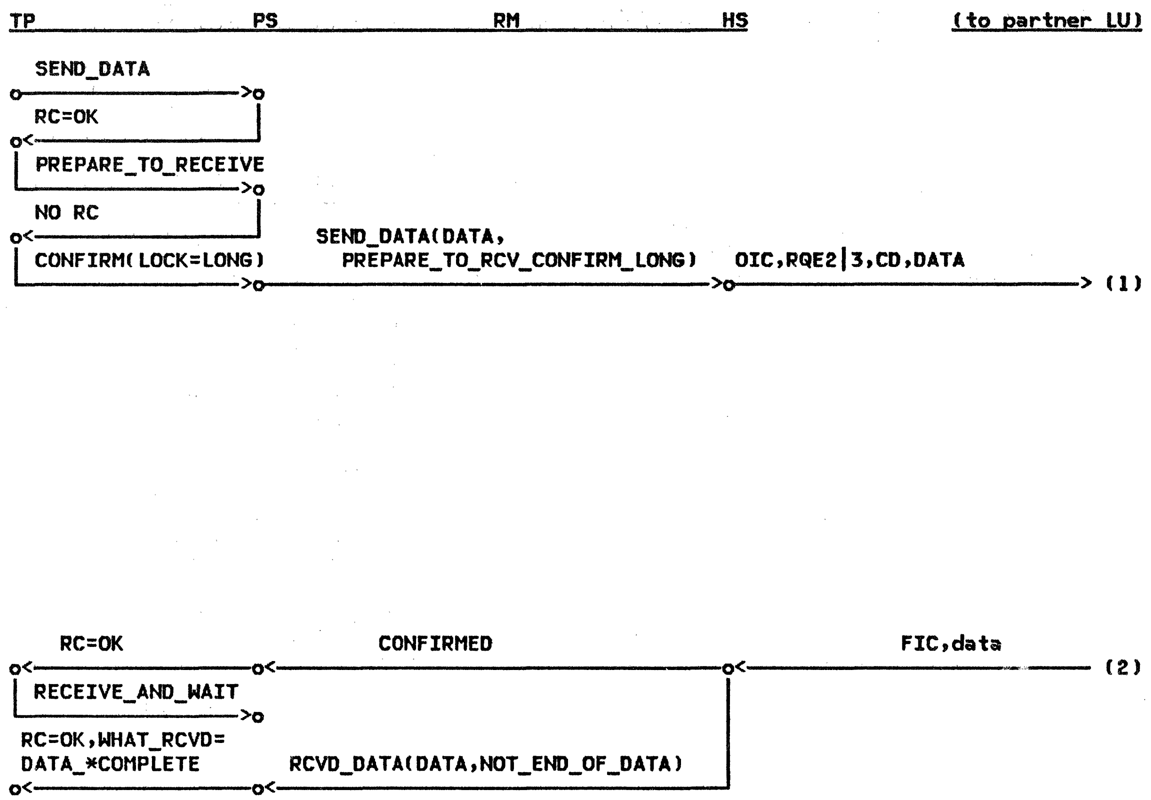


Figure 2-93. CONFIRM (RQE2|3) --Local LU

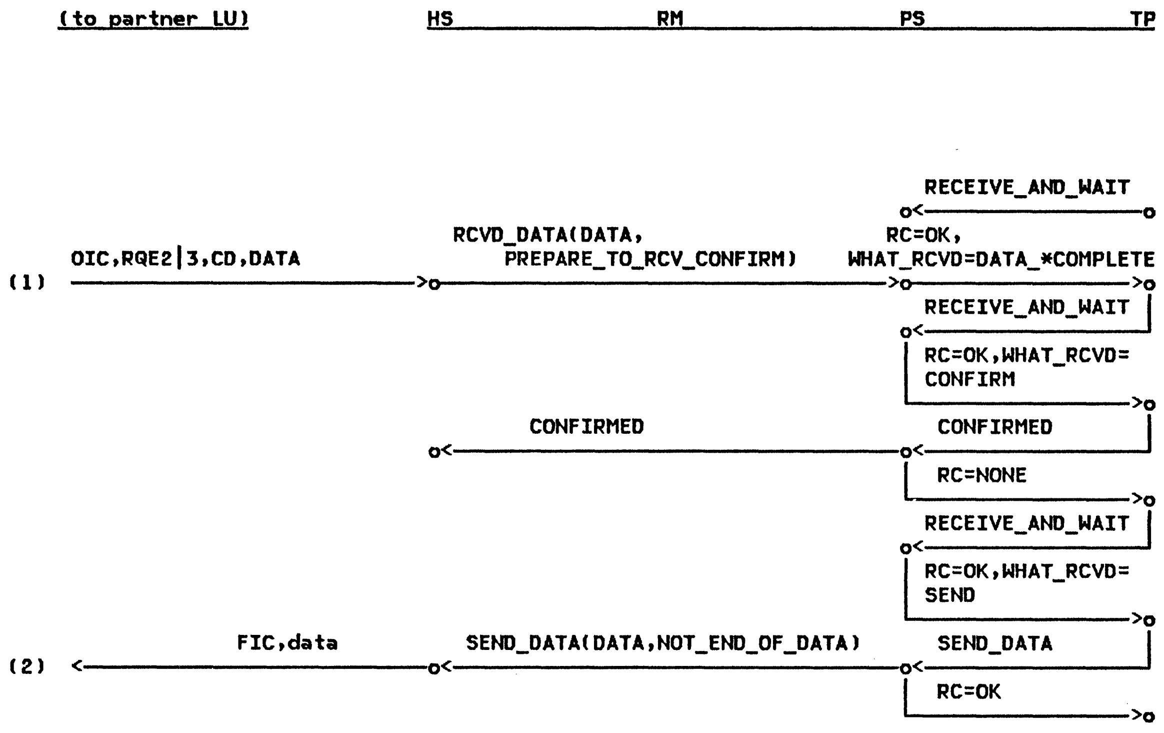


Figure 2-94. CONFIRM (RQE2|3) --Remote LU

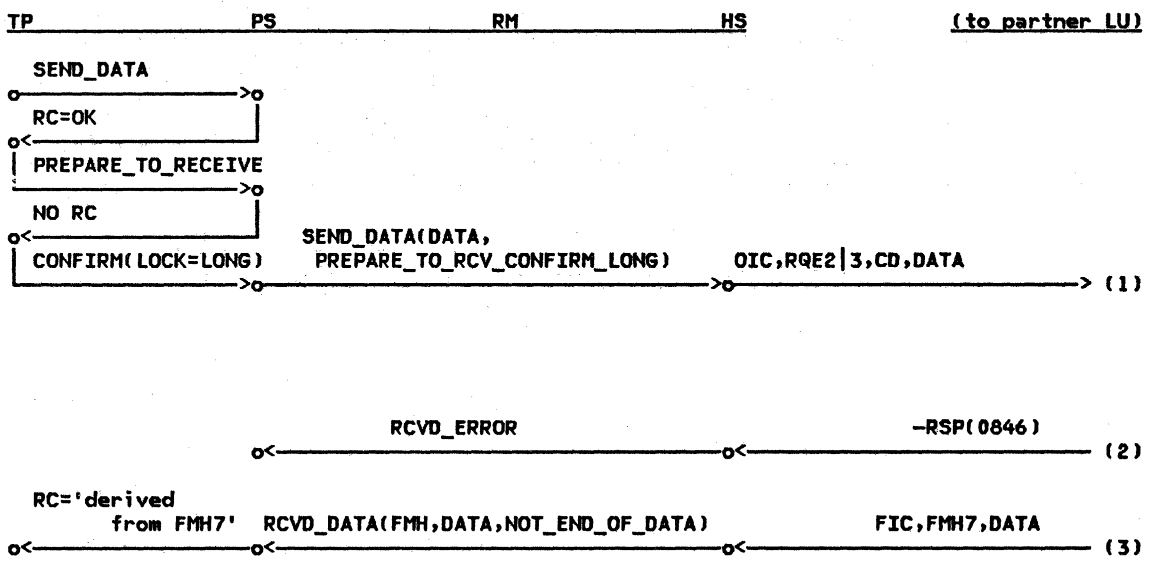


Figure 2-95. CONFIRM (RQE2|3), SEND_ERROR --Local LU

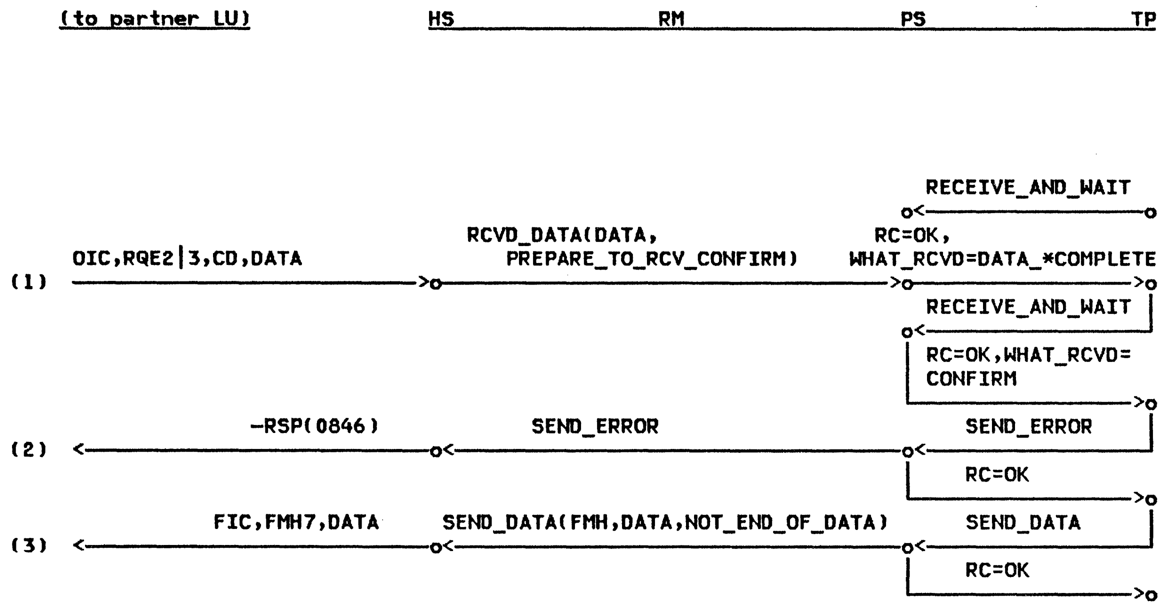


Figure 2-96. CONFIRM (RQE2|3), SEND_ERROR --Remote LU

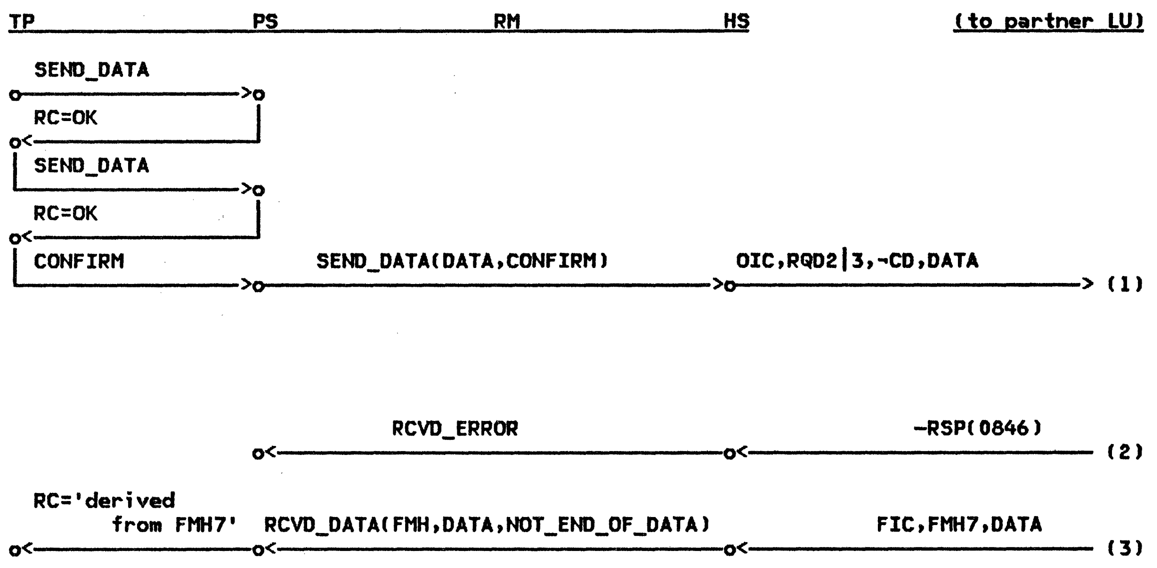


Figure 2-97. CONFIRM (RQD2|3), SEND_ERROR --Local LU

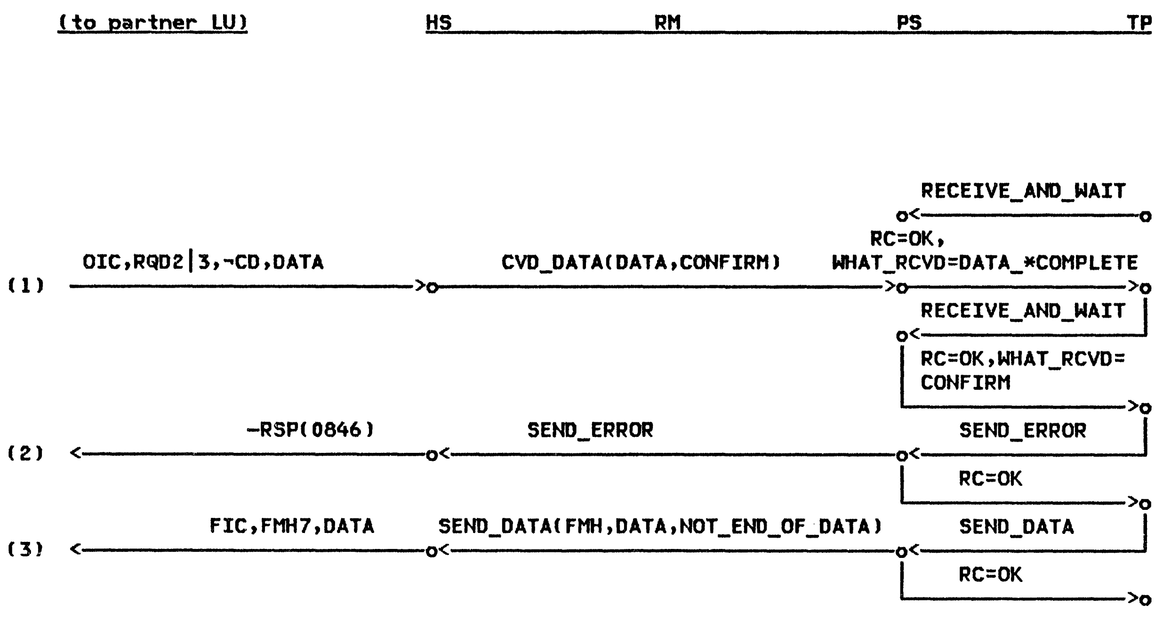


Figure 2-98. CONFIRM (RQD2|3), SEND_ERROR --Remote LU

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

    SEND_DATA
o───────────────────>o
    RC=OK            │
o<───────────────────┘
│   SEND_DATA           SEND_DATA(DATA,NOT_END_OF_DATA)       FIC,data
└───────────────────>o────────────────────────────────────>o──────────────────────────> (1)
    RC=OK            │
o<───────────────────┘   SEND_DATA(DATA,
│ RECEIVE_AND_WAIT          PREPARE_TO_RCV_FLUSH)             LIC,CD,RQE1
└───────────────────>o────────────────────────────────────>o──────────────────────────> (2)
```

Figure 2-99. RECEIVE_AND_WAIT Causing RQE,CD --Local LU

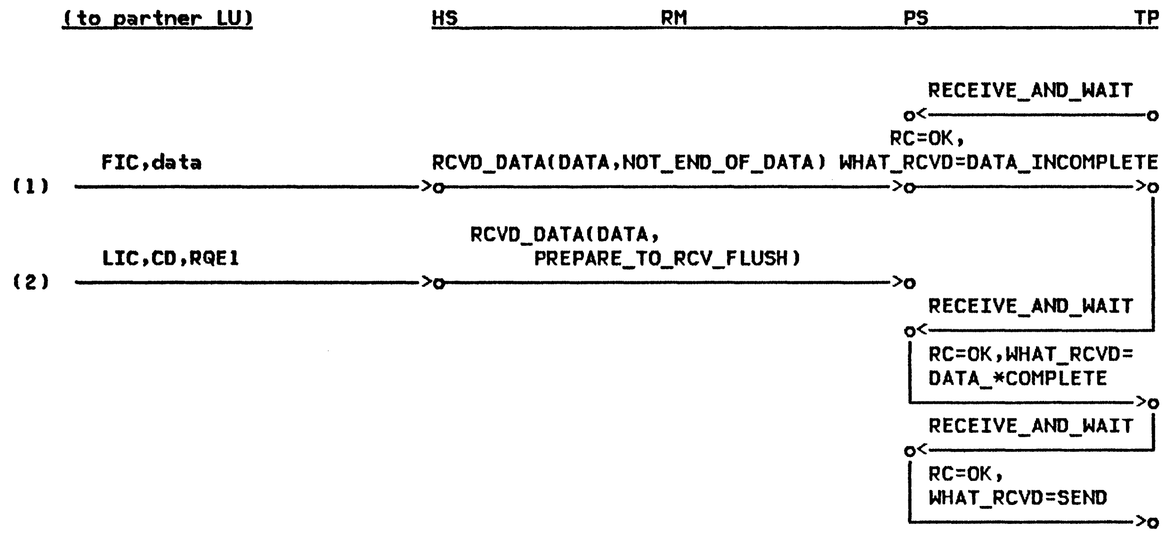


Figure 2-100. RECEIVE_AND_WAIT Causing RQE,CD --Remote LU

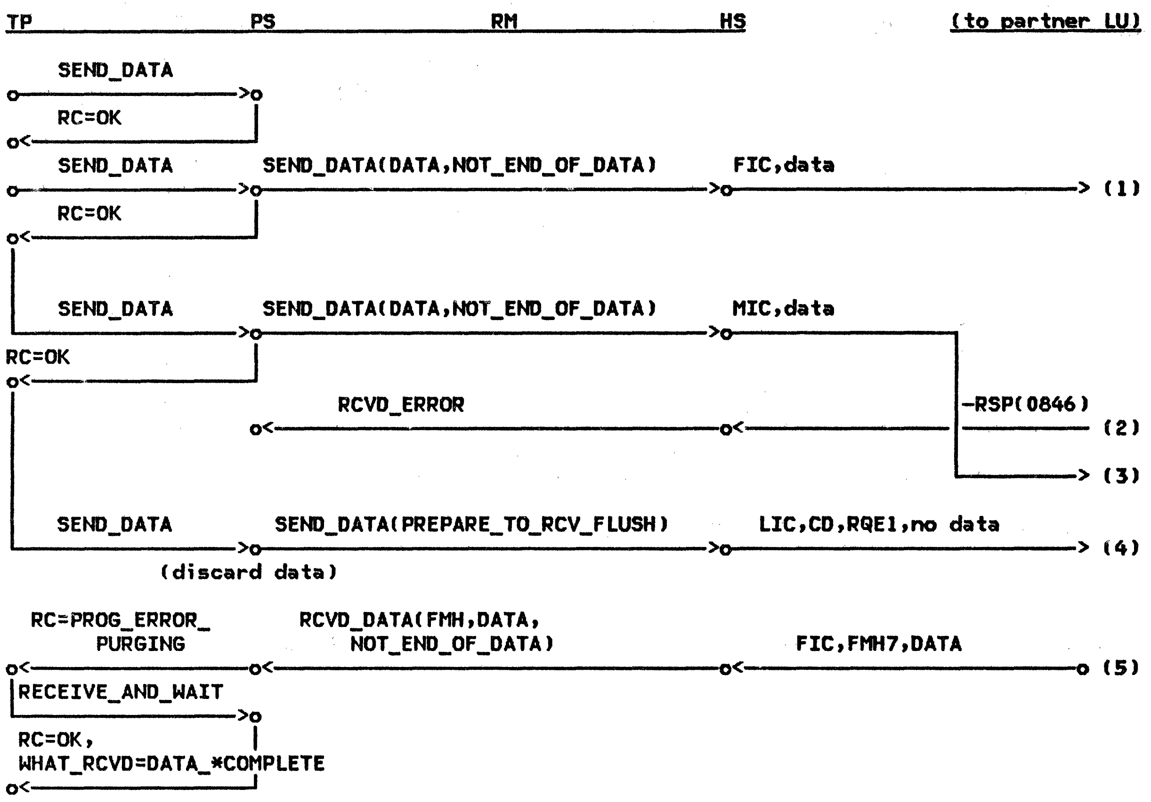


Figure 2-101. SEND_ERROR before SEND_DATA --Remote LU

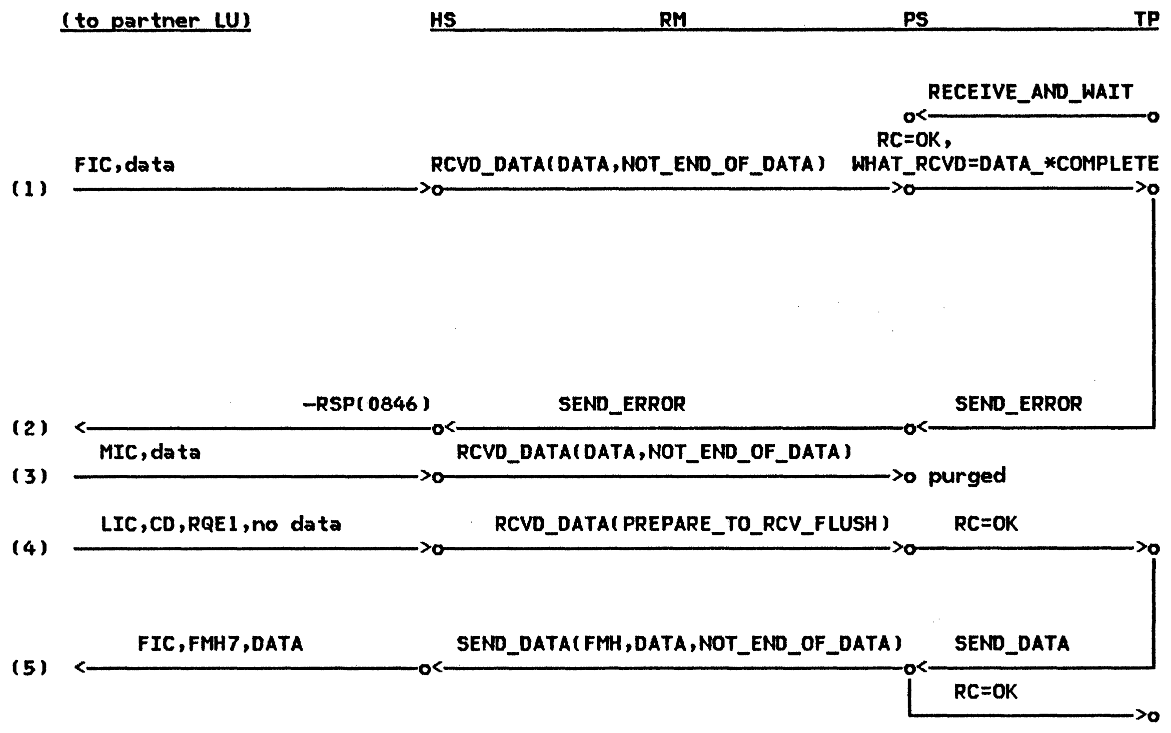


Figure 2-102. SEND_ERROR before SEND_DATA --Local LU

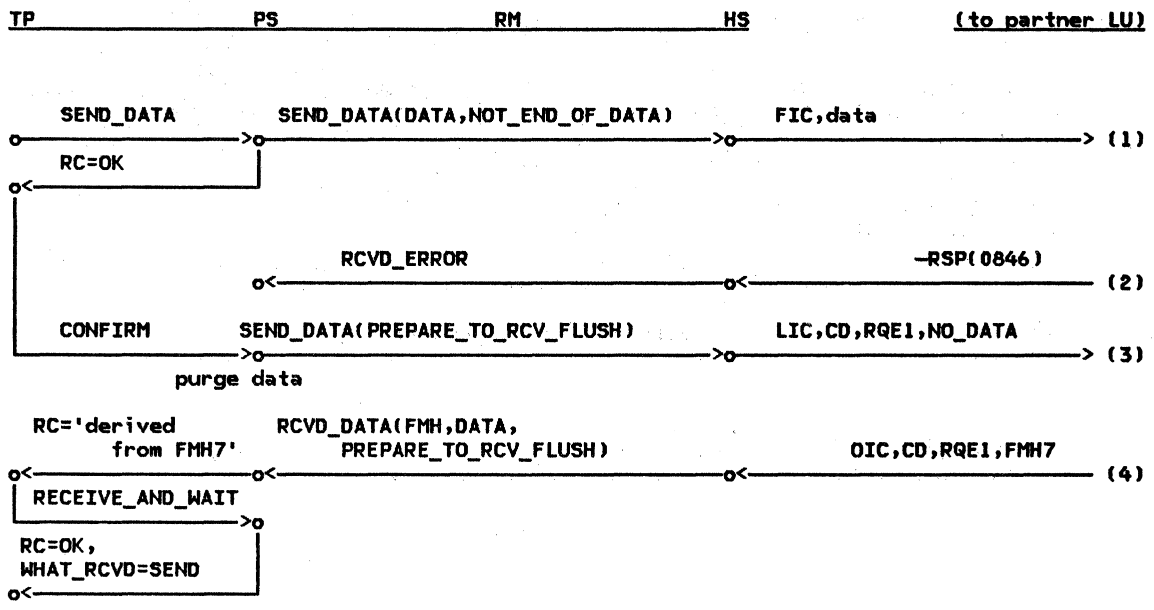


Figure 2-103. SEND_ERROR before CONFIRM --Remote LU

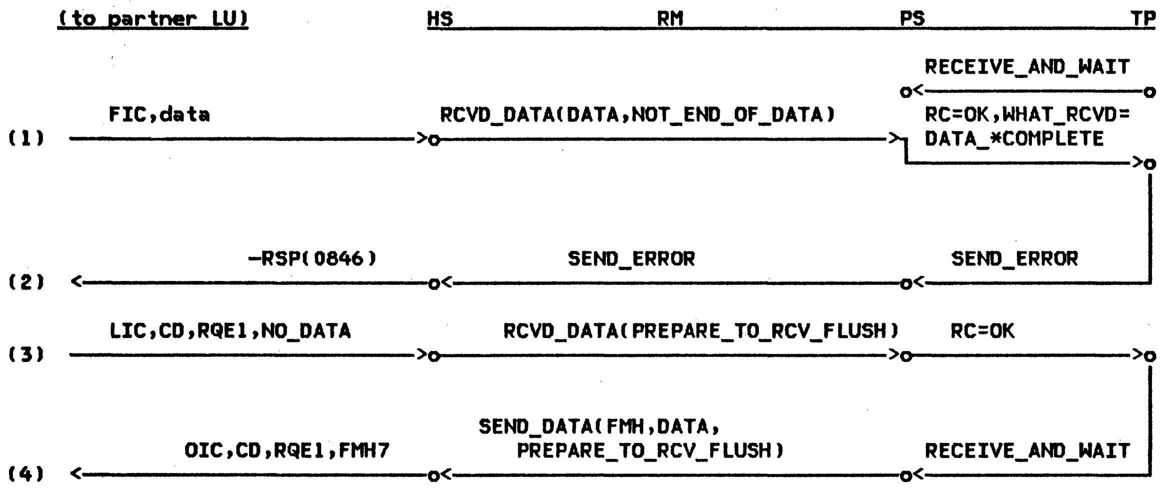


Figure 2-104. SEND_ERROR before CONFIRM --Local LU

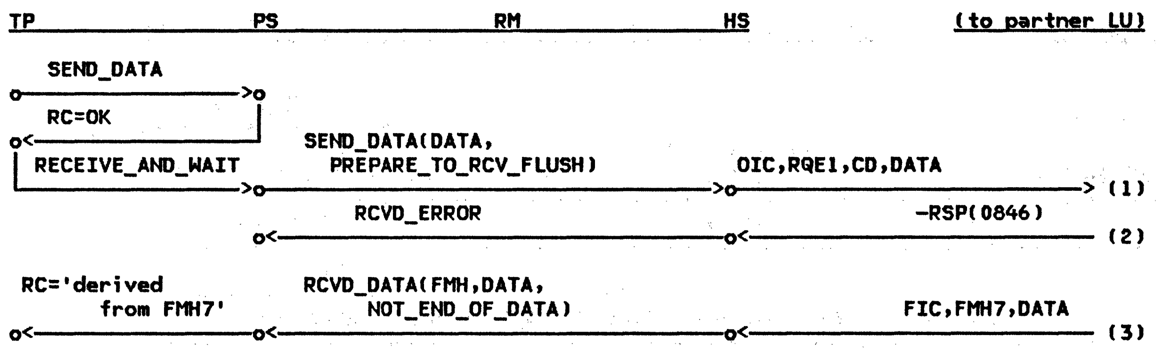


Figure 2-105. SEND_ERROR at End-of-Chain --Remote LU

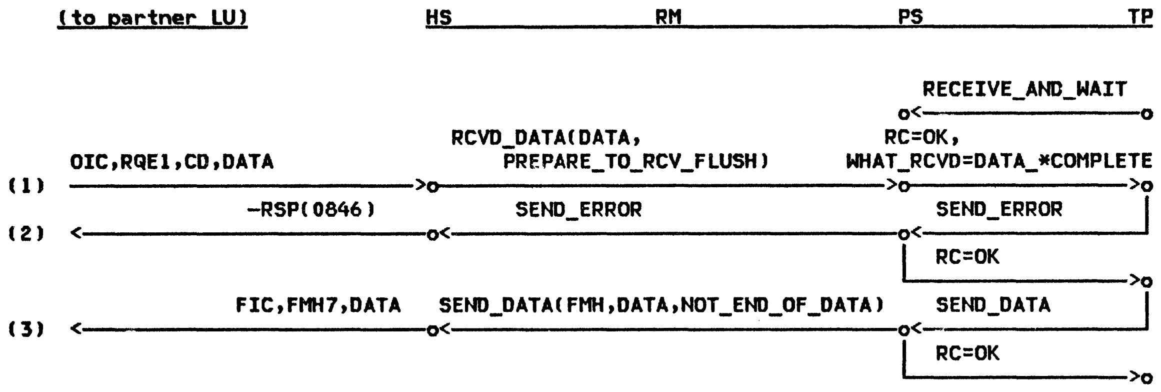


Figure 2-106. SEND_ERROR at End-of-Chain --Local LU

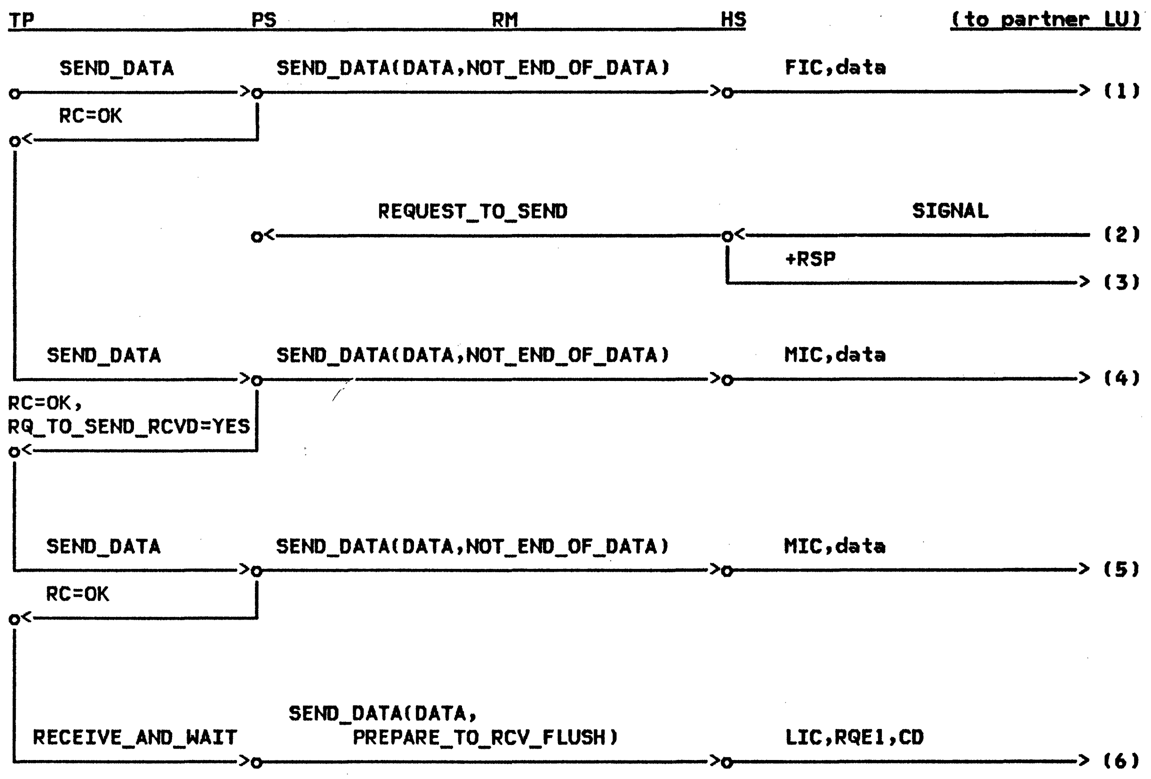


Figure 2-107. REQUEST_TO_SEND, Received in Send State --Remote LU

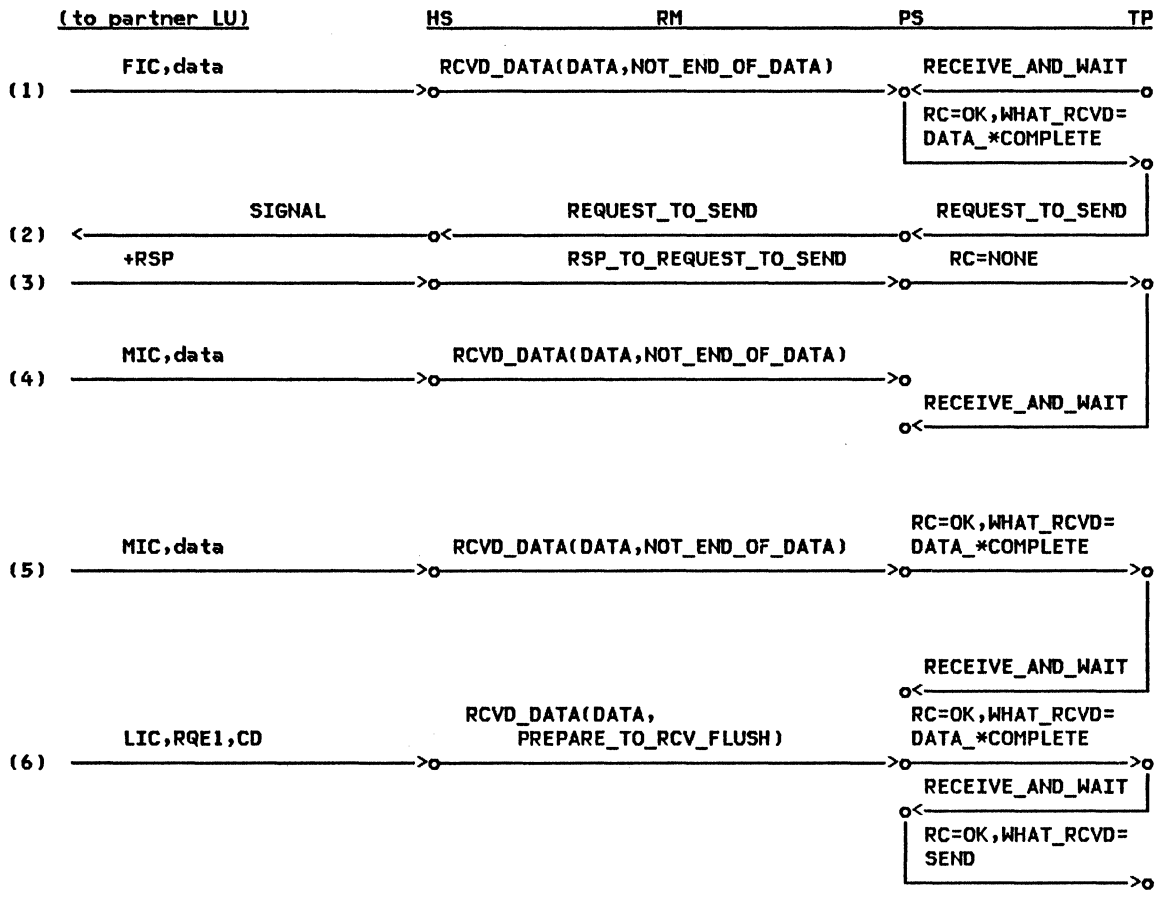


Figure 2-108. REQUEST_TO_SEND, Received in Send State --Local LU

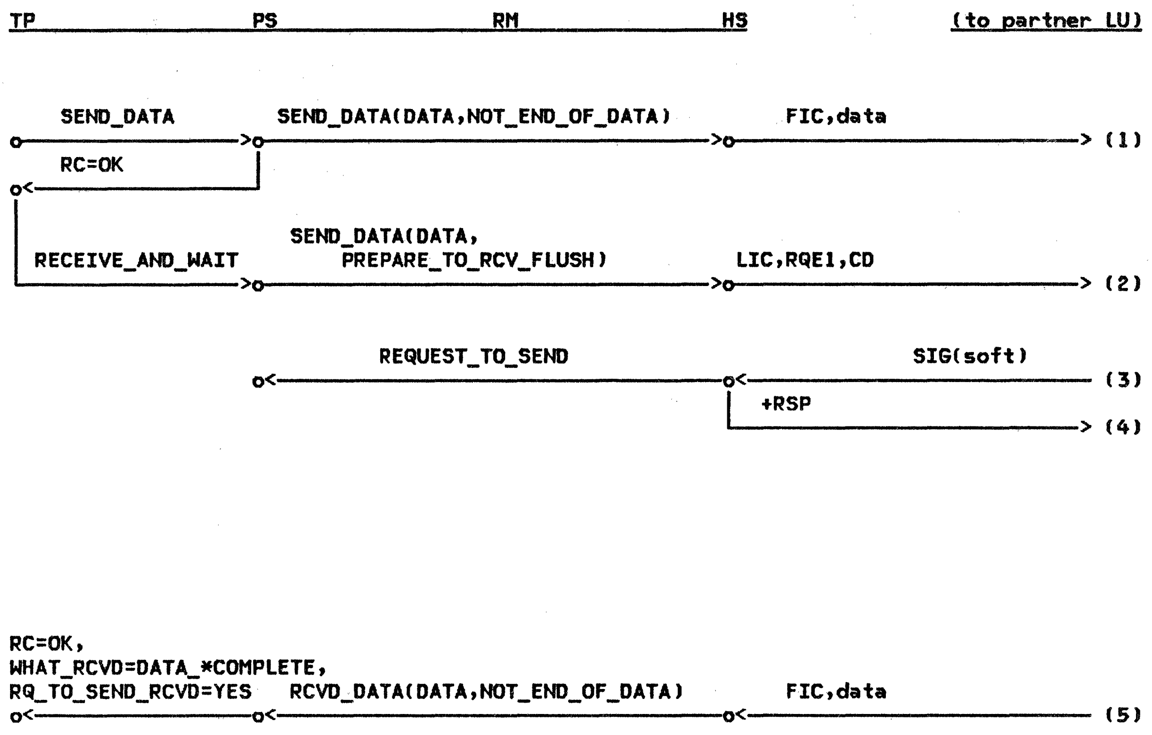


Figure 2-109. REQUEST_TO_SEND, Received in Receive State --Remote LU

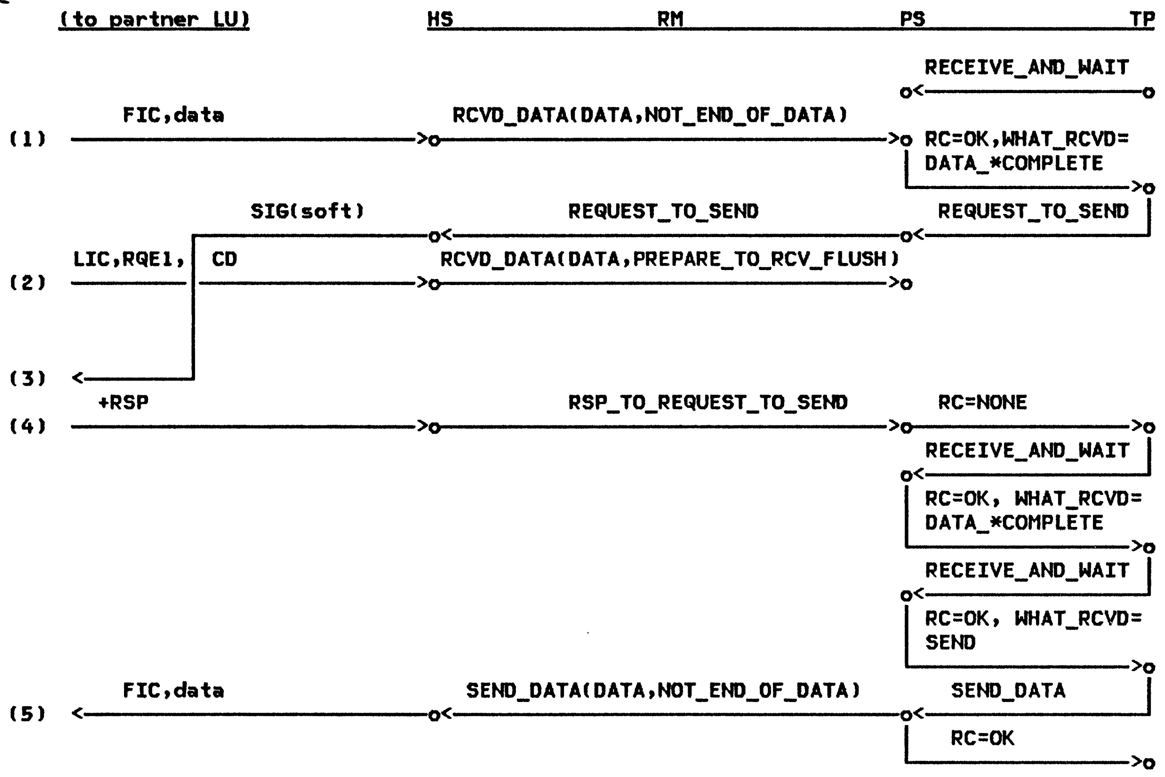


Figure 2-110. REQUEST_TO_SEND, Received in Receive State --Local LU

# CHAPTER 3. LU RESOURCES MANAGER

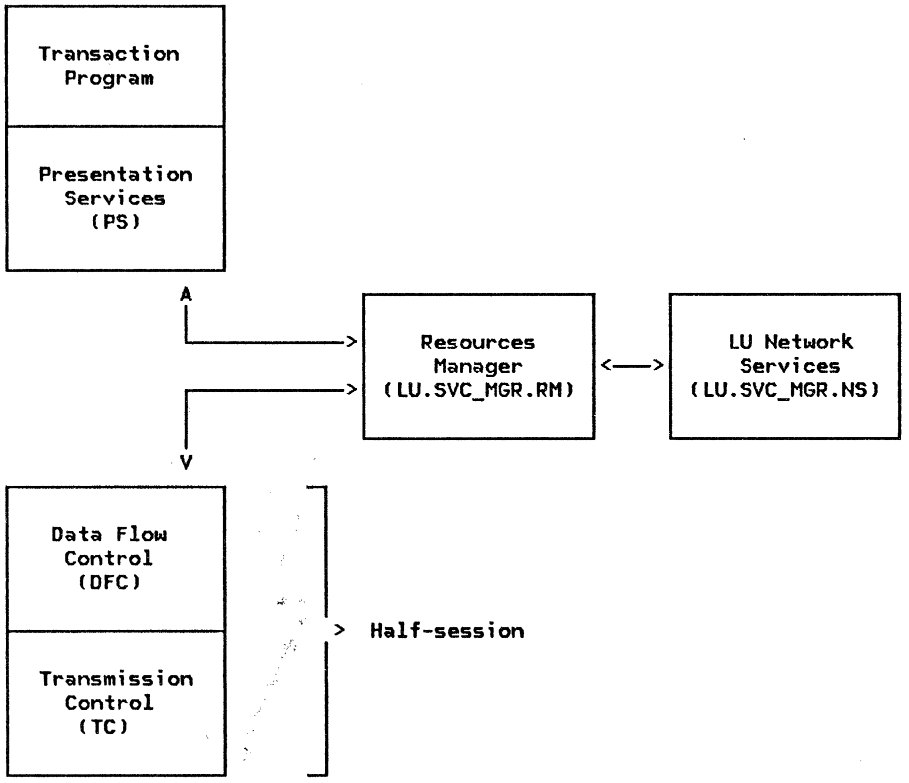

Transaction Program
Presentation Services (PS)
Resources Manager (LU.SVC_MGR.RM)
LU Network Services (LU.SVC_MGR.NS)
Data Flow Control (DFC)
Transmission Control (TC)
Half-session

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

## GENERAL DESCRIPTION

Any time one transaction program wishes to communicate with another, the LU needs to establish, 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 function of the resources manager is to attach new instances of transaction programs when requested to do so by an existing transaction program or the LU operator, to establish a conversation between two transaction programs over an LU-LU session, to later deactivate the conversation and free the session for use by another conversation, and to destroy transaction program instances that have completed processing.
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 the SCB (see "Resources Manager Data-Base" on page 3-2 for additional information).

The resources manager interacts with other components within the LU. 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"), LNS ("Chapter 4. LU Network Services"), and HS ("Chapter 6.0. Half-Session").

## RESOURCES MANAGER FUNCTIONS

The resources manager (RM) coordinates the following functions:

- Creating new instances, and destroying existing instances, of presentation services
- Choosing sessions to be used by a conversation and, if necessary, requesting (bidding for) use of the session
- Requesting network services (LNS) to activate a new session or to deactivate an existing session
- 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. "Chapter 5.3. Presentation Services--Sync Point Services Verbs" describes the content and use of the sync point log.

## COMPONENT INTERACTIONS

Other components 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 network services (LNS). 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 and performing other functions necessary for acquiring the session for use by the requested conversation, such as creating the appropriate control blocks (see "Resources Manager Data-Base" 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 another LU over one of its half-sessions, it 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 3-9 and "Creation and Termination of Presentation Services" on page 3-16 provide additional details).

Data that the resources manager wishes to send to another resources manager in the network is first sent to the local HS component in 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 it finds that there are no free sessions with the required characteristics, the resources manager sends a request to LNS asking it to activate a new session. Similarly, the resources manager sends to LU network services 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. LNS replies to the resources manager after it has carried out the requested function. See "Activating a New Session" on page 3-13 and "Changing the Maximum Session Limit" on page 3-14 for more details on session activation and deactivation.

## RESOURCES MANAGER DATA-BASE

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 buffer 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). There is one SCB for each active session between this LU and a partner LU. Information contained in an SCB includes a half-session identifier (HS_ID) and the partner LU_NAME and 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 fcr the particular LU. The MODE entry contains information that is fixed on a mode name basis such as session counts and limits.

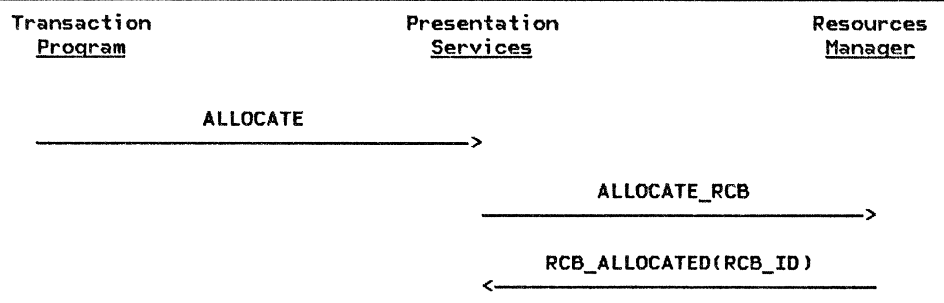


Figure 3-2. Allocation of a Resource Control Block (RCB)

## ESTABLISHING A CONVERSATION

When the resources manager receives an ATTACH_HEADER record (from HS if the Attach was received on an LU-LU session, or from UPM_IPL if the Attach was generated locally, perhaps as the result of an operator command), 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 3-9). Once the transaction program is attached, it can initiate conversations with other transaction programs.

## 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. It 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 3-2 for the flows that take place.

## OBTAINING A SESSION

Once presentation services (PS) is informed of the ID of the new RCB, it creates an Attach FM header (FMH-5) and places it in the RCB. At some point, it requests that an LU-LU session be allocated to the conversation. PS can choose to return control to the transaction program and later obtain the necessary session, or it can obtain the session before returning to the transaction program. PS makes the decision of when to ask for the session based on information the transaction program supplied in the ALLOCATE verb (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. It also contains an indicator that tells the resources manager whether PS wants RM to send out the Attach FM header as part of the session allocation processing, or whether PS is to be responsible for sending the Attach after the session has been allocated by RM.

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 LUs 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. On the other hand, 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.

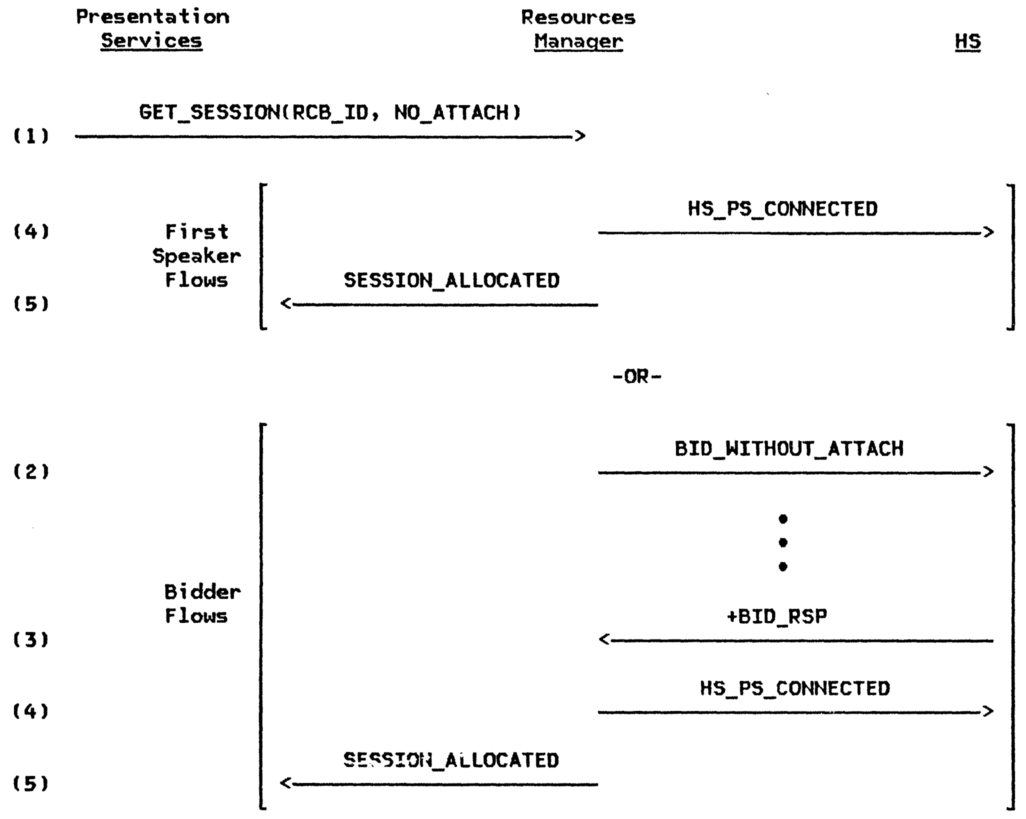


Figure 3-3. Allocation of Session Using BID_WITHOUT_ATTACH

The resources manager will always allocate a first speaker session in preference to a bidder session, to avoid the bidding procedure. Figure 3-3 illustrates the flows that take place when the resources manager attempts to allocate a session and presentation services has specified that the Attach is not to be sent by RM as part the the session allocation. The records used in the figure are defined in "Appendix A. Node Data Structures" in more detail. The following description refers to the numbers to the left of each flow 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. The NO_ATTACH parameter informs the resources manager that it should not send the Attach FM header as part of the session allocation processing

(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 must retain 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 on 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 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 to presentation services to inform it that a session has been allocated to the conversation, satisfying the GET_SESSION request.

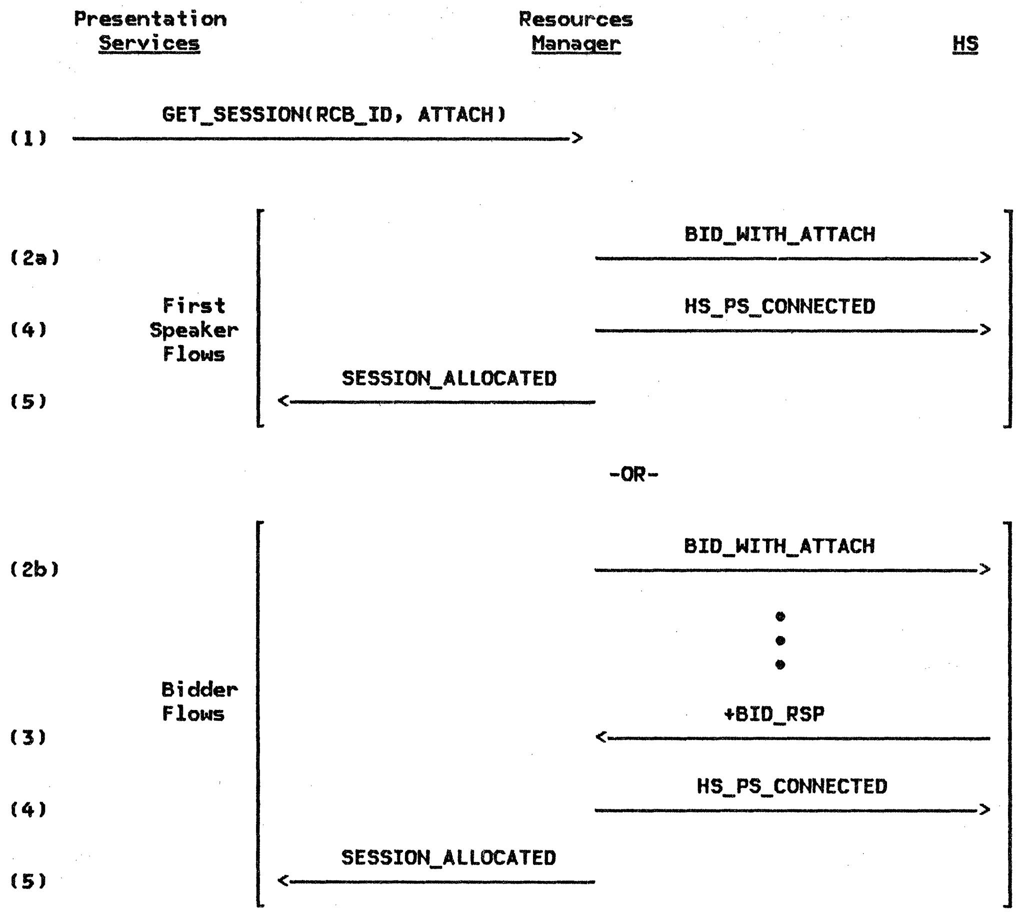


Figure 3-4. Allocation of Session Using BID_WITH_ATTACH

Figure 3-4 illustrates the flows that take place when the resources manager attempts to allocate a session and presentation services has specified that the Attach _is_ to be sent by RM as part the the session allocation. The records used in the figure are fully defined in "Appendix A. Node Data Structures". The following description refers to the numbers to the left of each flow 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. The ATTACH parameter informs the resources manager that it should send the Attach FM header as part of the session allocation processing

(2a) If a first speaker session is available, a BID_WITH_ATTACH to the half-session. The BID_WITH_ATTACH contains the Attach FM header as a field. Since this is a first speaker session, the BID_WITH_ATTACH is not really a bid for the session, and RM may immediately proceed with session allocation without waiting for a BID_RSP (none will be forthcoming).

(2b) If no first speaker session is available, the resources manager must bid for use of a session. It sends BID_WITH_ATTACH to the half-session. BID_WITH_ATTACH includes the Attach FM header as a field. The Attach is sent on the half-session along with the bid. Otherwise, the processing is the same as in Figure 3-3.

(3) Same as in Figure 3-3

(4) Same as in Figure 3-3

(5) Same as in Figure 3-3

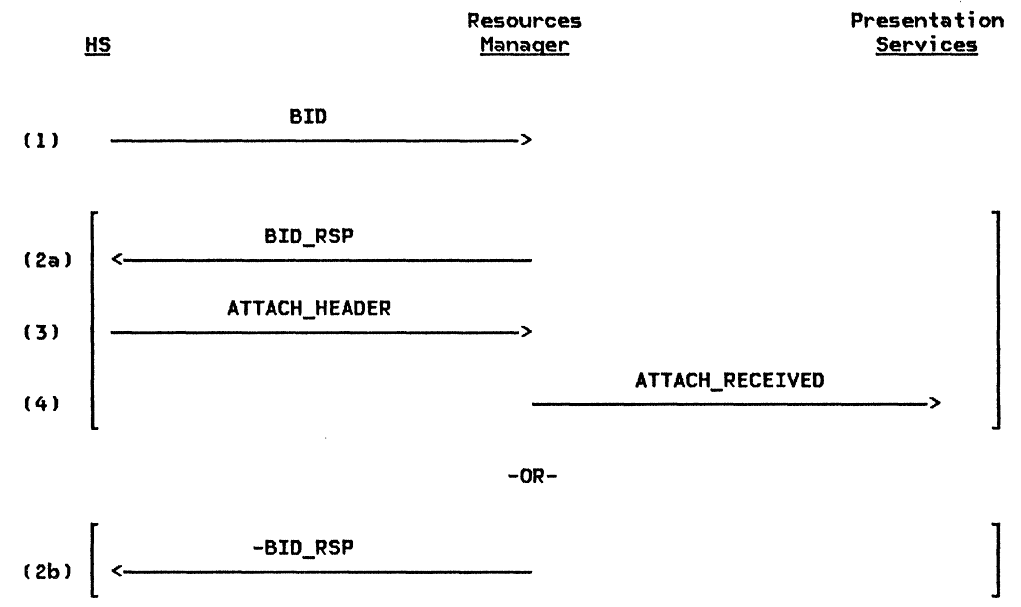


Figure 3-5. Responding to a Bid for a Session

Figure 3-5 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 to the left of each flow 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 (flows 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) An ATTACH_HEADER record, which includes the FMH-5, is sent from the half-session to RM.

(4) RM creates a new TP-PS and sends ATTACH_RECEIVED to PS. See "Attaching a Transaction Program" on page 3-9 for further details.

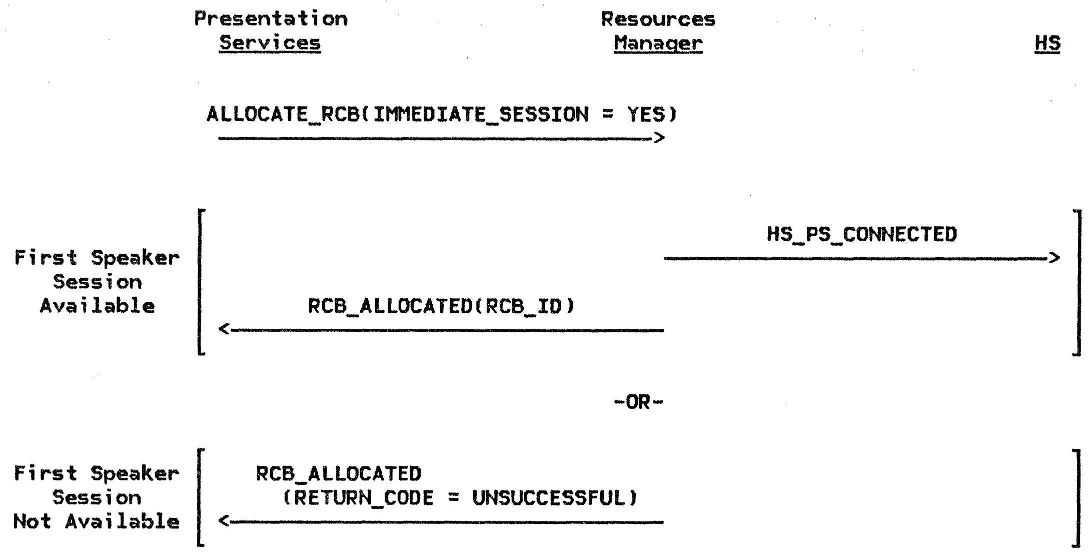

Presentation Services
Resources Manager
HS
ALLOCATE_RCB(IMMEDIATE_SESSION = YES)
First Speaker Session Available
HS_PS_CONNECTED
RCB_ALLOCATED(RCB_ID)
-OR-
First Speaker Session Not Available
RCB_ALLOCATED (RETURN_CODE = UNSUCCESSFUL)

Figure 3-6. 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 to be performed. See Figure 3-6 for the specific flows involved.

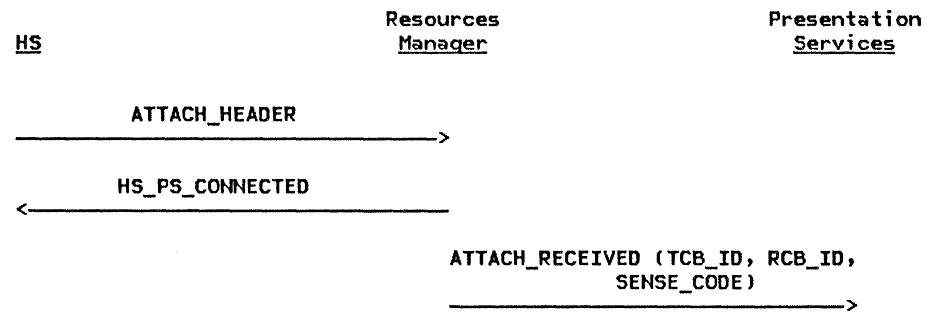


Figure 3-7. Attach Flows

## 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 Attach. Only one Attach is allowed per conversation. RM processes the Attach and later sends it to PS_INITIALIZE in the newly created TP-PS process for further processing.

RM is responsible for checking certain fields of the Attach, such as the transaction program name field. (PS_INITIALIZE later checks the remaining fields). It notifies presentation services of the result of the checking through a field in the ATTACH record that RM sends to PS. Regardless of whether all the fields checked by RM are valid, the resources manager creates a new instance of presentation services and sends to PS the valid or invalid Attach. If the Attach is invalid, presentation services is responsible for creating an FMH-7, or for causing an UNBIND to be generated, to notify the transaction program that initiated the Attach of the error.

After checking the Attach, 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. It then sends an ATTACH record to the new instance of the TP-PS process. The ATTACH record contains the Attach FM header, the FMH-7 sense data field and the IDs of the new TCB and RCB. Finally, it notifies the half-session that it has been connected to a TP-PS process via a HS_PS_CONNECTED record. Figure 3-7 depicts the flows involved in Attach processing.

```
                  First-Speaker                                 Bidder

      Resources                                                         Resources
       Manager                    HS   •••   HS                          Manager


                                                      BID_WITH_ATTACH or
            BID_WITH_ATTACH                           BID_WITHOUT_ATTACH
      ───────────────────────────•                    •──────────────────────────
                                  \                 /
  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
      <──────────────────────────•                    •──────────────────────────>

                                                            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(SENSE_CODE = 0813 or 0814)
      ───────────────────────────────────────────────────────────────────────────>

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

Figure 3-8. 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 BID_RSP. When the bidder RM receives the negative BID_RSP record, it checks the free session pool to see if there is another session available and retries the GET_SESSION processing on that session. Figure 3-8 illustrates an example of a bid race and shows the RCB and SCB settings that allow a race condition to be detected.

The negative BID_RSP 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 -BID_RSP(0813) or -BID_RSP(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 -BID_RSP(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 informs the partner LU that it can bid on the session again by sending an RTR_RQ.

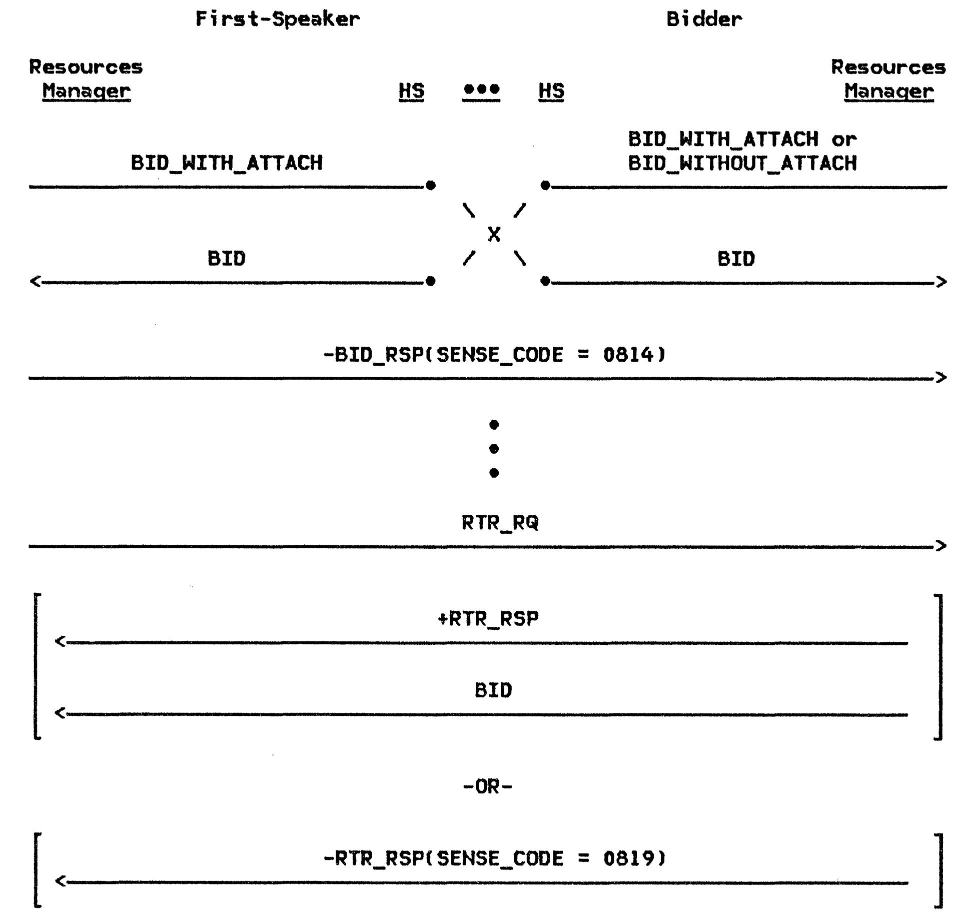


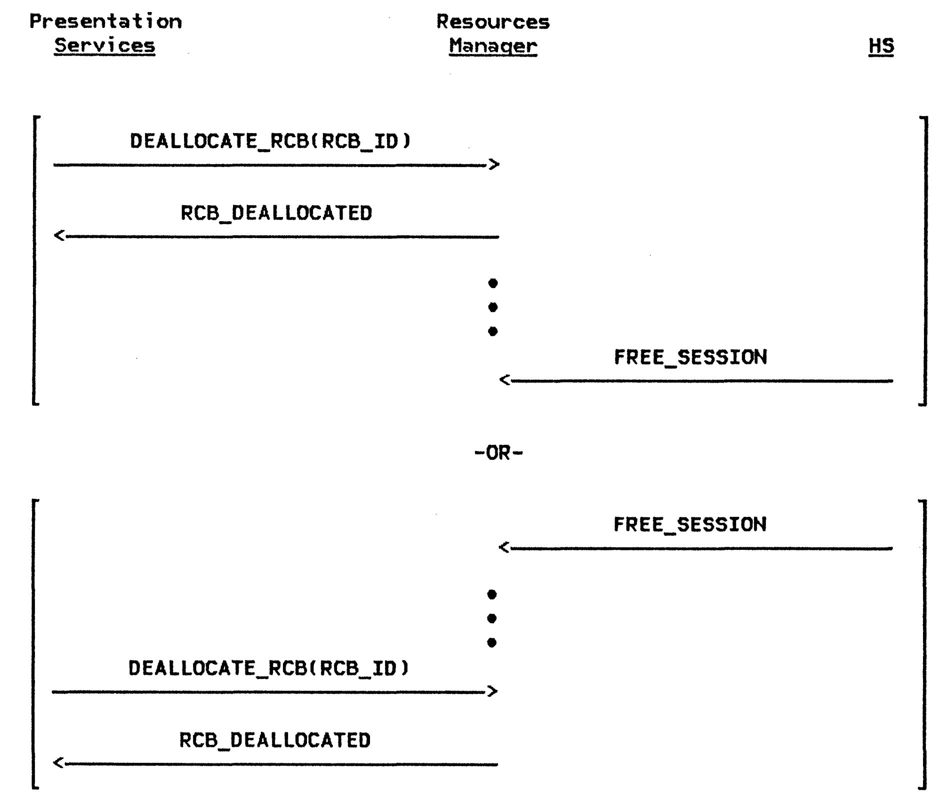
Figure 3-9. READY TO RECEIVE (RTR) Flows

Figure 3-9 depicts possible RTR flows. In the situation where there is a bid race and -BID_RSP(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_RQ from the first-speaker RM. Upon receipt of a -BID_RSP(0814), the bidder resources manager updates a field in the SCB to remember that -RSP(0814) was received and retries the bid on another session. From this point until the RTR_RQ 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_RQ.

At some point, the resources manager at the first-speaker side sends an RTR_RQ 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_RQ, 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_RQ.

When the bidder RM receives the RTR_RQ, it places the session in the free session pool (for the first time since receiving the -BID_RSP(0814)). It then checks to see if a GET_SESSION record is waiting to be serviced, if so RM then sends a positive RTR_RSP (indicating that it intends to use the session) and a BID_WITHOUT_ATTACH or BID_WITH_ATTACH to the first-speaker resources manager. If no GET_SESSION records are waiting, the bidder sends a negative RTR_RSP with a sense code of 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.



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

Figure 3-10. 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 3-10). 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. The arrival of the two records is order-independent. Whichever record is received first triggers the resources manager to disconnect PS and HS.

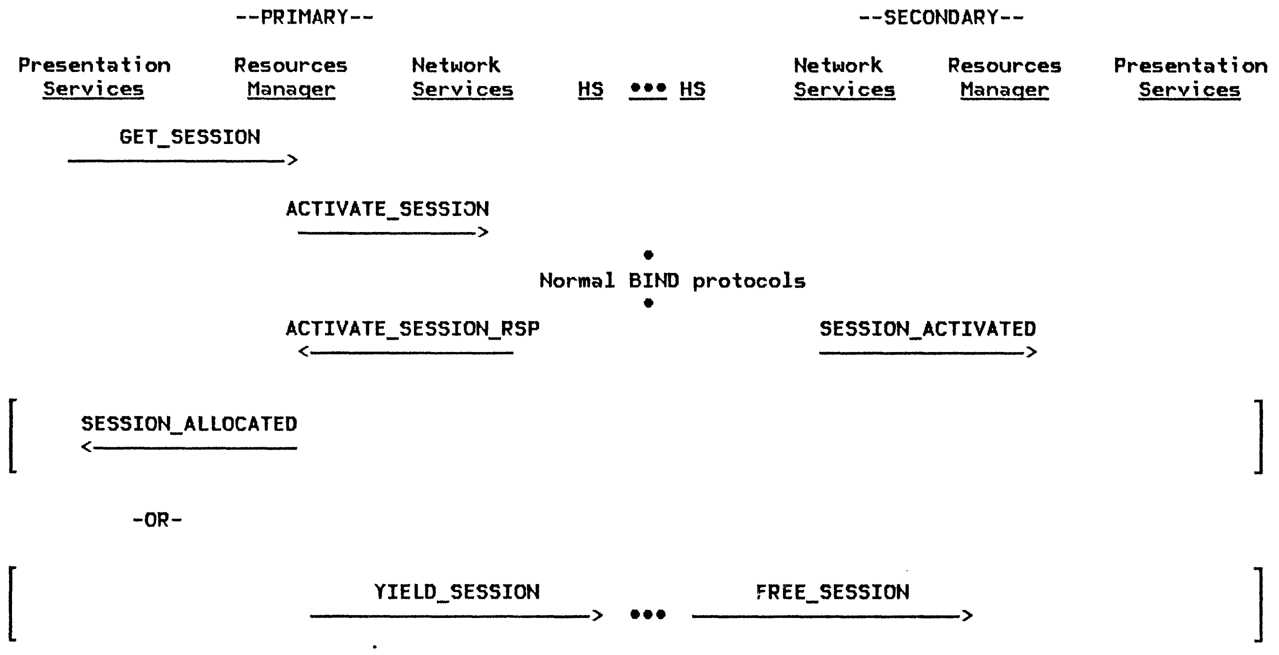


Figure 3-11. Activation of a New Session

## ACTIVATING A NEW SESSION

The resources manager is responsible for allocating 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 LU network services (LNS, "Chapter 4. LU Network Services"). Figure 3-11 illustrates the flows involved in activating a new session.

Although RM will not request session activation if it would cause the session limits to be exceeded, LNS 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 LNS component sends the resources manager a positive ACTIVATE_SESSION_RSP record informing RM of the SCB_ID of the new session. The resources manager at the partner LU is notified of the session activation by a SESSION_ACTIVATED record from its LNS component.

When a new session is activated, it comes up in-brackets 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. 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 then determines if any requests are waiting to be serviced. If so, it uses the new session to service the request and sends a SESSION_ALLOCATED record to presentation services. If no requests are waiting, RM sends a YIELD_SESSION record to HS, thus yielding its right to use the session and ending the bracket.

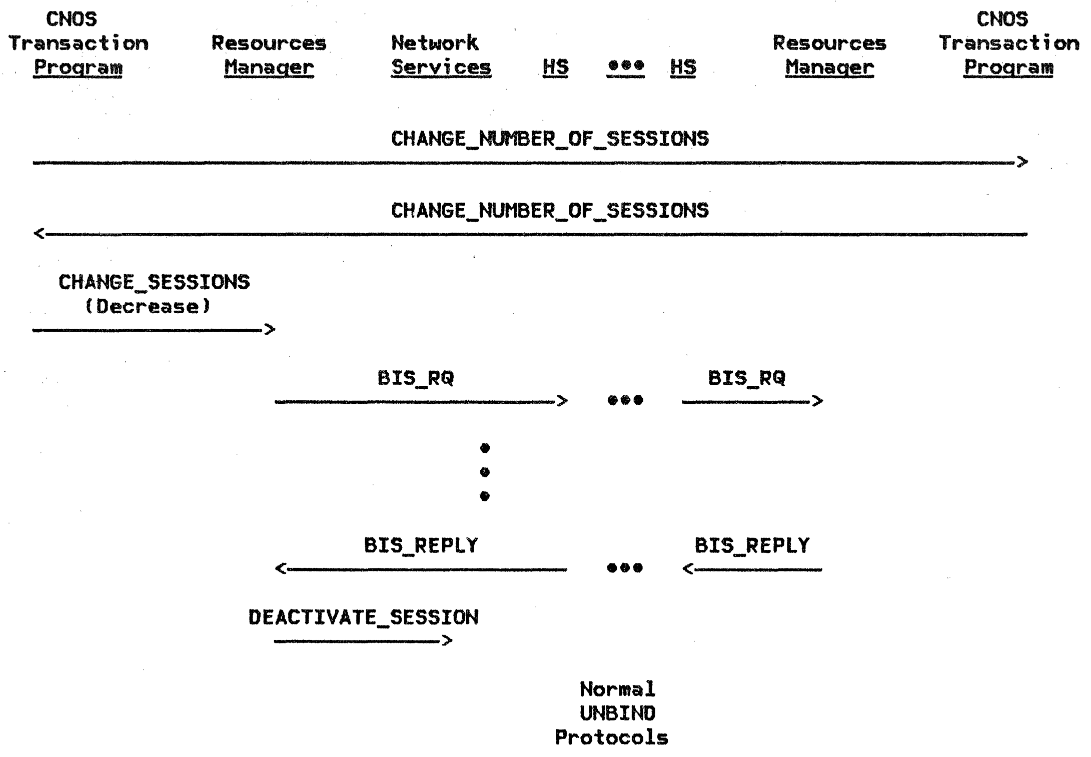


Figure 3-12. Decreasing the Number of Sessions

## 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 this LU can have with the partner LU and mode name represented by the MODE control block. The limit fields include:

- SESSION_LIMIT--limit on the total number of sessions
- MIN_CONWINNERS_LIMIT--limit on the 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--limit on the 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. All such sessions will be first speaker sessions.

The change number of sessions (CNOS) transaction program ("Chapter 5.4. Presentation Services--Control-Operator Verbs") can cause the 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 "Appendix H. FM Header and LU Services Commands"). After an agreement on the new session limits is reached, the 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 occured.

If the new session limits imply that new sessions may be activated, RM determines if there are any waiting requests. If so, it creates multiple ACTIVATE_SESSION records, one for each waiting request, and sends them to LU network services (see "Activating a New Session" on page 3-13 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 a number of sessions equal to the AUTO_ACTIVATIONS_LIMIT are active. After this number of sessions is active, RM will request 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 LU network services only five ACTIVATE_SESSION records.

When the session limits are decreased, one of the LUs is designated 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 a TERMINATION_COUNT, which is the number of sessions that this 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_RQ record on each of the sessions that it has chosen and removing the entry for that session from the free session pool. The BIS_RQ 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 record in response to its BIS_RQ, it decrements the TERMINATION_COUNT and sends to LU network services a DEACTIVATE_SESSION record for that session. LU network services then performs the normal UNBIND protocols. A BIS_RQ-BIS_REPLY exchange always precedes a normal UNBIND (i.e., types X'01', X'02', or X'03'). See Figure 3-12 on page 3-14 for the flows 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 BIS_RQs.

The value of the DRAIN_SELF field in the MODE control block determines whether RM will send BIS_RQ immediately when a session become s free. If DRAIN_SELF = NO (i.e., waiting session allocation requests are not to be satisfied before session deactivation), RM will send BIS_RQ 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_RQ only if there are no waiting requests when the session becomes free. In the same way, DRAIN_SELF determines when BIS_REPLY is sent in reply to a BIS_RQ from the partner LU; i.e., if DRAIN_SELF = NO, BIS_REPLY is sent immediately; otherwise, BIS_REPLY is sent only when there are no waiting requests.

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 BIS_RQ-BIS_REPLY exchange for normal deactivation) to LU network services to satisfy these control-operator requests.

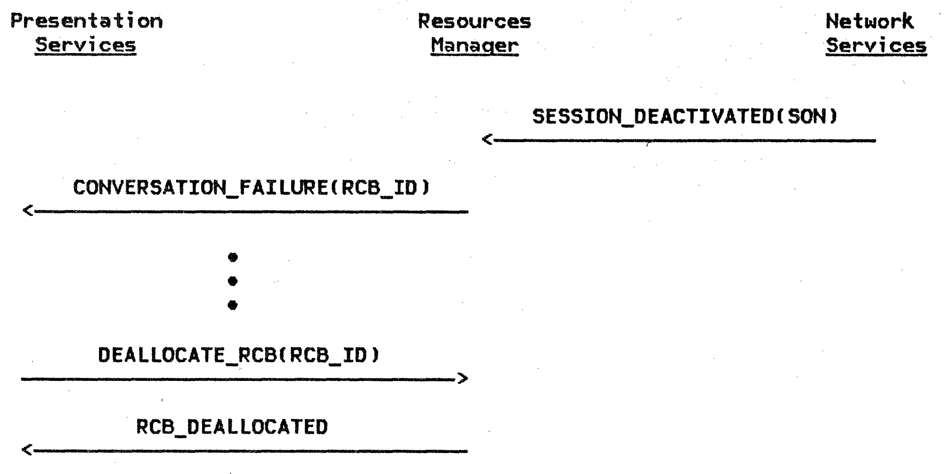


Figure 3-13. Session-Outage Actions

## SESSION OUTAGE

An active session between two LUs sometimes fails. The session outage could 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 LU network services. 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 3-13 illustrates the session-outage flows.

## 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.) The resources manager creates a new instance of presentation services on receipt of an ATTACH_HEADER record. Along with creating a new PS process, RM at this time also creates a new TCB and RCB, and informs PS of the HS_ID of the half-session over which the initial conversation is flowing. Finally, it sends to presentation services the FMH-5 contained in the ATTACH_HEADER record, and the IDs of the new TCB and RCB.

When a transaction program finishes its processing, presentation services notifies the resources manager via a TERMINATE_PS record. RM destroys the PS process and the associated TCB.

## RM

| | |
|---|---|
| FUNCTION: | This process initializes RM_PROCESS_DATA and receives all input to the resources manager and routes the input to the appropriate procedure for processing. |
| INPUT: | RM_RECORD is received asynchronously from network services (LNS), half-session (HS), presentation services (PS), and the undefined protocol machine, UPM_IPL. |
| OUTPUT: | Refer to the procedures that are called from this process for the outputs resulting from records received from other processes. |
| NOTE: | UPM_IPL is an implementation-defined process. It sends an Attach to RM when a transaction program is to be started locally. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PROCESS_LNS_TO_RM_RECORD | page 3-19 |
| PROCESS_HS_TO_RM_RECORD | page 3-18 |
| ATTACH_PROC | page 3-26 |
| PROCESS_PS_TO_RM_RECORD | page 3-20 |

```
Do forever:
   Receive a record.
   Select based on the sender of the record:
      When LNS
         Call PROCESS_LNS_TO_RM_RECORD(record received) (page 3-19).
      When HS
         Call PROCESS_HS_TO_RM_RECORD(record received) (page 3-18).
      When UPM_IPL
         Call ATTACH_PROC(record received, UPM) (page 3-26).
      When PS
         Call PROCESS_PS_TO_RM_RECORD(record received) (page 3-20).
```

## PROCESS_HS_TO_RM_RECORD

| | | |
|---|---|---|
| FUNCTION: | | This procedure routes records received from HS to the appropriate procedure 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 HS_TO_PS_RECORD.HS_ID, 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 -RTR_RSP after having sent BIS_REPLY. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| BID_PROC | page 3-27 |
| BID_RSP_PROC | page 3-29 |
| ATTACH_PROC | page 3-26 |
| FREE_SESSION_PROC | page 3-41 |
| RTR_RQ_PROC | page 3-47 |
| RTR_RSP_PROC | page 3-48 |
| BIS_RQ_PROC | page 3-33 |
| BIS_REPLY_PROC | page 3-32 |
| FSM_BIS_BIDDER | page 3-65 |
| FSM_BIS_FSP | page 3-66 |
| HS_TO_RM_RECORD | page A-13 |
| SCB | page A-9 |

```
If there is not an SCB where SCB.HS_ID = HS_TO_PS_RECORD.HS_ID  then
   Discard the HS_TO_RM_RECORD (see Note 1).
Else
   If the state of #FSM_BIS ≠ CLOSED (page 3-65) then
       Select based on HS_TO_RM_RECORD type:
          When BID
             Call BID_PROC(HS_TO_RM_RECORD) (page 3-27).
          When BID_RSP
             Call BID_RSP_PROC(HS_TO_RM_RECORD) (page 3-29).
          When ATTACH_HEADER
             Call ATTACH_PROC(HS_TO_RM_RECORD, HS) (page 3-26).
          When FREE_SESSION
             Call FREE_SESSION_PROC(HS_TO_RM_RECORD) (page 3-41).
          When RTR_RQ
             Call RTR_RQ_PROC(HS_TO_RM_RECORD) (page 3-47).
          When RTR_RSP
             Call RTR_RSP_PROC(HS_TO_RM_RECORD) (page 3-48).
          When BIS_RQ
             Call BIS_RQ_PROC(HS_TO_RM_RECORD) (page 3-33).
          When BIS_REPLY
             Call BIS_REPLY_PROC(HS_TO_RM_RECORD) (page 3-32).
   Else
     Do nothing (see Note 2).
```

## PROCESS_LNS_TO_RM_RECORD

| | |
|---|---|
| FUNCTION: | This procedure routes records received from NS to the appropriate procedure for processing. |
| INPUT: | LNS_TO_RM_RECORD, the current record from LNS |
| 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 3-22 |
| SESSION_ACTIVATED_PROC | page 3-53 |
| SESSION_DEACTIVATED_PROC | page 3-54 |
| CTERM_DEACTIVATE_SESSION_PROC | page 3-37 |
| LNS_TO_RM_RECORD | page A-19 |

```
Select based on LNS_TO_RM_RECORD type:
   When ACTIVATE_SESSION_RSP
      Call ACTIVATE_SESSION_RSP_PROC(LNS_TO_RM_RECORD) (page 3-22).
   When SESSION_ACTIVATED
      Call SESSION_ACTIVATED_PROC(LNS_TO_RM_RECORD) (page 3-53).
   When SESSION_DEACTIVATED
      Call SESSION_DEACTIVATED_PROC(LNS_TO_RM_RECORD) (page 3-54).
   When CTERM_DEACTIVATE_SESSION
      Call CTERM_DEACTIVATE_SESSION_PROC(LNS_TO_RM_RECORD) (page 3-37).
```

## PROCESS_PS_TO_RM_RECORD

| | |
|---|---|
| FUNCTION: | This procedure routes records received from presentation services to the appropriate procedure 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:

| | |
|---|---|
| ALLOCATE_RCB_PROC | page 3-23 |
| GET_SESSION_PROC | page 3-42 |
| CHANGE_SESSIONS_PROC | page 3-35 |
| RM_ACTIVATE_SESSION_PROC | page 3-45 |
| RM_DEACTIVATE_SESSION_PROC | page 3-46 |
| UNBIND_PROTOCOL_ERROR_PROC | page 3-61 |
| PS | page 5.0-5 |
| PS_TO_RM_RECORD | page A-25 |
| DEALLOCATE_RCB | page A-26 |
| RCB_DEALLOCATED | page A-32 |

```
Select based on PS_TO_RM_RECORD type:
   When ALLOCATE_RCB
      Call ALLOCATE_RCB_PROC(PS_TO_RM_RECORD) (page 3-23).
   When GET_SESSION
      Call GET_SESSION_PROC(PS_TO_RM_RECORD) (page 3-42).
   When DEALLOCATE_RCB
      Discard the RCB with RCB.ID equal to DEALLOCATE_RCB.RCB_ID.
      Build and send an RCB_DEALLOCATED record to PS (Chapter 5.0).
   When TERMINATE_PS
      Discard the TCB and PS corresponding to TERMINATE_PS.TCB_ID.
   When CHANGE_SESSIONS
      Call CHANGE_SESSIONS_PROC(PS_TO_RM_RECORD) (page 3-35).
   When RM_ACTIVATE_SESSION
      Call RM_ACTIVATE_SESSION_PROC(PS_TO_RM_RECORD) (page 3-45).
   When RM_DEACTIVATE_SESSION
      Call RM_DEACTIVATE_SESSION_PROC(PS_TO_RM_RECORD) (page 3-46).
   When UNBIND_PROTOCOL_ERROR
      Call UNBIND_PROTOCOL_ERROR_PROC(PS_TO_RM_RECORD) (page 3-61).
```

## ACTIVATE_NEEDED_SESSIONS

| | |
|---|---|
| FUNCTION: | This procedure activates sessions as required by change-number-of-sessions (CNOS) processing.<br><br>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: | LU_NAME and MODE_NAME, the LU name and mode name, respectively, of the partner LU |
| OUTPUT: | Zero or more ACTIVATE_SESSION records to LNS |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SESSION_ACTIVATION_POLARITY | page 3-53 |
| SEND_ACTIVATE_SESSION | page 3-48 |
| LU_NAME | page 3-69 |
| MODE_NAME | page 3-69 |
| ACTIVATE_SESSION | page A-31 |
| MODE | page A-3 |

```
Get addressability to the MODE control block associated with LU_NAME and
 MODE_NAME.
Do while the number of waiting requests for sessions to (LU_NAME, MODE_NAME)
 is less than MODE.PENDING_SESSION_COUNT, and the polarity returned by
 SESSION_ACTIVATION_POLARITY(LU_NAME, MODE_NAME) (page 3-53) ≠ NONE.
   If polarity = FIRST_SPEAKER then
      Call SEND_ACTIVATE_SESSION(LU_NAME, MODE_NAME, FIRST_SPEAKER) (page 3-48)
       to send an ACTIVATE_SESSION record to LNS.
   Else (BIDDER)
      Call SEND_ACTIVATE_SESSION(LU_NAME, MODE_NAME, BIDDER) (page 3-48).
Do while the minimum of (MODE.AUTO_ACTIVATIONS_LIMIT, MODE.MIN_CONWINNERS_LIMIT) <
 (MODE.ACTIVE_CONWINNERS_COUNT + MODE.PENDING_CONWINNERS_COUNT), and the polarity
 returned by SESSION_ACTIVATION_POLARITY(LU_NAME, MODE_NAME) (page 3-53) = FIRST_SPEAKER.
   Call SEND_ACTIVATE_SESSION(LU_NAME, MODE_NAME, FIRST_SPEAKER) (page 3-48).
```

## 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 LNS |
| OUTPUT: | SESSION_ALLOCATED to PS, or no output |
| 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 3-59 |
| UNSUCCESSFUL_SESSION_ACTIVATION | page 3-62 |
| ACTIVATE_SESSION_RSP | page A-20 |
| MODE | page A-3 |
| MODE_NAME | page 3-69 |

```
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 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.
   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(LU name of pending activation,
       MODE_NAME of pending activation,
       ACTIVATE_SESSION_RSP.SESSION_INFORMATION) (page 3-59).
   Else (negative response)
      Call UNSUCCESSFUL_SESSION_ACTIVATION(LU name of pending activation,
       MODE_NAME of pending activation,
       ACTIVATE_SESSION_RSP.ERROR_TYPE) (page 3-62).
Else
   Do nothing (see Note).
```

## ALLOCATE_RCB_PROC

| | |
|---|---|
| FUNCTION: | This procedure handles the allocation of resource control blocks (RCBs). <br><br> This procedure first creates the RCB_ALLOCATED record, which is sent to PS.CONV to inform it of the outcome of the ALLOCATE_RCB request, 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 before the RCB_ALLOCATED is finally sent to PS.CONV (Chapter 5.1) |
| 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.CONV and the request completes successfully. (If IMMEDIATE_SESSION is set to NO, PS.CONV 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 3-60 |
| CREATE_RCB | page 3-36 |
| PS | page 5.0-5 |
| ALLOCATE_RCB | page A-25 |
| RCB_ALLOCATED | page A-32 |

```
Initialize an RCB_ALLOCATED record with RETURN_CODE set to OK and
 RCB_ID set 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 3-60).
Else
   Call CREATE_RCB(ALLOCATE_RCB, RCB_ALLOCATED) (page 3-36).
Send the RCB_ALLOCATED record to PS.CONV (Chapter 5.1).
```

## ATTACH_CHECK

FUNCTION: This procedure checks particular fields of the passed ATTACH_HEADER for validity. (PS is responsible for checking the remaining fields.)

INPUT: ATTACH_HEADER

OUTPUT: X'00000000', if no error; or sense data returned by ATTACH_LENGTH_CHECK; or one of the following sense data:

| | |
|---|---|
| X'084B6031' | TP Not Available--Retry Allowed |
| X'084C0000' | TP Not Available--No Retry |
| X'1008600B' | Unrecognized FMH Command |
| 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 3-25 |
| ATTACH_HEADER | page A-13 |

```
Call ATTACH_LENGTH_CHECK(ATTACH_HEADER.HEADER) (page 3-25) 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.
Select based on the Command field of the FMH-5:
   When Attach
      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 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).
         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).
      Else
         Return with sense data X'10086021' (TP Name Not Recognized).
   Otherwise
      Return with sense data X'1008600B' (Unrecognized FMH Command).
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 "Appendix H. FM Header and LU Services Commands" )

OUTPUT: Sense data reflecting the result of the length checks. One of the following sense data 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 OFFSET to 5 (offset of Fixed Length Parameters field in Attach).
If the Attach length ≤ 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 OFFSET to OFFSET + the value of the Fixed Length Parameters field + 1
 (offset of TP name Length field).
If the Attach length ≤ OFFSET then
   Return with X'10086000' (FMH Length Not Correct).
Set OFFSET to 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 < OFFSET
      Return with X'10086000' (FMH Length Not Correct).
   When the Attach length = OFFSET
      Return with X'00000000' (Access Security Information and following fields not present).
   When the Attach length > OFFSET (Access Security Information present)
      Do nothing.
If the value of the Access Security Information Length field > 0 then
 (Access Security information is present)
   If 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 OFFSET to 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 < OFFSET
      Return with X'10086000' (FMH Length Not Correct).
   When the Attach length = OFFSET
      Return with X'00000000' (LUW Identifier and following fields not present).
   When the Attach length > 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 OFFSET to OFFSET + the value of the LUW Identifier Length field + 1
 (offset of byte following ATTACH).
If the Attach length < 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. |
| | If the Attach FM header was sent by HS, 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, and to start a new conversation with a new PS process. |
| INPUT: | ATTACH_HEADER and an indicator stating whether the Attach was sent by HS or UPM_IPL |
| OUTPUT: | None |
| NOTES: 1. | RM does not generate a +RSP(Attach). HS does generates a positive BID_RSP, however, so the RM that sent the Attach gets a positive response to the Attach. |
| 2. | If the state of #FSM_SCB_STATUS 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. Although RM can bid with an Attach, HS on the receive side of the half-session converts the BID_WITH_ATTACH record into separate BID and ATTACH_HEADER records. When RM receives the BID, if the half-session is not in use, it changes the status of the half-session to PENDING_ATTACH. |
| 3. | 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:

| | |
|---|---|
| RM_PROTOCOL_ERROR | page 3-46 |
| ATTACH_CHECK | page 3-24 |
| PS_CREATION_PROC | page 3-44 |
| COMPLETE_HS_ATTACH | page 3-33 |
| PS | page 5.0-5 |
| FSM_SCB_STATUS_BIDDER | page 3-63 |
| FSM_SCB_STATUS_FSP | page 3-64 |
| ATTACH_HEADER | page A-13 |
| ATTACH_RECEIVED | page A-32 |
| TCB_ID | page 3-69 |
| RCB_ID | page 3-69 |
| SCB | page A-9 |

```
Set TCB_ID and RCB_ID to null.
Select based on the process that sent the Attach:
   When HS
      Find the SCB corresponding to the HS process that sent the Attach.
      If the state of #FSM_SCB_STATUS ≠ PENDING_ATTACH (page 3-63) then
         Call RM_PROTOCOL_ERROR(SCB.HS_ID, X'20030000') (page 3-46 and see Note 3).
      Else
         Call ATTACH_CHECK(ATTACH_HEADER) (page 3-24) to determine if the
          Attach contains any errors.
         Select based on the Command field of the Attach:
            When Attach (start a new conversation)
               Call PS_CREATION_PROC(ATTACH_HEADER, ATTACH_SENDER, TCB_ID, RCB_ID)
                (page 3-44).
               Call COMPLETE_HS_ATTACH(ATTACH_HEADER.HS_ID, RCB_ID, TCB_ID) (page 3-33).
               Create an ATTACHED_RECEIVED record with ATTACH_RECEIVED.TCB_ID and
                ATTACH_RECEIVED.RCB_ID initialized to TCB_ID and RCB_ID, respectively;
                the SENSE_CODE field initialized to the sense data (if any) set
                by ATTACH_CHECK; and the FMH_5 field initialized to the
                Attach FM header.
               Send the ATTACH_RECEIVED record to PS (Chapter 5.0).
```

```
When UPM
   Call PS_CREATION_PROC(ATTACH_HEADER, ATTACH_SENDER, TCB_ID, RCB_ID) (page 3-44).
   Create an ATTACHED_RECEIVED record with ATTACH_RECEIVED.TCB_ID and
    ATTTACH_RECEIVED.RCB_ID initialized to TCB_ID and RCB_ID, respectively;
    the SENSE_CODE field initialized to X'00000000';
    and the FMH_5 field initialized to the Attach FM header.
   Send the ATTACH_RECEIVED record to PS (Chapter 5.0).
```

## 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 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.

NOTES:

1. RM can bid with an Attach. However, when HS on the receive side of the conversation gets the BID_WITH_ATTACH record, it splits it into two records: a BID and a separate ATTACH_HEADER. Therefore, when RM receives a bid for a session, it will always be via a BID record. When RM receives the ATTACH_HEADER, the state of #FSM_SCB_STATUS is always PENDING_ATTACH.
2. If RM has issued an RTR to the partner LU and has received a positive response to the RTR, the HS_ID of the session over which the RTR flowed will not be free when the BID is received. When RM issued the RTR, it removed that session from the free-session pool.

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| RM_PROTOCOL_ERROR | page 3-46 |
| HS | page 6.0-3 |
| FSM_SCB_STATUS_BIDDER | page 3-63 |
| FSM_SCB_STATUS_FSP | page 3-64 |
| FSM_BIS_BIDDER | page 3-65 |
| FSM_BIS_FSP | page 3-66 |
| BID | page A-14 |
| MODE | page A-3 |
| BID_RSP | page A-28 |

```
If the state of #FSM_BIS is BIS_RCVD (page 3-65) then
   Call RM_PROTOCOL_ERROR(BID.HS_ID, X'20080000') (page 3-46)
    (optional receive check, No Begin Bracket).
Else
   Get addressability to the MODE control block associated with the LU and
    mode name for this session.
   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 3-65) 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 3-63) then
         Call #FSM_SCB_STATUS(R, BID, UNDEFINED) (page 3-63)
          (State of #FSM_SCB_STATUS is PENDING_ATTACH).
         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 RM_PROTOCOL_ERROR(BID.HS_ID, X'20030000') (page 3-46)
             (optional receive check, Bracket Error).
```

BID_RSP_PROC

| | | |
|---|---|---|
| FUNCTION: | | This procedure handles the processing of responses to bids for the use of half-sessions.<br><br>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_RSP, 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.CONV that a session has been successfully allocated via a SESSION_ALLOCATED record.<br><br>When the input is a negative BID_RSP, this procedure changes the RCB so that it no longer points to the SCB that sent the BID_RSP, and retries the GET_SESSION request, which was stored in the RCB when the BID_RQ was issued, on another half-session.<br><br>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.<br><br>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 3-42) |
| 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.CONV 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.CONV, the BID_WITH_ATTACH or BID_WITHOUT_ATTACH record that RM sends to HS also triggers HS to set BBI to BB. Since PS.CONV 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 changes the information in the RCB to indicate to HS that the next record it receives from PS.CONV is not the start of a conversation. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SEND_DEACTIVATE_SESSION | page 3-51 |
| SET_RCB_AND_SCB_FIELDS | page 3-57 |
| CONNECT_RCB_AND_SCB | page 3-34 |
| GET_SESSION_PROC | page 3-42 |
| PS | page 5.0-5 |
| FSM_RCB_STATUS_FSP | page 3-68 |
| FSM_RCB_STATUS_BIDDER | page 3-67 |
| BID_RSP | page A-14 |
| GET_SESSION | page A-26 |
| RCB | page A-7 |
| SESSION_ALLOCATED | page A-33 |

```
If BID_RSP.RTI = NEG and BID_RSP.SENSE_CODE = X'088B0000' then (see Note 2)
  CALL SEND_DEACTIVATE_SESSION(ACTIVE, BID_RSP.HS_ID, CLEANUP, X'00000000')
   (page 3-51).
Else
   Find the RCB associated with the conversation where
    state of #FSM_RCB_STATUS = PENDING_SCB (page 3-67) 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 3-57).
      Call CONNECT_RCB_AND_SCB(RCB.RCB_ID, BID_RSP.HS_ID, REPLY) (page 3-34).
      Set RCB.PS_TO_HS_RECORD.ALLOCATE to NO (see Note 3).
      Create a SESSION_ALLOCATED record with RETURN_CODE set to OK.
      Send the SESSION_ALLOCATED to PS (Chapter 5.1).
   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 3-67).
       (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 3-42).
```

## BIDDER_PROC

| | |
|---|---|
| FUNCTION: | This procedure handles the allocation processing for a bidder half-session.<br><br>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 or a BID_WITH_ATTACH record, depending on an indicator in the GET_SESSION record, and sends it to HS. If PS.CONV instructed RM to bid with an Attach, the Attach and any accompanying data has already been stored in the RCB by PS.CONV before it issued the GET_SESSION request. |
| INPUT: | GET_SESSION and HS_ID, the ID of the bidder half-session that was chosen by GET_SESSION_PROC (page 3-42) |
| OUTPUT: | BID_WITHOUT_ATTACH or BID_WITH_ATTACH to HS. No SESSION_ALLOCATED record is sent to PS.CONV until confirmation that the bidder may use the session is received from the first-speaker (i.e., until a positive BID_RSP is received). |
| 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 3-68 |
| FSM_RCB_STATUS_BIDDER | page 3-67 |
| GET_SESSION | page A-26 |
| HS_ID | page 3-69 |
| BID_WITH_ATTACH | page A-28 |
| BID_WITHOUT_ATTACH | page A-29 |
| RCB | page A-7 |

```
Find the RCB associated with the conversation identified by GET_SESSION.RCB_ID.
Set RCB.HS_ID to HS_ID.
Initialize #FSM_RCB_STATUS to FSM_RCB_STATUS_BIDDER (page 3-67).
Call #FSM_RCB_STATUS(S, GET_SESSION, UNDEFINED) (page 3-67).
Save the contents of the GET_SESSION record in the RCB (see Note).
If GET_SESSION.BID_INDICATOR = ATTACH then
   Build a BID_WITH_ATTACH record where the BID_WITH_ATTACH.SEND_PARM fields
    are initialized with the corresponding RCB.PS_TO_HS_RECORD fields.
   Send the BID_WITH_ATTACH record to HS (Chapter 6.0).
Else (GET_SESSION.BID_INDICATOR ≠ ATTACH)
   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.<br><br>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 |
| 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 3-50 |
| SESSION_DEACTIVATION_POLARITY | page 3-56 |
| HS | page 6.0-3 |
| HS_ID | page 3-69 |
| LU_NAME | page 3-69 |
| MODE_NAME | page 3-69 |
| BIS_REPLY | page A-29 |
| MODE | page A-3 |

```
Let LU_NAME and MODE_NAME be the LU and mode names of the session identified
 by HS_ID.
Get addressability to the MODE control block associated with (LU_NAME, MODE_NAME).
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 3-56).
 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 3-50).
   Remove the session from the free-session pool.
```

## BIS_REPLY_PROC

| | |
|---|---|
| FUNCTION: | This procedure processes BIS replies.<br><br>The procedure invokes #FSM_BIS associated with the half-session over which the BIS_REPLY was received. |
| INPUT: | BIS_REPLY |
| OUTPUT: | #FSM_BIS is invoked |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_BIS_BIDDER | page 3-65 |
| FSM_BIS_FSP | page 3-66 |
| BIS_REPLY | page A-14 |

```
Call #FSM_BIS(R, BIS_REPLY, HS_ID) (page 3-65)
 for the half-session over which the BIS_REPLY was received.
```

## BIS_RQ_PROC

| | |
|---|---|
| FUNCTION: | This procedure processes BIS requests.<br><br>This procedure invokes #FSM_BIS associated with the half-session over which the BIS_RQ was received. |
| INPUT: | BIS_RQ |
| OUTPUT: | #FSM_BIS is invoked |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_BIS_BIDDER | page 3-65 |
| FSM_BIS_FSP | page 3-66 |
| BIS_RQ | page A-14 |

```
Call #FSM_BIS(R, BIS_RQ, HS_ID) (page 3-65)
 associated with the half-session over which the BIS_RQ was received.
```

## COMPLETE_HS_ATTACH

| | |
|---|---|
| FUNCTION: | This procedure performs processing that is required only if the Attach came to RM from HS (as opposed to UPM_IPL).<br><br>The SCB corresponding to the session over which the Attach was received is changed to point to the appropriate RCB, and the status of the SCB is set to IN_USE. |
| INPUT: | HS_ID, the ID of the session from which the Attach was received, RCB_ID, the ID of the conversation resource that is to use the session, and TCB_ID, the ID of the PS that was created as a result of the Attach |
| OUTPUT: | None |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| CONNECT_RCB_AND_SCB | page 3-34 |
| FSM_SCB_STATUS_BIDDER | page 3-63 |
| FSM_SCB_STATUS_FSP | page 3-64 |
| HS_ID | page 3-69 |
| RCB_ID | page 3-69 |
| TCB_ID | page 3-69 |
| SCB | page A-9 |

```
Call #FSM_SCB_STATUS(R, ATTACH, UNDEFINED) (page 3-63)
 associated with the half-session identified by HS_ID.
 (State of #FSM_SCB_STATUS = IN_USE).
Set SCB.RCB_ID to RCB_ID.
Call CONNECT_RCB_AND_SCB(RCB_ID, HS_ID, REPLY) (page 3-34).
```

## 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: | HS_PS_CONNECTED record is sent to HS. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| HS | page 6.0-3 |
| RCB_ID | page 3-69 |
| HS_ID | page 3-69 |
| HS_PS_CONNECTED | page A-29 |

```
Connect the PS process that is using the conversation identified by RCB_ID to
 the half-session identified by HS_ID.
Create an HS_PS_CONNECTED record and send it to HS (Chapter 6.0).
```

## 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.<br><br>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.<br><br>After pending 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 speakers and bidders changes. |
| INPUT: | CHANGE_SESSIONS |
| OUTPUT: | ACTIVATE_SESSION to LNS, BIS_RQ to HS, or none |
| NOTE: | 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. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| CHANGE_SESSIONS | page A-26 |
| MODE | page A-3 |
| SESSION_ALLOCATED | page A-33 |
| DEACTIVATE_PENDING_SESSIONS | page 3-38 |
| DEACTIVATE_FREE_SESSIONS | page 3-38 |
| ACTIVATE_NEEDED_SESSIONS | page 3-21 |

```
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 max(0, MODE.MIN_CONWINNERS_LIMIT - CONWINNER_COUNT).
   Set SESSION_DECREMENT to max(0, PLATEAU - MODE.SESSION_LIMIT).
   Set CONLOSER_INCREMENT to max(0, MODE.MIN_CONLOSERS_LIMIT - CONLOSER_COUNT).
   Set NEED_TO_ACTIVATE to CONWINNER_INCREMENT + CONLOSER_INCREMENT.
   Set ROOM_FOR_ACTIVATION to max(0, MODE.SESSION_LIMIT - PLATEAU).
   Set DECREMENT_FOR_POLARITY to max(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 3-38, see Note).
   If MODE.TERMINATION_COUNT > 0 then
       Call DEACTIVATE_FREE_SESSIONS(CHANGE_SESSIONS.LU_NAME, CHANGE_SESSIONS.MODE_NAME)
        (page 3-38).
If MODE.SESSION_LIMIT = 0, and
 MODE.DRAIN_SELF = NO or MODE.ACTIVE_SESSION_COUNT = 0 then
   Do for each waiting request 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.
      Discard the waiting request.
Call ACTIVATE_NEEDED_SESSIONS(CHANGE_SESSIONS.LU_NAME, CHANGE_SESSIONS.MODE_NAME) to
 activate new sessions if possible and if needed (page 3-21).
```

## 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 |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SEND_BIS_REPLY | page 3-49 |
| HS_ID | page 3-69 |
| MODE | page A-3 |

```
Get addressability to the MODE control block associated with the LU and mode
 name of the session identified by HS_ID.
If MODE.DRAIN_SELF = NO or
 (MODE.DRAIN_SELF = YES and there are no waiting requests for the LU and mode name) then
   If the session identified by HS_ID is free then
      Call SEND_BIS_REPLY(HS_ID) (page 3-49).
      Remove the session from the free-session pool.
```

## CREATE_RCB

| | |
|---|---|
| FUNCTION: | This procedure handles the creation of new RCBs. It places the RCB_ID of the newly created entry into the passed RCB_ALLOCATED record. |
| INPUT: | ALLOCATE_RCB and RCB_ALLOCATED. The RCB_ALLOCATED was created by ALLOCATE_RCB_PROC (page 3-23). |
| OUTPUT: | RCB_ALLOCATED with the RCB_ID field set to the ID of the new RCB |
| 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. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| ALLOCATE_RCB | page A-25 |
| RCB_ALLOCATED | page A-32 |
| RCB | page A-7 |

```
Create RCB, set RCB.RCB_ID to a unique value and RCB.HS_ID to a null value.
Move TCB_ID, LU_NAME, and MODE_NAME from the ALLOCATE_RCB record to the RCB.
Place the RCB_ID in the RCB_ALLOCATED record.
Set #FSM_RCB_STATUS = FSM_RCB_STATUS_FSP (page 3-68; see Note).
Call #FSM_RCB_STATUS(S, ALLOCATE_RCB, UNDEFINED) (state of the FSM is set to FREE,
 page 3-67).
Set RCB.SYNC_LEVEL to ALLOCATE_RCB.SYNC_LEVEL.
Set RCB.PS_TO_HS_RECORD type to SEND_DATA_RECORD and RCB.PS_TO_HS_RECORD data to a null value.
```

## 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:

| | |
|---|---|
| LU_NAME | page 3-69 |
| MODE_NAME | page 3-69 |
| SESSION_INFORMATION | page A-35 |
| SCB | page A-9 |

```
Create an SCB, set SCB.HS_ID to SESSION_INFORMATION.HS_ID, SCB.LU_NAME to
 LU_NAME, SCB.MODE_NAME to MODE_NAME, and SCB.RCB_ID to a null value.
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 3-66)
    and #FSM_SCB_STATUS to FSM_SCB_STATUS_FSP (page 3-64).
Else (bidder session)
   Assign finite-state machines to be used by setting
    #FSM_BIS to FSM_BIS_BIDDER (page 3-65)
    and #FSM_SCB_STATUS to FSM_SCB_STATUS_BIDDER (page 3-63).
```

## CTERM_DEACTIVATE_SESSION_PROC

| | |
|---|---|
| FUNCTION: | This procedure handles the processing that occurs when CTERM_DEACTIVATE_SESSION record is received from LNS. <br><br> The session identified by CTERM_DEACTIVATE_SESSION.HS_ID is deactivated with a BIS_RQ-BIS_REPLY exchange followed by UNBIND(NORMAL). The processing of this record is identical to that of an operator verb DEACTIVATE_SESSION(NORMAL). |
| INPUT: | CTERM_DEACTIVATE_SESSION |
| OUTPUT: | Session deactivation processing is initiated. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| RM_DEACTIVATE_SESSION_PROC | page 3-46 |
| CTERM_DEACTIVATE_SESSION | page A-20 |
| RM_DEACTIVATE_SESSION | page A-27 |

```
Create an RM_DEACTIVATE_SESSION record with TCB_ID set to a null value, SESSION_ID set to
 CTERM_DEACTIVATE_SESSION.HS_ID, and TYPE set to NORMAL.
Call RM_DEACTIVATE_SESSION_PROC(RM_DEACTIVATE_SESSION) (page 3-46).
```

## 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 DEACTIVATE_SESSION records to LNS |
| NOTE: | First-speaker sessions are deactivated before bidder sessions. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SESSION_DEACTIVATION_POLARITY | page 3-56 |
| SEND_DEACTIVATE_SESSION | page 3-51 |
| SEND_BIS | page 3-49 |
| LU_NAME | |
| MODE_NAME | |

```
Do while there exists a free session of a polarity matching that returned by
 SESSION_DEACTIVATION_POLARITY(LU_NAME, MODE_NAME) (page 3-56):
 (If SESSION_DEACTIVATION_POLARITY returns EITHER, a first-speaker session is
 deactivated in preference to a bidder session.)
   Let HS_ID be the identifier of the session to be deactivated.
   Call SEND_BIS(HS_ID) (page 3-49).
   Remove the session from the free-session pool.
```

## 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: | Zero or more DEACTIVATE_SESSION records to LNS |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SESSION_DEACTIVATION_POLARITY | page 3-56 |
| SEND_DEACTIVATE_SESSION | page 3-51 |
| LU_NAME | page 3-69 |
| MODE_NAME | page 3-69 |
| MODE | page A-3 |

```
Get addressability to the MODE control block associated with (LU_NAME, MODE_NAME).
Do while there are pending-active first-speaker sessions for (LU_NAME, MODE_NAME), and
 SESSION_DEACTIVATION_POLARITY(LU_NAME, MODE_NAME) (page 3-56)
 indicates FIRST_SPEAKER or EITHER:
   Call SEND_DEACTIVATE_SESSION(PENDING, PENDING_ACTIVATION.CORRELATOR, NORMAL, X'00000000')
    (page 3-51).
   Decrement MODE.TERMINATION_COUNT by 1.
Do while there are pending-active bidder sessions for (LU_NAME, MODE_NAME), and
 SESSION_DEACTIVATION_POLARITY(LU_NAME, MODE_NAME) (page 3-56)
 indicates BIDDER or EITHER.
   Call SEND_DEACTIVATE_SESSION(PENDING, PENDING_ACTIVATION.CORRELATOR, NORMAL, X'00000000')
    (page 3-51).
   Decrement MODE.TERMINATION_COUNT by 1.
```

## DEQUEUE_WAITING_REQUEST

| | |
|---|---|
| FUNCTION: | This procedure checks to see if there are any GET_SESSION requests waiting to be serviced. If so, this procedure dequeues the first request and invokes GET_SESSION_PROC (page 3-42) to process the request. |
| INPUT: | HS_ID, the ID of a half-session. |
| OUTPUT: | GET_SESSION_PROC is invoked to process the waiting request |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| GET_SESSION_PROC | page 3-42 |
| GET_SESSION | page A-26 |
| HS_ID | page 3-69 |
| LU_NAME | page 3-69 |
| MODE_NAME | page 3-69 |

```
Let LU_NAME and MODE_NAME be the LU name and mode name of the session identified by HS_ID.
If there is a waiting request for a session on (LU_NAME, MODE_NAME) then
   Initialize a GET_SESSION record with the information from the waiting request.
   Call GET_SESSION_PROC(GET_SESSION) (page 3-42) to service the request.
   Remove the waiting request from the queue.
```

## FIRST_SPEAKER_PROC

| | |
|---|---|
| FUNCTION: | This procedure handles the allocation processing for a first-speaker half-session. <br><br> 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. If PS.CONV indicated that RM is to be responsible for sending the Attach, it creates a BID_WITH_ATTACH record from information that PS.CONV stored in the RCB and sends it to HS. It then creates a SESSION_ALLOCATED record, which it sends to PS.CONV to inform PS.CONV 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 3-42) |
| OUTPUT: | SESSION_ALLOCATED to PS; and, if PS.CONV has indicated that RM is to send the Attach for the conversation, BID_WITH_ATTACH to HS |
| NOTE: | Since GET_SESSION_PROC was able to obtain a first-speaker half-session, the Attach that RM sends to HS is not really a bid for the use of the session. After RM sends the Attach it does not have to wait for a response from HS but can report immediately to PS.CONV. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SET_RCB_AND_SCB_FIELDS | page 3-57 |
| CONNECT_RCB_AND_SCB | page 3-34 |
| HS | page 6.0-3 |
| PS | page 5.0-5 |
| GET_SESSION | page A-26 |
| BID_WITH_ATTACH | page A-28 |
| RCB | page A-7 |
| SESSION_ALLOCATED | page A-33 |
| HS_ID | page 3-69 |

```
Call SET_RCB_AND_SCB_FIELDS(GET_SESSION.RCB_ID, HS_ID) (page 3-57).
If GET_SESSION.BID_INDICATOR is ATTACH then
   Create BID_WITH_ATTACH (see Note) with the SEND_PARM subfields initialized
    to the corresponding RCB.PS_TO_HS_RECORD subfields.
   Send the BID_WITH_ATTACH to HS (Chapter 6.0).
Call CONNECT_RCB_AND_SCB(GET_SESSION.RCB_ID, HS_ID, NORMAL) (page 3-34).
Create a SESSION_ALLOCATED record, set RETURN_CODE to OK, and send record to PS
 (Chapter 5.1).
```

## FREE_SESSION_PROC

| | |
|---|---|
| FUNCTION: | This procedure handles the processing that occurs when a session becomes free.<br><br>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: | BIS_RQ, BIS_REPLY, or RTR_RQ to HS; or GET_SESSION to GET_SESSION_PROC (page 3-42); or no output |
| NOTE: | 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:

| | |
|---|---|
| DEQUEUE_WAITING_REQUEST | page 3-39 |
| SHOULD_SEND_BIS | page 3-58 |
| SEND_BIS | page 3-49 |
| HS | page 6.0-3 |
| FSM_SCB_STATUS_BIDDER | page 3-63 |
| FSM_SCB_STATUS_FSP | page 3-64 |
| FSM_BIS_BIDDER | page 3-65 |
| FSM_BIS_FSP | page 3-66 |
| FREE_SESSION | page A-15 |
| SCB | page A-9 |
| RCB | page A-7 |
| RTR_RQ | page A-30 |

```
Find the SCB associated with the session identified by FREE_SESSION.HS_ID.
Set SCB.RCB_ID to a null value.
Call #FSM_SCB_STATUS(R, FREE_SESSION, UNDEFINED) (page 3-63).
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).
Else if RTR is owed on this session then
   If this is a first-speaker session (i.e., this LU owes RTR) then
      If there are no waiting requests for sessions, and
       RTR is to be sent now (implementation-defined choice) then
         Send RTR_RQ to HS (Chapter 6.0).
         Reset RTR owed indication for this session.
   Else (bidder session; i.e., other LU owes RTR) then
      Take no action and return to the calling routine (see Note).
Else
   Call SHOULD_SEND_BIS(SCB.HS_ID) (page 3-58) to determine
    whether BIS should be sent now.
   If BIS should be sent now then
      Call SEND_BIS(SCB.HS_ID) (page 3-49).
   If the state of #FSM_BIS (page 3-65) 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.
      Call DEQUEUE_WAITING_REQUEST(SCB.HS_ID) (page 3-39).
```

## 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 there are no half-sessions available and the current session limit has not been reached, SEND_ACTIVATE_SESSION is called, which requests that LNS activate a new session. |
| INPUT: | GET_SESSION |
| OUTPUT: | See called procedures for output. |
| NOTES: 1. | When PS.CONV requests a session from the resources manager, 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 LU network services 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 there are no sessions 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:

| | |
|---|---|
| FIRST_SPEAKER_PROC | page 3-40 |
| BIDDER_PROC | page 3-31 |
| SESSION_ACTIVATION_POLARITY | page 3-53 |
| SEND_ACTIVATE_SESSION | page 3-48 |
| PS | page 5.0-5 |
| GET_SESSION | page A-26 |
| RCB | page A-7 |
| SESSION_ALLOCATED | page A-33 |

```
If the (GET_SESSION.LU_NAME, GET_SESSION.MODE_NAME) sessions do not support the
 requested sync level then
   Send SESSION_ALLOCATED record with a return code of SYNC_LEVEL_NOT_SUPPORTED.
Else
   If a free session exists then
      If first-speaker half-session then
         Call FIRST_SPEAKER_PROC(GET_SESSION, HS_ID) (page 3-40).
      Else (bidder half-session)
         Call BIDDER_PROC(GET_SESSION, HS_ID) (page 3-31).
      Remove the session from the free-session pool.
   Else (no free session exists)
      If the mode is closed then (see Note 2)
         Send SESSION_ALLOCATED record with a return code of UNSUCCESSFUL_NO_RETRY to
          PS (Chapter 5.1).
      Else
         Call SESSION_ACTIVATION_POLARITY(GET_SESSION.LU_NAME, GET_SESSION.MODE_NAME)
          (page 3-53)
          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)
                   Do nothing.
                When FIRST_SPEAKER
                   Call SEND_ACTIVATE_SESSION(GET_SESSION.LU_NAME, GET_SESSION.MODE_NAME,
                    FIRST_SPEAKER) (page 3-48).
                When BIDDER
                   Call SEND_ACTIVATE_SESSION(GET_SESSION.LU_NAME, GET_SESSION.MODE_NAME,
                    BIDDER) (page 3-48).
         Queue the waiting request for a session.
```

## PS_CREATION_PROC

| | |
|---|---|
| FUNCTION: | This procedure creates a new instance of the PS process.<br><br>This procedure is called upon receipt of an Attach from HS or UPM_IPL. 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 Attach that it received. |
| INPUT: | ATTACH_HEADER, an indicator stating whether the Attach sender was HS or UPM_IPL, and variables in which the TCB_ID and RCB_ID will be returned |
| OUTPUT: | The IDs of the newly created TCB and RCB |
| NOTE: | If the Attach sender is UPM_IPL, the status of the FSM associated with the RCB is set to INITIAL. This indicates that the ATTACH that caused this RCB to be created came from UPM_IPL and that there is no half-session associated with this conversation. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| FSM_RCB_STATUS_FSP | page 3-68 |
| FSM_RCB_STATUS_BIDDER | page 3-67 |
| ATTACH_HEADER | page A-13 |
| TCB | page A-10 |
| RCB | page A-7 |

```
Create a TCB with a unique TCB_ID, initializing TRANSACTION_PROGRAM_NAME
 to null and CONTROLLING_COMPONENT to TP.
Create an RCB with a unique RCB_ID, initializing RCB.TCB_ID to TCB_ID,
 RCB.PS_TO_HS_RECORD.VARIANT_NAME to SEND_DATA_RECORD, RCB.PS_TO_HS_RECORD.DATA
 to null, and RCB.HS_TO_PS_BUFFER_LIST to empty.
Select based on Attach sender:
   When HS
      If the session is a first speaker then
         Set #FSM_RCB_STATUS to FSM_RCB_STATUS_FSP (page 3-68).
      Else
         Set #FSM_RCB_STATUS to FSM_RCB_STATUS_BIDDER (page 3-67).
      Call #FSM_RCB_STATUS(R, ATTACH, HS) (page 3-67)
       (State of #FSM_RCB_STATUS = IN_USE).
      Set RCB.HS_ID to ATTACH_HEADER.HS_ID.
   When UPM_IPL
      Set #FSM_RCB_STATUS to FSM_RCB_STATUS_FSP (page 3-68).
      Call #FSM_RCB_STATUS(R, ATTACH, UPM) (page 3-67)
       (State of #FSM_RCB_STATUS = INITIAL).
      RCB.HS_ID = ATTACH_HEADER.HS_ID;
Create a new PS process (page 5.0-5).
```

## RM_ACTIVATE_SESSION_PROC

| | |
|---|---|
| FUNCTION: | This procedure performs the processing of 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 LNS, or RM_SESSION_ACTIVATED with RETURN_CODE = LU_MODE_SESSION_LIMIT_EXCEEDED to PS |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SESSION_ACTIVATION_POLARITY | page 3-53 |
| SEND_ACTIVATE_SESSION | page 3-48 |
| PS | page 5.0-5 |
| RM_ACTIVATE_SESSION | page A-27 |
| RM_SESSION_ACTIVATED | page A-33 |

```
Create an RM_SESSION_ACTIVATED record.
Call SESSION_ACTIVATION_POLARITY(RM_ACTIVATE_SESSION.LU_NAME,
 RM_ACTIVATE_SESSION.MODE_NAME) (page 3-53)
 to determine the polarity of the next activated session (if any).
Select based on the activation polarity:
   When NONE (session limit exceeded)
      Set RM_SESSION_ACTIVATED.RETURN_CODE to LU_MODE_SESSION_LIMIT_EXCEEDED.
      Send the RM_SESSION_ACTIVATED record to PS (Chapter 5.4).
   When FIRST_SPEAKER
      Call SEND_ACTIVATE_SESSION(RM_ACTIVATE_SESSION.LU_NAME,
       RM_ACTIVATE_SESSION.MODE_NAME, FIRST_SPEAKER) (page 3-48).
      Save the RM_SESSION_ACTIVATED record as a pending CNOS operator activation request.
   When BIDDER
      Call SEND_ACTIVATE_SESSION(RM_ACTIVATE_SESSION.LU_NAME,
       RM_ACTIVATE_SESSION.MODE_NAME, BIDDER) (page 3-48).
      Save the RM_SESSION_ACTIVATED record as a pending CNOS operator activation request.
```

## RM_DEACTIVATE_SESSION_PROC

| | |
|---|---|
| FUNCTION: | This procedure performs the processing of the RM_DEACTIVATE_SESSION record.<br><br>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 SESSION_ID. An RM_DEACTIVATE_SESSION record is also generated internally in RM during the processing of a CTERM_DEACTIVATE_SESSION record from LU network services. |
| INPUT: | RM_DEACTIVATE_SESSION |
| OUTPUT: | DEACTIVATE_SESSION to LNS, BIS_RQ to HS, or no output |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SEND_DEACTIVATE_SESSION | page 3-51 |
| SEND_BIS_RQ | page 3-50 |
| FSM_BIS_BIDDER | page 3-65 |
| FSM_BIS_FSP | page 3-66 |
| RM_DEACTIVATE_SESSION | page A-27 |

```
Select based on RM_DEACTIVATE_SESSION.TYPE:
   When CLEANUP
      Call SEND_DEACTIVATE_SESSION(ACTIVE, RM_DEACTIVATE_SESSION.SESSION_ID,
       CLEANUP, X'00000000') (page 3-51).
   When NORMAL
      If session exists then
         If the session is in use then
            If state of #FSM_BIS (page 3-65) ≠ BIS_SENT then (BIS not already sent)
               Queue the deactivation request.
         Else (session not in use)
            Call SEND_BIS_RQ(HS_ID) (page 3-50).
            Remove the session from the free-session pool.
```

## RM_PROTOCOL_ERROR

| | |
|---|---|
| FUNCTION: | This procedure processes receive error conditions. These errors occur when the other half-session violates the architecture. This procedure takes the following actions:<br><br>• Ends the session by requesting LU network services to send UNBIND. (The other half-session has committed a serious violation of the architecture.) The UNBIND is type X'FE', indicating invalid session protocol, and carries sense data indicating the nature of the receive check error.<br><br>• Notifies the appropriate operator associated with the NAU (the terminal or subsystem operator). Some implementations may not have an appropriate operator to report to.<br><br>• Logs the error. |
| INPUT: | HS_ID, the ID of the half-session and SENSE_CODE, the sense data to be placed in the UNBIND |
| OUTPUT: | See FUNCTION. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SEND_DEACTIVATE_SESSION | page 3-51 |
| HS_ID | page 3-69 |
| SENSE_CODE | page 3-70 |

```
Call SEND_DEACTIVATE_SESSION(ACTIVE, HS_ID, ABNORMAL, SENSE_CODE) (page 3-51).
Log the protocol error.
```

## 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:

| | |
|---|---|
| GET_SESSION_PROC | page 3-42 |
| RM_PROTOCOL_ERROR | page 3-46 |
| SHOULD_SEND_BIS | page 3-58 |
| SEND_BIS | page 3-49 |
| HS | page 6.0-3 |
| RTR_RQ | page A-15 |
| GET_SESSION | page A-26 |
| RTR_RSP | page A-30 |

```
If the partner LU owes an RTR then
   If there are any waiting requests for sessions with the partner LU and mode name then
      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).
      Create a GET_SESSION record from the information saved in the waiting request.
      Call GET_SESSION_PROC(GET_SESSION) (page 3-42) 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 3-58) to determine whether
       BIS should be sent on this session.
      If BIS should be sent then
         Call SEND_BIS(RTR_RQ.HS_ID) (page 3-49).
      Else
         Return the session to the free-session pool.
   Remember that the partner LU no longer owes an RTR.
Else (RTR not expected)
  Call RM_PROTOCOL_ERROR(RTR_RQ.HS_ID, X'20030000') (page 3-46).
```

## RTR_RSP_PROC

| | |
|---|---|
| FUNCTION: | This procedure handles the receipt of RTR responses from a bidder half-session.<br><br>If the input is a positive RTR_RSP, no processing is performed. If the input is a negative RTR_RSP(SENSE_CODE = 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: | None |
| 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 3-39 |
| SHOULD_SEND_BIS | page 3-58 |
| SEND_BIS | page 3-49 |
| FSM_BIS_BIDDER | page 3-65 |
| FSM_BIS_FSP | page 3-66 |
| RTR_RSP | page A-15 |

```
If RTR_RSP.RTI = NEG and the state of #FSM_BIS = RESET (page 3-65) then
 (see Note)
   Call SHOULD_SEND_BIS(RTR_RSP.HS_ID) (page 3-58)
    to determine whether BIS should be sent on this session.
   If SHOULD_SEND_BIS indicates that BIS should be sent then
      Call SEND_BIS(RTR_RSP.HS_ID) (page 3-49).
   Else
      Return the session to the free-session pool.
      Call DEQUEUE_WAITING_REQUEST(RTR_RSP.HS_ID) (page 3-39) to process any waiting requests.
```

## SEND_ACTIVATE_SESSION

| | |
|---|---|
| FUNCTION: | This procedure sends an ACTIVATE_SESSION record to LU network services to request activation of a new 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; and the session polarity (FIRST_SPEAKER or BIDDER) |
| OUTPUT: | ACTIVATE_SESSION to LNS |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LNS | page 4-47 |
| ACTIVATE_SESSION | page A-31 |
| MODE | page A-3 |
| LU_NAME | page 3-69 |
| MODE_NAME | page 3-69 |

```
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.
Increment MODE.PENDING_SESSION_COUNT by 1.
Increment MODE.PENDING_CONWINNERS_COUNT or MODE.PENDING_CONLOSERS_COUNT by 1,
 as appropriate to the session polarity.
Send ACTIVATE_SESSION to LNS (Chapter 4).
```

## 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 3-50 |
| SEND_BIS_REPLY | page 3-49 |
| FSM_BIS_BIDDER | page 3-65 |
| FSM_BIS_FSP | page 3-66 |
| HS_ID | page 3-69 |

```
Select based on the state of #FSM_BIS (page 3-65):
   When RESET
      Call SEND_BIS_RQ(HS_ID) (page 3-50).
   When BIS_RCVD
      Call SEND_BIS_REPLY(HS_ID) (page 3-49).
   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 to HS |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| HS | page 6.0-3 |
| FSM_BIS_BIDDER | page 3-65 |
| FSM_BIS_FSP | page 3-66 |
| BIS_REPLY | page A-29 |
| MODE | page A-3 |
| HS_ID | page 3-69 |

```
Create a BIS_REPLY record and send it to HS (Chapter 6.0).
Call #FSM_BIS(S, BIS_REPLY, HS_ID) (page 3-65) 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.
```

## SEND_BIS_RQ

| | |
|---|---|
| FUNCTION: | This procedure creates a BIS_RQ and sends it to HS.<br><br>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 |
| NOTE: | The TERMINATION_COUNT is not decremented if the BIS_RQ was sent as a result of a control operator DEACTIVATE_SESSION request. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| HS | page 6.0-3 |
| FSM_BIS_BIDDER | page 3-65 |
| FSM_BIS_FSP | page 3-66 |
| BIS_RQ | page A-29 |
| MODE | page A-3 |
| HS_ID | page 3-69 |

```
Create a BIS_RQ record and send it to HS (Chapter 6.0).
Call #FSM_BIS(S, BIS_RQ, HS_ID) (page 3-65) 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 pending CNOS operator session deactivation request for the session
 identified by HS_ID then
   Discard all pending CNOS operator session deactivation request 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 LNS. <br><br> 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. LNS does not send SESSION_DEACTIVATED in reply to DEACTIVATE_SESSION. Thus, the DEACTIVATE_SESSION 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 ≠ ABNORMAL) |
| OUTPUT: | DEACTIVATE_SESSION to LNS |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SESSION_DEACTIVATED_PROC | page 3-54 |
| MODE | page A-3 |
| DEACTIVATE_SESSION | page A-31 |
| SCB | page A-9 |
| SESSION_DEACTIVATED | page A-21 |
| SESSION_ALLOCATED | page A-33 |

```
Select based on the value of session status:
   When PENDING
      If there is a pending session activation 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, and
          DEACTIVATE_SESSION.TYPE set to TYPE.
         If TYPE = ABNORMAL then
            Set DEACTIVATE_SESSION.SENSE_CODE to SENSE_CODE.
         Else
            Set DEACTIVATE_SESSION.SENSE_CODE to X'00000000'.
         Send the DEACTIVATE_SESSION to LNS (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.
         Discard the pending activation.
         If MODE.ACTIVE_SESSION_COUNT + MODE.PENDING_SESSION_COUNT = 0 then
            Do for each waiting request 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.
               Discard the waiting request.
   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, and
          DEACTIVATE_SESSION.TYPE set to TYPE.
         If TYPE = ABNORMAL then
            Set DEACTIVATE_SESSION.SENSE_CODE to SENSE_CODE.
         Else
            Set DEACTIVATE_SESSION.SENSE_CODE to X'00000000'.
         Send the DEACTIVATE_SESSION to LNS (Chapter 4).
         Create a SESSION_DEACTIVATED record with HS_ID set to CORRELATOR.
         If TYPE = NORMAL then
            Set SESSION_DEACTIVATED.REASON to NORMAL.
         Else
            Set SESSION_DEACTIVATED.REASON to ABNORMAL_NO_RETRY.
         Call SESSION_DEACTIVATED_PROC(SESSION_DEACTIVATED) (page 3-54).
```

## SESSION_ACTIVATED_ALLOCATION

| | |
|---|---|
| FUNCTION: | This procedure handles the allocation processing for a newly activated first-speaker or bidder half-session.<br><br>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. If PS.CONV indicated that RM is to be responsible for sending the Attach, it creates a BID_WITH_ATTACH record from information that PS.CONV stored in the RCB and sends it to HS. It then creates a SESSION_ALLOCATED record, which it sends to PS.CONV 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, if PS.CONV has indicated that RM is to send the Attach for the conversation, BID_WITH_ATTACH to HS |
| NOTE: | Since a new session is in the in-brackets state when it is activated, the Attach that RM sends to HS is not really a bid for the use of the session. After RM sends the Attach, it does not have to wait for a response from HS, but can report immediately to PS.CONV. Also, if PS.CONV does not request RM to send the Attach, RM does not send a BID_WITHOUT_ATTACH record to HS even if the half-session is a bidder, since the new session is already in the in-brackets state and no bidding is necessary. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SET_RCB_AND_SCB_FIELDS | page 3-57 |
| CONNECT_RCB_AND_SCB | page 3-34 |
| HS | page 6.0-3 |
| PS | page 5.0-5 |
| FSM_RCB_STATUS_FSP | page 3-68 |
| FSM_RCB_STATUS_BIDDER | page 3-67 |
| GET_SESSION | page A-26 |
| HS_ID | page 3-69 |
| BID_WITH_ATTACH | page A-28 |
| SESSION_ALLOCATED | page A-33 |
| RCB | page A-7 |

```
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 3-67).
     (State of #FSM_RCB_STATUS = PENDING_SCB.)
Call SET_RCB_AND_SCB_FIELDS(GET_SESSION.RCB_ID, HS_ID) (page 3-57).
If GET_SESSION.BID_INDICATOR = ATTACH then
   Find the RCB associated with the conversation identified by GET_SESSION.RCB_ID.
   Create a BID_WITH_ATTACH record with the SEND_PARM fields initialized from
    the corresponding RCB.PS_TO_HS_RECORD fields.
   Send the BID_WITH_ATTACH record to HS (Chapter 6.0; see Note).
Call CONNECT_RCB_AND_SCB(GET_SESSION.RCB_ID, HS_ID, NORMAL) (page 3-34).
Create a SESSION_ALLOCATED record with RETURN_CODE set to OK, and send the
 record to PS (Chapter 5.1).
```

## SESSION_ACTIVATED_PROC

| | |
|---|---|
| FUNCTION: | This procedure performs the processing of a SESSION_ACTIVATED record from LNS. SESSION_ACTIVATED is received from LNS as a result of session activation initiated by the partner LU. |
| INPUT: | SESSION_ACTIVATED from LNS |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SUCCESSFUL_SESSION_ACTIVATION | page 3-59 |
| SESSION_ACTIVATED | page A-20 |
| 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 3-59).
```

## SESSION_ACTIVATION_POLARITY

| | |
|---|---|
| FUNCTION: | This procedure determines the polarity for a session activation request.<br><br>If no session can be activated now (because LUCB.LU_SESSION_LIMIT or 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 will be 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:

| | |
|---|---|
| LU_NAME | page 3-69 |
| MODE_NAME | page 3-69 |
| MODE | page A-3 |

```
Get addressability to the MODE control block associated with LU_NAME and MODE_NAME.
If the number of sessions to the partner LU identified by LU_NAME and
 on mode name identified by MODE_NAME is ≥ MODE.SESSION_LIMIT then
   Return with an indication that no additional sessions can be activated.
If the total number of sessions to the partner LU identified by LU_NAME
 is greater than 0 and parallel sessions are not supported to the
 partner LU identified by LU_NAME 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, 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 = SON or PROTOCOL_VIOLATION and the session was being used by a conversation, this procedure sends a CONVERSATION_FAILURE record to PS.CONV. 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, or no output

NOTES:

1. When PS.CONV receives a CONVERSATION_FAILURE, it generates a DEALLOCATE_RCB and sends it to RM, which performs the usual RCB deallocation processing.
2. 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:

| | |
|---|---|
| GET_SESSION_PROC | page 3-42 |
| ACTIVATE_NEEDED_SESSIONS | page 3-21 |
| PS | page 5.0-5 |
| FSM_SCB_STATUS_BIDDER | page 3-63 |
| FSM_SCB_STATUS_FSP | page 3-64 |
| FSM_RCB_STATUS_FSP | page 3-68 |
| FSM_RCB_STATUS_BIDDER | page 3-67 |
| SESSION_DEACTIVATED | page A-21 |
| CONVERSATION_FAILURE | page A-32 |
| GET_SESSION | page A-26 |
| RM_SESSION_ACTIVATED | page A-33 |
| SCB | page A-9 |
| RCB | page A-7 |
| MODE | page A-3 |

```
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 and
    mode name of the deactivated session.
   If SESSION_DEACTIVATED.REASON ≠ NORMAL then (abnormal deactivation)
      If the state of #FSM_SCB_STATUS (page 3-63) = IN_USE then
         If the RCB identified by the SCB.RCB_ID exists then
            Disconnect the PS and HS processes that are using the deactivated session.
            Create a CONVERSATION_FAILURE record with RCB_ID set to SCB.RCB_ID.
            Select based on SESSION_DEACTIVATED.REASON:
               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 (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 3-67) then
       (A bid for the deactivated session is in progress; see Note 2).
         Set RCB.HS_ID to a null value.
         Call #FSM_RCB_STATUS(R, NEG_BID_RSP, UNDEFINED) (page 3-67).
         Create a GET_SESSION record from information saved in the RCB.
         Call GET_SESSION_PROC(GET_SESSION) (page 3-42)
          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.
   Call ACTIVATE_NEEDED_SESSIONS(SCB.LU_NAME, SCB.MODE_NAME) (page 3-21).
   If MODE.ACTIVE_SESSION_COUNT + MODE.PENDING_SESSION_COUNT = 0 then
      Do for each waiting request 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.
         Discard the waiting request.
      Do for each pending CNOS operator session activation 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.
         Discard the activation request.
   Discard the SCB.
```

## 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: | 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 3-69 |
| MODE_NAME | page 3-69 |
| MODE | page A-3 |

```
Get addressability to the MODE control block associated with LU_NAME and MODE_NAME.
If MODE.TERMINATION_COUNT = 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 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 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 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 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 are initialized. |
| NOTE: | When this procedure is called from BID_RSP_PROC, RCB.HS_ID has already been initialized. (It was initialized when the BID 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 3-63 |
| FSM_SCB_STATUS_FSP | page 3-64 |
| FSM_RCB_STATUS_FSP | page 3-68 |
| FSM_RCB_STATUS_BIDDER | page 3-67 |
| RCB_ID | page 3-69 |
| HS_ID | page 3-69 |
| SCB | page A-9 |
| RCB | page A-7 |

```
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 3-63).
    (State of #FSM_SCB_STATUS = IN_USE.)
   Call #FSM_RCB_STATUS(S, GET_SESSION, UNDEFINED) (page 3-67).
    (State of #FSM_RCB_STATUS = IN_USE.)
Else (bidder session)
   Call #FSM_SCB_STATUS(R, POS_BID_RSP, UNDEFINED) (page 3-63).
    (State of #FSM_SCB_STATUS = IN_USE.)
   Call #FSM_RCB_STATUS(R, BOS_BID_RSP, UNDEFINED) (page 3-67).
    (State of #FSM_RCB_STATUS = IN_USE.)
```

## 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 |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SESSION_DEACTIVATION_POLARITY | page 3-56 |
| FSM_BIS_BIDDER | page 3-65 |
| FSM_BIS_FSP | page 3-66 |
| HS_ID | page 3-69 |
| LU_NAME | page 3-69 |
| MODE_NAME | page 3-69 |
| MODE | page A-3 |

```
Get addressability to the MODE control block associated with the half-session
 identified by HS_ID.
SELECT based on the state of #FSM_BIS (page 3-65):
   When RESET
      Call SESSION_DEACTIVATION_POLARITY(LU_NAME, MODE_NAME) (page 3-56)
       to determine the type of session (if any) to deactivate.
      If the deactivation polarity = EITHER, or
       the deactivation polarity matches the session polarity then
        If MODE.DRAIN_SELF = NO, or
         MODE.DRAIN_SELF = YES and there are no waiting requests for
         sessions to this LU and mode name then
           Return with an indication that BIS should be sent on this session.
        If there is a pending CNOS operator session deactivation for this session then
           Return with an indication that BIS should be sent on this session.
        Else
           Return with an indication that BIS should not be sent on this session.
   When BIS_RCVD
      If MODE.DRAIN_SELF = NO, or
       MODE.DRAIN_SELF = YES and there are no waiting requests for sessions
       to this LU and mode name then
         Return with an indication that BIS should be sent on this session.
      Else
         Return with an indication that BIS should not be sent on this session.
   When BIS_SENT (BIS already sent)
      Return with an indication that BIS should not be sent on this session.
```

## SUCCESSFUL_SESSION_ACTIVATION

| | |
|---|---|
| FUNCTION: | This procedure handles the processing that occurs when a new session is successfully activated.<br><br>When a new session is successfully activated, it 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 a primary and there is a waiting request, the support levels (i.e., sync level) specified in the request are checked against the support levels of the session. If the support levels are compatible, the request is sent to SESSION_ACTIVATED_ALLOCATION (page 3-52) to be processed. If the support levels are not compatible, the request is rejected with an ALLOCATION_ERROR return code. If there are no waiting requests, the session is returned to the free-session pool and a YIELD_SESSION record is created and sent to HS to inform the secondary side of the half-session that the primary side 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.<br><br>If the new half-session is a secondary half-session, 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-35), which describes the attributes of the activated session |
| OUTPUT: | GET_SESSION to SESSION_ACTIVATED_ALLOCATION (page 3-52), YIELD_SESSION to HS, SESSION_ALLOCATED to PS, or no output |
| NOTE: | PS.CONV stores in the RCB information that tells HS what bit settings to use when HS sends data out over a link. Part of the information states 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 that is unknown by PS.CONV), RM changes the information in the RCB to indicate to HS that the next record it will receive from PS.CONV will not be the start of a conversation. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| CREATE_SCB | page 3-37 |
| SESSION_ACTIVATED_ALLOCATION | page 3-52 |
| PS | page 5.0-5 |
| HS | page 6.0-3 |
| FSM_SCB_STATUS_BIDDER | page 3-63 |
| FSM_SCB_STATUS_FSP | page 3-64 |
| LU_NAME | page 3-69 |
| MODE_NAME | page 3-69 |
| SESSION_INFORMATION | page A-35 |
| SCB | page A-9 |
| RM_SESSION_ACTIVATED | page A-33 |
| GET_SESSION | page A-26 |
| YIELD_SESSION | page A-30 |
| SESSION_ALLOCATED | page A-33 |

SUCCESSFUL_SESSION_ACTIVATION

```
Call CREATE_SCB(LU_NAME, MODE_NAME, SESSION_INFORMATION) (page 3-37).
If this is a primary half-session then
   Call #FSM_SCB_STATUS(R, SESSION_ACTIVATED, PRI) (page 3-63).
    (State of #FSM_SCB_STATUS = SESSION_ACTIVATION).
   Do until the activated session is used to service a waiting request, or
    the session is yielded.
      If there is a waiting request for this LU and mode name then
         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.
            Discard the waiting request.
         Else (session support is OK)
            Create a GET_SESSION record initialized with information from
             the waiting request.
            Call SESSION_ACTIVATED_ALLOCATION(GET_SESSION, SCB.HS_ID) (page 3-52).
            Discard the waiting request.
      Else (no waiting requests)
         Call #FSM_SCB_STATUS(S, YIELD_SESSION, UNDEFINED) (page 3-63).
         Create a YIELD_SESSION record and send it to the HS (Chapter 6.0)
          representing the newly activated session.
Else (secondary half-session)
   Call #FSM_SCB_STATUS(R, SESSION_ACTIVATED, SEC) (page 3-63).
    (State of #FSM_SCB_STATUS = PENDING_ATTACH).
If there is a pending CNOS operator session activation request 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.
```

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 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. If the support levels are compatible, the half-session is allocated to the RCB, the ID of the RCB is placed in the passed RCB_ALLOCATED record.<br><br>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. RCB_ALLOCATED was 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 3-36 |
| SET_RCB_AND_SCB_FIELDS | page 3-57 |
| CONNECT_RCB_AND_SCB | page 3-34 |
| ALLOCATE_RCB | page A-25 |
| RCB_ALLOCATED | page A-32 |

```
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 3-36).
   If sync level requested in ALLOCATE_RCB is not supported by 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 3-57).
      Call CONNECT_RCB_AND_SCB(RCB_ID, HS_ID) (page 3-34).
      Remove the session from the free-session pool.
Else (no free first-speaker sessions).
   set RCB_ALLOCATED.RETURN_CODE to UNSUCCESSFUL.
```

UNBIND_PROTOCOL_ERROR_PROC

| | |
|---|---|
| FUNCTION: | This procedure processes an UNBIND_PROTOCOL_ERROR record from a presentation services component. Presentation services sends an UNBIND_PROTOCOL_ERROR to the resources manager when it discovers that the other side of the half-session has committed a protocol violation. |
| INPUT: | UNBIND_PROTOCOL_ERROR |
| OUTPUT: | The session identified by UNBIND_PROTOCOL_ERROR.HS_ID is deactivated with an UNBIND(X'FE') and sense data specified by UNBIND_PROTOCOL_ERROR.SENSE_CODE. The resources manager sends a CONVERSATION_FAILURE(PROTOCOL_VIOLATION) to presentation services. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| RM_PROTOCOL_ERROR | page 3-46 |
| UNBIND_PROTOCOL_ERROR | page A-28 |

Call RM_PROTOCOL_ERROR(UNBIND_PROTOCOL_ERROR.HS_ID, UNBIND_PROTOCOL_ERROR.SENSE_CODE) (page 3-46).

## UNSUCCESSFUL_SESSION_ACTIVATION

| | |
|---|---|
| FUNCTION: | This procedure handles the processing that occurs when a new session could not be activated by LU network services.<br><br>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 previously allocated session(s) will eventually be available for use by the transaction program(s) 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 network services 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.<br><br>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 network services), 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.<br><br>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, or no output |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| LU_NAME | page 3-69 |
| MODE_NAME | page 3-69 |
| MODE | page A-3 |
| RM_SESSION_ACTIVATED | page A-33 |
| SESSION_ALLOCATED | page A-33 |

```
Get addressability to the MODE control block associated with LU_NAME and MODE_NAME.
If MODE.ACTIVE_SESSION_COUNT = 0 and MODE.PENDING_SESSION_COUNT = 0 then
     Do for each waiting request for a session to (LU_NAME, 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.
        Discard the waiting request.
     Do for each pending CNOS operator session activation request for a
      session to (LU_NAME, MODE_NAME):
        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.
        Discard the pending session activation 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 3-37).

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-brackets 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

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_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 half-session receives an Attach, it converts the Attach into separate BID and ATTACH_HEADER records. 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_HEADER to RM. RM on the first-speaker side does not generate the separate BID and ATTACH_HEADER records, and furthermore does not expect a BID_RSP since a first-speaker half-session always gains access to the session.

3. A YIELD_SESSION will move the FSM from SESSION_ACTIVATION state to the IN_USE state. A FREE_SESSION is expected from the half-session to then change the state to FREE.

| STATE NAMES----><br>INPUTS STATE NUMBERS--> | SESSION ACTIVATION 01 | FREE 02 | PENDING ATTACH 03 | IN USE 04 |
|---|---|---|---|---|
| R, POS_BID_RSP | 4 | 4 | / | / |
| R, BID<br>R, ATTACH | /<br>/ | 3<br>/ | /<br>4 | /<br>/ |
| R, FREE_SESSION | / | / | 2 | 2 |
| S, YIELD_SESSION | 4 | / | / | / |
| R, SESSION_ACTIVATED, PRI<br>R, SESSION_ACTIVATED, SEC | -<br>3 | /<br>/ | /<br>/ | /<br>/ |

## 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 will move the FSM from SESSION_ACTIVATION state to the IN_USE state. A FREE_SESSION is expected from the half-session to then change the state to FREE.

| STATE NAMES----> <br> INPUTS STATE NUMBERS--> | SESSION ACTIVATION 01 | FREE 02 | PENDING ATTACH 03 | IN USE 04 |
|---|---|---|---|---|
| S, GET_SESSION | 4 | 4 | / | / |
| R, BID | / | 3 | / | / |
| R, ATTACH | / | / | 4 | / |
| R, FREE_SESSION | / | / | 2 | 2 |
| S, YIELD_SESSION | 4 | / | / | / |
| R, SESSION_ACTIVATED, PRI | - | / | / | / |
| R, SESSION_ACTIVATED, SEC | 3 | / | / | / |

#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 3-37).

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 3-51 |
| CHECK_FOR_BIS_REPLY | page 3-36 |
| BIS_RACE_LOSER | page 3-32 |
| RM_PROTOCOL_ERROR | page 3-46 |
| HS_ID | page 3-69 |

| 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 3-51). |
| B | Call CHECK_FOR_BIS_REPLY(HS_ID) (page 3-36). |
| C | Call BIS_RACE_LOSER(HS_ID) (page 3-32). |
| ERROR | Call RM_PROTOCOL_ERROR(HS_ID, X'20100000') (page 3-46). |

## 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 3-51 |
| CHECK_FOR_BIS_REPLY | page 3-36 |
| RM_PROTOCOL_ERROR | page 3-46 |
| HS_ID | page 3-69 |

| STATE NAMES----> | RESET | BIS SENT | BIS RCVD | CLOSED |
|---|---|---|---|---|
| INPUTS STATE NUMBERS--> | 01 | 02 | 03 | 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 3-51). |
| B | Call CHECK_FOR_BIS_REPLY(HS_ID) (page 3-36). |
| ERROR | Call RM_PROTOCOL_ERROR(HS_ID, X'20100000') (page 3-46). |

## #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. #FSM_RCB_STATUS is initialized to either FSM_RCB_STATUS_BIDDER or FSM_RCB_STATUS_FSP, depending on the polarity of the underlying session, resources manager creates the conversation resource. This initialization occurs in BIDDER_PROC (page 3-31), CREATE_RCB (page 3-36), and PS_CREATION_PROC (page 3-44).

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)
- INITIAL (FSP only)--the conversation is the initial conversation established by UPM_IPL

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_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. |

| STATE NAMES----> | FREE | IN USE | PENDING SCB |
|---|---|---|---|
| INPUTS STATE NUMBERS--> | 01 | 02 | 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----> | FREE | IN USE | INITIAL |
|---|---|---|---|
| INPUTS STATE NUMBERS--> | 01 | 02 | 03 |
| S, ALLOCATE_RCB | - | / | / |
| S, GET_SESSION | 2 | / | / |
| R, ATTACH, HS | 2 | / | / |
| R, ATTACH, UPM | 3 | / | / |
| S, DEALLOCATE_RCB | - | 1 | 1 |

LOCAL DATA STRUCTURES

| LU_NAME |
|---|

LU_NAME: LU name

| MODE_NAME |
|---|

MODE_NAME: mode name

| HS_ID |
|---|

HS_ID: half-session identifier

| RCB_ID |
|---|

RCB_ID: conversation resource identifier

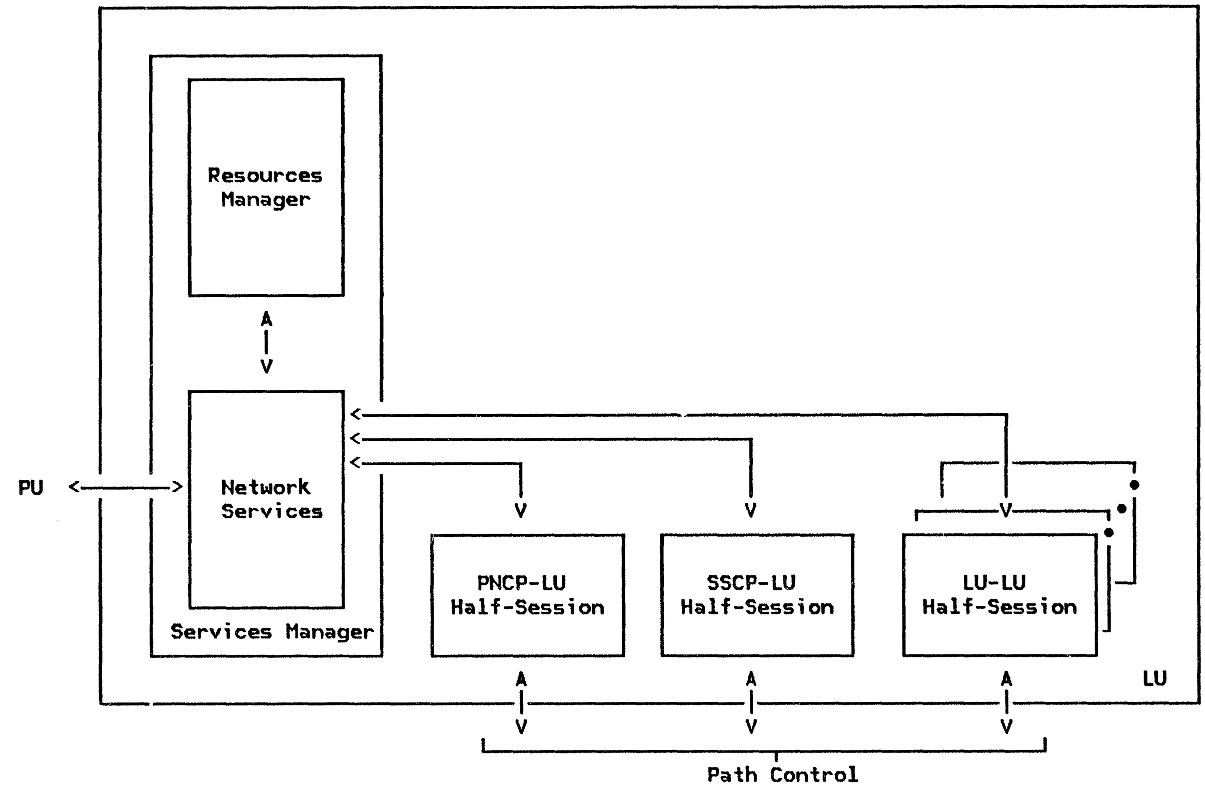
| TCB_ID |
|---|

TCB_ID: TP-PS process identifier

| |
|---|
| SENSE_CODE |

SENSE_CODE: 4-byte sense data

# CHAPTER 4. LU NETWORK SERVICES



Notes:
- Only the LU components having a protocol boundary with LU network services are shown.
- The PNCP-LU half-session is present only in peripheral nodes.

Figure 4-1. Protocol Boundaries Between LU Network Services and Other Components

## GENERAL DESCRIPTION

This chapter describes the network services component within an LU. Figure 4-1 shows the LU network services component and its relation to other components within the node. The arrows joining the components represent the protocol boundaries that exist between the LU network services component and the other components.

The LU network services component (abbreviated LNS) initiates and terminates LU-LU sessions in response to requests from the resources manager and from the remote LU. LNS also activates and deactivates CP-LU sessions.

The initiation and termination of LU-LU sessions involves exchanging session-services RUs between the LU and a 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 LU-LU sessions. The exchange of session-services RUs precedes and follows the activation and deactivation of the LU-LU sessions.

Session-control requests and responses are sent on the expedited flow with the RU category indicating session control (SC). Session-control RUs are sent field-formatted.

Full details of the formats for field-formatted RUs are given in "Appendix E. Request-Response Unit (RU) Formats".

Session-services requests and responses belong to the network-services (NS) format of RUs. All session-services requests and responses are sent on the normal flow with the RU category indicating FM data (FMD).

Session-services requests flowing between an LU and a CP may be field-formatted (RH Format indicator set to 0) or character-coded (RH Format indicator set to 1). Character-coded requests contain RUs consisting of character strings that can be translated into equivalent field-formatted RUs. A translation protocol is provided by the CP. The format of character-coded requests and the translation rules that apply to them are implementation-dependent and are not defined in this book.

All nodes contain a CP. The CP in a type-5 (subarea) node is called a system services control point (SSCP). The CP in a peripheral node is called a peripheral node control point (PNCP). When initiating or terminating an LU-LU session, the LU exchanges session services RUs with one CP—the one configured to mediate the initiation or termination of the particular LU-LU session.

When the LU-LU session is between subarea nodes or between a subarea node and a peripheral node, an SSCP mediates the session initiation or termination. In this case, the LU-LU session uses a route in a subarea path-control network. When the LU-LU session is between peripheral nodes, the PNCP in one of the two nodes mediates the session initiation or termination. In this case, the LU-LU session does not use subarea path-control, but instead uses a direct route between the two peripheral nodes.

When initiating and terminating SSCP-mediated sessions, the session-services RUs are exchanged between the LU and the SSCP over an SSCP-LU session. Similarly, for PNCP-mediated sessions, the RUs are exchanged between the LU and the PNCP over a PNCP-LU session within the peripheral node, and no session-services RUs flow outside the node.

Activation and deactivation of a CP-LU session is accomplished by exchanging session-control RUs between the CP and LNS. The CP-LU session is activated prior to initiating any LU-LU sessions for which that CP is the mediator.

LNS informs the CP—either an external SSCP or the internal CP—about the characteristics and current status of the LU during the activation of the CP-LU session. LNS negotiates session parameters with the LNS component in the partner LU during LU-LU session activation.

The LU resources manager (abbreviated RM) in one of the two LUs directs the activation or deactivation of an LU-LU session. Upon completion of the activation or deactivation, LNS in each of the two LUs informs its local RM that the LU-LU session has been activated or deactivated.

LNS is aware of the type of node (peripheral or subarea) in which the LU resides and of the identity of the CP-mediator (PNCP or SSCP) for each LU-LU session. LNS absorbs differences in protocols that result from the type of node in which it resides. This permits other components of the LU to be independent of the node type. LNS also isolates the other components from the CP-mediator for the LU-LU sessions, and thus the routing—subarea path-control or direct—used for the sessions.

## OVERVIEW OF CP-LU SESSION ACTIVATION

The CP directs the LU to activate a CP-LU session by sending it an ACTLU request. The PU in the node receives the ACTLU request, determines which LU is to receive the request, and passes the request to the LNS component in the LU. LNS processes the ACTLU request and activates a CP-LU half-session. LNS's processing of the ACTLU request includes the following:

- Check for error conditions associated with the request, and for conditions that prevent activation of the session.
- Notify path control within the node that a new session is being activated.
- Send an ACTLU response to the CP.
- Create and initialize the half-session process for the LU's side of the CP-LU session.

After the CP-LU session is activated, the LU can initiate LU-LU sessions for which that CP is the mediator.

## OVERVIEW OF CP-LU SESSION DEACTIVATION

The CP directs the LU to deactivate a CP-LU session by sending it a DACTLU(Normal) request (in contrast to a DACTLU(SON) resulting from session outage). The PU in the node receives the DACTLU request, determines which LU is to receive the request, and passes the request to the LNS component in the LU. LNS processes the DACTLU request and deactivates the CP-LU half-session. LNS's processing of the DACTLU request includes the following:

- Check for error conditions associated with the request.
- Send a DACTLU response to the CP.
- Notify path control within the node that the session has been deactivated.
- Reset all LU-LU half-sessions for which this CP was the session-initiation mediator.

- Destroy the half-session process for the LU's side of the CP-LU session.

When the LU receives a DACTLU(SON) request, LNS performs similar processing except that it does not reset any LU-LU half-sessions.

After the CP-LU session is deactivated, the LU cannot initiate LU-LU sessions for which that CP is the mediator.

## OVERVIEW OF LU-LU SESSION INITIATION

RM directs the LU to activate an LU-LU session by sending LNS an ACTIVATE_SESSION record across an protocol boundary. LNS processes the ACTIVATE_SESSION record and initiates an LU-LU half-session. The LNS components in the two LUs activate the LU-LU session by exchanging a BIND request and response. LNS's processing of the ACTIVATE_SESSION record, which constitutes its part of the LU-LU session initiation, includes the following:

- Check for conditions that prevent activation of the session.
- Obtain the identification of the CP that will mediate the LU-LU session initiation.
- Send an INIT-SELF request to the CP. The request directs the CP to mediate the initiation of the LU-LU session.
- Obtain the session parameters and activate the LU-LU session, as follows:
  - If the LU is to be the primary for the session, then receive a CINIT request from the CP, build a BIND request that specifies the desired parameters for the LU-LU session, send the BIND request to the partner (secondary) LU, and receive the BIND response.
  - If the LU is to be the secondary for the session, then receive the BIND request, build a negotiated BIND response that specifies the agreed-to parameters for the LU-LU session, and send the BIND response to the partner (primary) LU.
- Notify path control within the node that a new session is being activated.
- Create and initialize the half-session process for this LU's side of the LU-LU session.
- Notify the CP that a new LU-LU session is activated.
- Notify RM that the requested LU-LU session is active.

The partner LU to the one initiating the LU-LU session is directed to activate the LU-LU session by means of receiving either the CINIT request when it is the primary LU, or the BIND request when it is the secondary LU. Its processing following receipt of the CINIT or BIND request is similar to the processing just outlined. However, instead of replying to RM with a notification that a requested session is activated, LNS informs RM that an LU-LU session has been activated at the direction of the remote LU. After the LU-LU session is activated, the two LUs can allocate the session for conversations between transaction programs.

The parameters used for the LU-LU 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 design of the implementation of the node.
- User 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 request and sent in the BIND request.

## OVERVIEW OF LU-LU SESSION TERMINATION

RM directs the LU to deactivate an LU-LU session by sending LNS a DEACTIVATE_SESSION record across an internal protocol boundary. LNS processes the DEACTIVATE_SESSION record and terminates the LU-LU half-session. The two LUs deactivate the LU-LU session by exchanging an UNBIND request and response. LNS's processing of the DEACTIVATE_SESSION record, which constitutes its part of the LU-LU session termination, includes the following:

- Send an UNBIND request to the partner LU and receive the UNBIND response.
- Notify path control within the node that the session has been deactivated.
- Notify the CP that the LU-LU session has been deactivated.
- Destroy the half-session process for this LU's side of the LU-LU session.

The partner LU to the one terminating the LU-LU session is directed to deactivate the LU-LU session by means of receiving the UNBIND request. Its processing following receipt of the UNBIND request is similar to the processing just outlined. However, after the session has been deactivated, LNS informs RM that an LU-LU session has been deactivated at the direction of the remote LU.

RM may request deactivation of an LU-LU session that is pending activation, that is, a session for which LNS has sent an INIT-SELF request to the CP and has not yet received the CINIT or BIND request for the session. LNS terminates a pending-active session by sending the CP a TERM-SELF request. The TERM-SELF request directs the CP to terminate the pending-active session without completing the initiation. LNS terminates the pending-active session when it sends TERM-SELF, without waiting for the response.

### 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 a reset of one or both of their half-sessions, 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 LNS receives a session outage notification, it notifies RM for each LU-LU session affected by the session outage.

When session outage occurs, RM may direct LNS to reinitiate the sessions. For example, RM requests session reinitiation when the session outage causes the number of active sessions for which the LU is the contention winner to decrease below a minimum number. See "Chapter 3. LU Resources Manager" and "Chapter 5.4. Presentation Services--Control-Operator Verbs" for more details.

## NETWORK CONTEXT FOR SESSION INITIATION AND TERMINATION

Certain terms are used that relate to LU-LU session initiation and termination. The terms are used to identify the roles of the LUs in the context of initiating and terminating LU-LU sessions. The terms are:

- Initiating LU (ILU)
- Terminating LU (TLU)
- Origin LU (OLU)
- Destination LU (DLU)
- Primary LU (PLU)
- Secondary LU (SLU)

The abbreviations in parentheses following the terms appear in the format descriptions of the session-services and session-control RUs given in "Appendix E. Request-Response Unit (RU) Formats" and are also used throughout this chapter.

### ILU AND TLU

ILU and TLU refer to the role of an LU in initiating and terminating a particular LU-LU session. The LU that initiates an LU-LU session is the ILU, and the LU that terminates an LU-LU session is the TLU. The ILU or TLU may be one of the session partners, in which case the LU the CP an INIT-SELF or TERM-SELF request, respectively. The ILU or TLU may, instead, be a third-party LU that is not one of the session partners. Session initiation or termination by a third-party LU applies only to SSCP-mediated sessions. Details of the formats and protocols for third-party LUs are not described.

The ILU or TLU may reside in either a subarea node or peripheral node for SSCP-mediated sessions, except that a third-party ILU or TLU always resides in a subarea node. The ILU or TLU resides in a peripheral node for PNCP-mediated sessions.

### OLU AND DLU

OLU and DLU refer to the role of an LU and its CP during session initiation or termination. An ILU or TLU that is one of the session partners is also the OLU. An LU whose SSCP receives an initiation or termination request from a third-party LU is the OLU. The OLU's session partner is the DLU. The INIT-SELF includes the name of the DLU.

The OLU or DLU may reside in either a subarea node or peripheral node for SSCP-mediated sessions. The OLU or DLU reside in a peripheral node for PNCP-mediated sessions.

### PLU AND SLU

PLU and SLU refer to the role of an LU in providing, respectively, primary or secondary half-session control for an LU-LU session of which it is a partner. The PLU sends the BIND request and receives the BIND response. Correspondingly, the SLU receives the BIND request and sends the BIND response.

The PLU resides in a subarea node for SSCP-mediated sessions, and a peripheral node for PNCP-mediated sessions. The SLU may reside in either a subarea node or peripheral node for SSCP-mediated sessions; it resides in a peripheral node for PNCP-mediated sessions.

## RU PARAMETERS

The following sections define some parameters that are common to many session-services and session-control field-formatted RUs.

### NETWORK NAME

A network name is the name by which an LU is known throughout an individual SNA network. Network names are unique within an individual network.

### FULLY QUALIFIED NETWORK NAME

A fully qualified network name is the name by which an LU is known throughout an interconnected SNA network. An interconnected network comprises one or more individual networks. A fully qualified network name consists of a network identifier and a network LU name. Fully qualified network names are unique throughout an interconnected network.

### UNINTERPRETED NAME

An uninterpreted name is any name by which an LU and its CP know another LU for the purpose of initiating an LU-LU session. It can be used by an ILU to identify a DLU. An uninterpreted name requires interpretation (or transformation) by the CP in order to yield the network name. An uninterpreted name may be the same as a network name.

### USER REQUEST CORRELATION

A user request correlation (URC) field denotes a variable-length byte string consisting of a Length field and the URC itself. It is assigned by the end user for placement in an INIT-SELF or TERM-SELF request. Its usage allows subsequent requests involving the ILU or TLU to be associated with the INIT-SELF or TERM-SELF request. The associated requests either contain a field specifically defined for this purpose or use a session key (discussed under "Session Key and Session Key Content").

### MODE NAME

The CP has information about the LU that aids in the construction of the BIND image (carried in CINIT). The CP derives the BIND image contents from the mode name. The LU supplies the mode name in INIT-SELF requests.

In addition to the BIND image, the CP uses the mode name to select a class of service for the LU-LU session. As an example, some sessions may require service with a fast response time (implying, for example, high-speed links, shortest distance, and high transmission priority), while others may require large bandwidth or more secure paths. Different mode names can be defined in order to select the different classes of service.

Using the mode name, a transaction program is able to select for a conversation the session characteristics it desires. Then, when allocating the conversation to a session, LRM supplies the mode name for the session in its session-activation request to LNS.

The derivation of the BIND image and the class of service from the mode name is implementation-dependent and installation-specified.

### SESSION KEY AND SESSION KEY CONTENT

There are various ways of denoting which LU-LU session a request is referring to; this may be, for example, by name pair, address pair, or by the URC. The session key and session key content permit requests that refer to sessions to do so in one or more ways. The session key content contains the particular fields denoted by the session key. The format description, in "Appendix E. Request-Response Unit (RU) Formats", of a request specifying a session key and session key content also specifies the keys permitted (or required) with that request.

When the session key content contains a name pair or an address pair, it is an ordered pair. The order is (PLU,SLU) unless otherwise specified by the session key definition. Exceptions exist for requests whose formats use the LU designations, OLU and DLU. For these formats the session key content order is (OLU,DLU) and other related fields specify which is PLU and which is SLU.

### 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 on is determined by implementation design.

## 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 on is determined by the user at the node installation. Installation-specified values can be established during system configuration of a node, or later during its operation. The method for establishing values of installation-specified parameters is implementation-dependent.

## SESSION-SERVICES RU'S

This section describes the session-services requests and extended responses that LNS sends and receives. These RUs belong to the FM-data category of network-services RUs.

Preceding the individual descriptions is a list of the RUs, grouped according to their use. Listed with each RU is the number of the page on which the description of the RU begins. In addition, Figure 4-2 on page 4-8 shows the RH formats for the session-services requests and responses that LNS sends and receives.

Each RU description includes the RU flow and a discussion of the function and use of the RU. Refer to "Appendix E. Request-Response Unit (RU) Formats" for specifications of the RU formats.

Session-services RUs pertaining to LU-LU session initiation are:

| RU | Page |
|---|---|
| INITIATE-SELF (INIT-SELF) | 4-9 |
| CONTROL INITIATE (CINIT) | 4-9 |
| RSP(CINIT) | 4-10 |
| SESSION STARTED (SESSST) | 4-11 |
| BIND FAILURE (BINDF) | 4-11 |

RUs pertaining to session termination are:

| RU | Page |
|---|---|
| TERMINATE-SELF (TERM-SELF) | 4-11 |
| CONTROL TERMINATE (CTERM) | 4-12 |
| CLEAN UP SESSION (CLEANUP) | 4-12 |
| SESSION ENDED (SESSEND) | 4-13 |
| UNBIND FAILURE (UNBINDF) | 4-13 |

The following RU pertains to reporting the status of the session initiation or termination, or of the LU:

| RU | Page |
|---|---|
| NOTIFY | 4-14 |

| Header Indicators | Session-Services RU ---> | INIT-SELF<br>CINIT<br>TERM-SELF<br>CTERM<br>CLEANUP<br>NOTIFY | SESSST<br>SESSEND<br>BINDF<br>UNBINDF | |
|---|---|---|---|---|
| TH | EFI | Normal | Normal | A |
| RH Byte 0 Bit 0 | RRI | RQ | RQ | Request |
| Bits 1-2 | RU_CTGY | FMD | FMD | |
| Bit 3 | reserved | 0 | 0 | |
| Bit 4 | FI | 1 | 1 | |
| Bit 5 | SDI | *SD | *SD | |
| Bit 6 | BCI | BC | BC | |
| Bit 7 | ECI | EC | EC | |
| RH Byte 1 Bit 0 | DR1I | DR1 | ¬DR1 | |
| Bit 1 | reserved | 0 | 0 | |
| Bit 2 | DR2I | ¬DR2 | ¬DR2 | |
| Bit 3 | ERI | ¬ER | ¬ER | |
| Bits 4-5 | reserved | 00 | 00 | |
| Bit 6 | QRI | ¬QR | ¬QR | |
| Bit 7 | PI | ¬PAC | ¬PAC | |
| RH Byte 2 Bit 0 | BBI | ¬BB | ¬BB | |
| Bit 1 | EBI | ¬EB | ¬EB | |
| Bit 2 | CDI | ¬CD | ¬CD | |
| Bit 3 | reserved | 0 | 0 | |
| Bit 4 | CSI | Code 0 | Code 0 | |
| Bit 5 | EDI | ¬ED | ¬ED | |
| Bit 6 | PDI | ¬PD | ¬PD | |
| Bit 7 | CEBI | ¬CEB | ¬CEB | V |
| TH | EFI | Normal | | A |
| RH Byte 0 Bit 0 | RRI | RSP | | Response |
| Bits 1-2 | RU_CTGY | FMD | | |
| Bit 3 | reserved | 0 | | |
| Bit 4 | FI | 1 | | |
| Bit 5 | SDI | *SD | | |
| Bit 6 | BCI | BC | | |
| Bit 7 | ECI | EC | | |
| RH Byte 1 Bit 0 | DR1I | DR1 | | |
| Bit 1 | reserved | 0 | | |
| Bit 2 | DR2I | ¬DR2 | | |
| Bit 3 | RTI | ±RSP | | |
| Bits 4-5 | reserved | 00 | | |
| Bit 6 | QRI | ¬QR | | |
| Bit 7 | PI | ¬PAC | | |
| RH Byte 2 Bits 0-7 | reserved | 00000000 | | V |

Notes:
1. *XX means either XX or ¬XX.
2. See "Appendix D. RH Formats" for complete RH descriptions.
3. The TH formats are not described in this book.
4. SESSST, SESSEND, BINDF, and UNBINDF are sent with no-response indicated.

Figure 4-2. Session-Services RH Formats

## INITIATE-SELF (INIT-SELF)

Flow: From ILU to CP (Normal)

INIT-SELF requests that the CP assist in the initiation of a session between the LU sending the request (the ILU, which also becomes the OLU) and the LU named in the request (the DLU). The INIT-SELF indicates definite-response requested.

For SSCP-mediated sessions, the ILU may be located in either a subarea node or peripheral node. For PNCP-mediated sessions, the ILU is located in a peripheral node.

The INIT-SELF request contains, among other parameters, the uninterpreted name of the DLU with which the session is to be initiated, the mode name for the session, and a URC for the initiation request.

The DLU may be unavailable for activation of an LU-LU session. This occurs when the DLU is not currently able to comply with the PLU|SLU specification, or when it is at its session limit. At CP-LU session activation time, the LU informs the CP of its availability by means of control vector X'0C' carried in its positive response to ACTLU. Subsequently, during the active CP-LU session, the LU reports changes in its availability (such as changes in its PLU|SLU capability or its session limit) by sending NOTIFY(Vector Key X'0C') to the CP.

The CP queues the initiation request if the INIT-SELF indicates that queuing is permitted, the CP supports queuing, and the DLU is currently unavailable. The CP queues the INIT-SELF request until the DLU becomes available.

The URC that the ILU sends in INIT-SELF is returned in the CINIT. When the PLU sends the INIT-SELF, the URC received in CINIT allows the PLU to correlate the CINIT with the INIT-SELF. When the SLU sends the INIT-SELF, the PLU copies the URC from CINIT into BIND to allow the SLU to correlate the BIND with the INIT-SELF.

The CP returns a positive response to the INIT-SELF request after it verifies the resource availability and mode name, and, if applicable, it queues the initiation request. If an initiation failure occurs after a positive response has been returned, the CP notifies the ILU by means of NOTIFY(Vector Key X'03'). The NOTIFY includes the URC from the INIT-SELF in order to allow the ILU to correlate the NOTIFY with the INIT-SELF.

## CONTROL INITIATE (CINIT)

Flow: From CP to PLU (Normal)

CINIT requests that the LU receiving the request attempt to activate an LU-LU session with the LU named in the request. The LU receiving CINIT is the PLU for the session. The LU named in the request is the SLU for the session. The CINIT indicates definite-response requested.

For SSCP-mediated sessions, the PLU is located in a subarea node. For PNCP-mediated sessions, the PLU is located in a peripheral node.

The parameters in CINIT include the suggested parameters for the BIND, which represent the <u>BIND image</u>. The BIND image parameters are selected by the CP. Selection is based on optional implementation-dependent and installation-specified parameters for the PLU or SLU to which the respective parameters apply, and on the mode-name parameter in the INIT-SELF associated with the CINIT.

The PLU inspects the Format and URC fields in the BIND image for errors. If Format 0 is not specified, or if the PLU initiated the session and the URC is omitted from the CINIT request or the URC included on the CINIT request does not match the URC that the PLU sent on a previous INIT-SELF, then the PLU rejects the CINIT by returning a negative response to CINIT.

The PLU also inspects the mode name carried in a control vector on CINIT. If the mode name does not match the one on the corresponding INIT-SELF that the PLU sent, the PLU rejects the CINIT. Similarly, if the SLU or a third-party LU initiated the session and the mode name does not match one that is system-defined for the SLU, the PLU rejects the CINIT.

If the PLU finds no errors with the CINIT, it sends back a positive CINIT response.

The PLU copies some CINIT parameters into the BIND without modification. These are the Staging indicators, the PLU Name field, and the SLU Name field. The URC field is copied into BIND when the SLU sends the INIT-SELF, and the Cryptography Options field is copied into BIND when both LUs support session-level mandatory cryptography. The mode name from the control vector on CINIT is copied into the Mode Name Structured Data Subfield of the User Data field of BIND.

CINIT may include a User Data field in the BIND image of CINIT. If it does, the PLU discards the user data and does not copy the field into the BIND.

When the SLU sends an INIT-SELF, the PLU Name field in the associated CINIT carries the uninterpreted name of the PLU sent in the INIT-SELF; otherwise, it carries the network name of the PLU.

If the INIT-SELF designated the PLU as the DLU, the PLU copies the URC from the BIND image of CINIT into the BIND. Otherwise, the PLU omits the URC from BIND.

If both the PLU and SLU support session-level mandatory cryptography and it is specified for the mode name sent in INIT-SELF, the associated CINIT carries the session cryptography key enciphered twice—once under the PLU master cryptography key and once under the SLU master cryptography key; the former is used at the PLU, while the latter (carried in the BIND image) is passed by the PLU in BIND for use at the SLU. The session cryptography key is a pseudo-random number. See "Chapter 6.2. Transmission Control" for details on cryptography.

The PLU may modify other parameters supplied in the CINIT before sending them in the BIND. Specifically, the PLU may change the primary and secondary TCs' pacing window sizes and the maximum RU sizes specified in CINIT. More details are given in the description of BIND in this chapter.

The changing of any of the pacing parameters and maximum RU sizes on one session may affect the performance characteristics of that session and of concurrently active sessions that share network resources with it.

See the description of BIND in this chapter for additional rules on TS Profile and TS Usage modifications that are allowed.

The route to be used for the LU-LU session is identified in the CINIT. For SSCP-mediated LU-LU sessions, CINIT carries a control vector that contains the mode name, class of service, and virtual route list associated with the subarea path-control route to be used for the session. The PLU and SLU addresses to be used for the session flows are network addresses, carried in either a session key field or a control vector.

For PNCP-mediated sessions, CINIT carries a control vector that contains the mode name for the LU-LU session. PNCP-mediated sessions use a direct route between peripheral nodes and do not use network addresses for the session flows. Therefore, in place of network addresses, CINIT carries an identifier of the adjacent link station associated with the node in which the SLU resides.
CINIT can result from an INIT-SELF from one of the session partners (the PLU or SLU). Alternatively, CINIT can result from an initiation request from a third-party LU. The formats and protocols for session initiation by a third-party LU are not described.

## RSP(CINIT)

Flow: From PLU to CP (Normal)

A positive response to CINIT informs the CP that the PLU accepts the CINIT request and will attempt to activate the requested LU-LU session with the LU named in the CINIT request.

For SSCP-mediated sessions, the PLU is located in a subarea node. For PNCP-mediated sessions, the PLU is located in a peripheral node.

The CINIT request includes a BIND image that the PLU uses in building the BIND request for the LU-LU session. The PLU inspects the Format and URC fields in the BIND image for errors. If Format 0 is not specified, or if the PLU initiated the session (PLU = ILU) and the URC is omitted from the CINIT request or the URC included on the CINIT request does not match the URC that the PLU sent on a previous INIT-SELF, then the PLU rejects the CINIT by returning a negative response to CINIT. Otherwise, the PLU accepts the CINIT by returning a positive CINIT response.

The CINIT response has an extended format that differs from the CINIT request. The CINIT response specifies control vector X'FE' as the only parameter of the response.

Control vector X'FE' contains a list of control vector keys, received on the CINIT request, that the PLU does not recognize. If the SLU receives on the CINIT request a control vector it does not recognize, the SLU includes control vector X'FE' on the CINIT response. Otherwise, the SLU omits control vector X'FE' from the CINIT response.

## SESSION STARTED (SESSST)

Flow: From LU to CP (Normal)

SESSST notifies the CP that an LU-LU session has been successfully activated. Both the PLU and SLU send SESSST to their CP. The SESSST indicates no-response requested.

For SSCP-mediated sessions, the PLU is located in a subarea node and sends SESSST to its SSCP. The SLU is located in either a subarea node or peripheral node and sends SESSST to either its SSCP or PNCP, respectively.

For PNCP-mediated sessions, the PLU and SLU are located in peripheral nodes. Each LU sends SESSST to its PNCP.

SESSST sent to the SSCP identifies the LU-LU session that is started. A session key containing the network addresses of the PLU and SLU is used for this purpose.

The SESSST sent to the SSCP may carry additional information by means of control vectors. Further details are not defined.

SESSST sent to the PNCP identifies the adjacent link station for the node in which the partner LU is located. This SESSST has an internal format different from the SESSST sent to the SSCP.

## BIND FAILURE (BINDF)

Flow: from PLU to CP (Normal)

BINDF informs the CP that an attempt to activate an LU-LU session has failed, for the reason indicated in the BINDF. The BINDF indicates no-response requested.

For SSCP-mediated sessions, the PLU is located in a subarea node and sends BINDF to its SSCP. The BINDF identifies the LU-LU session that failed to be activated. A session key containing the network addresses of the PLU and SLU is used for this purpose. Sense data identifying the error and a reason code for the error are included in the BINDF.

For PNCP-mediated sessions, the PLU is located in a peripheral node. The PLU does not send BINDF to its PNCP.

## TERMINATE-SELF (TERM-SELF)

Flow: From TLU to CP (Normal)

TERM-SELF requests that the CP assist in the termination of a session between the sender of the request (the TLU) and LU named in the request (the DLU). The TERM-SELF indicates definite-response requested.

For SSCP-mediated sessions, the TLU may be located in either a subarea node or peripheral node. For PNCP-mediated sessions, the TLU is located in a peripheral node.

The session to be terminated can be either queued or pending-active, from the TLU's perspective. Therefore, only the ILU can send TERM-SELF, because only the ILU is aware of sessions that are queued or pending-active. Note that from the SSCP's perspective, the session may be active, as well as pending-active or queued.

The LU does not send TERM-SELF to terminate an active LU-LU session. Instead, the LU sends UNBIND to the partner LU.

The TERM-SELF request identifies the session to be terminated by means of the URC session key. The URC session key is the same as the one sent in the INIT-SELF that initiated the session. The URC field (distinct from the URC session key) can be specified in TERM-SELF to correlate a TERM-SELF with a subsequent NOTIFY.

The TERM-SELF request designates the type of termination to be performed, which is always Forced. TERM-SELF(Forced) requests the CP to assist in terminating the pending-active or queued session immediately and unconditionally.

The CP returns a positive response once it has validated the TERM-SELF request. For SSCP-mediated sessions, if an error occurs after a positive response has been sent, the SSCP notifies the TLU by means of NOTIFY(Vector Key X'03'). The NOTIFY includes the URC from the TERM-SELF so that the TLU can correlate the NOTIFY with the TERM-SELF.

For SSCP-mediated sessions, if the SSCP's perspective of the session is that it is active, the SSCP sends CTERM(Forced) to the PLU. See the description of CTERM in this chapter for more information.

## CONTROL TERMINATE (CTERM)

Flow: From CP to PLU (Normal)

CTERM requests that the PLU attempt to deactivate an LU-LU session. The CTERM indicates definite-response requested.

CTERM is used to terminate an SSCP-mediated LU-LU session. The SSCP sends CTERM to the PLU, located in a subarea node, as a result of receiving a terminate request from an LU. The LU that sent the terminate request to the SSCP can be the PLU or SLU for the session, or a third-party LU. See the description of TERM-SELF for more details about when the PLU or SLU sends a termination request to the SSCP. Details of session termination resulting from a request sent by a third-party LU are not defined.

The CTERM identifies the session to be terminated by means of a session key containing the network addresses of the PLU and SLU. The CTERM also specifies the type of termination requested and the reason for termination.

The type of termination specified in CTERM is either Orderly or Forced. CTERM(Orderly) allows the PLU to delay deactivating the session. In particular, the PLU does not deactivate the session while a conversation is using the session. CTERM(Forced) requires an unconditional attempt to deactivate the session, even if a conversation is using the session.

CTERM(Forced) is the only type of termination that can result from a TERM-SELF sent to the SSCP by the PLU or SLU. Both CTERM(Orderly) and CTERM(Forced) can result from a termination request sent by a third-party LU.

CTERM is not used for PNCP-mediated sessions. When a PNCP-mediated session is to be terminated, the LU sends UNBIND to the partner LU. The PNCP does not send a terminate request to the LU.

## CLEAN UP SESSION (CLEANUP)

Flow: CP to LU (Normal)

CLEANUP informs the LU that it is to deactivate the LU-LU session immediately, even if a conversation is using the session. The CLEANUP indicates definite response requested.

CLEANUP is used to deactivate an SSCP-mediated LU-LU session. The SSCP sends CLEANUP to the LU, located in subarea node, as a result of receiving a terminate request from an LU. The LU that sent the terminate request to the SSCP can be the PLU or SLU for the session, or a third-party LU. See the description of TERM-SELF for more details about when the PLU or SLU sends a termination request to the SSCP. Details of session termination resulting from a request sent by a third-party LU are not defined.

CLEANUP can result from a TERM-SELF(Forced). This occurs when the SSCP is unable to process the TERM-SELF(Forced) and therefore must promote the forced termination to a cleanup termination.

The CLEANUP identifies the session to be terminated by means of a session key containing the network addresses of the PLU and SLU. The CLEANUP also specifies the the reason for termination.

In response to receiving CLEANUP, the LU sends UNBIND with the Type set to Cleanup. UNBIND(Cleanup) deactivates the sender's half-session, without waiting for a response to the UNBIND.

CLEANUP is received only by an LU in a subarea node. When the SLU of an SSCP-mediated session is located in a peripheral node, the SLU receives a DACTLU followed by ACTLU in place of CLEANUP.

CLEANUP is not used for PNCP-mediated sessions. When a PNCP-mediated session is to be terminated, the LU sends UNBIND to the partner LU. The PNCP does not send a terminate request to the LU.

## SESSION ENDED (SESSEND)

Flow: from LU to CP (Normal)

SESSEND notifies the CP that an LU-LU session has been successfully deactivated. Both the PLU and SLU send SESSEND to their CP. The SESSEND indicates no-response requested.

For SSCP-mediated sessions, the PLU is located in a subarea node and sends SESSEND to its SSCP. The SLU is located in either a subarea node or peripheral node and sends SESSEND to either its SSCP or PNCP, respectively.

For PNCP-mediated sessions, the PLU and SLU are located in peripheral nodes. Each LU sends SESSEND to its PNCP.

SESSEND sent to the SSCP identifies the LU-LU session that is ended. A session key containing the network addresses of the PLU and SLU is used for this purpose. SESSEND also indicates the cause of the deactivation.

SESSEND sent to the PNCP identifies the adjacent link station for the node in which the partner LU is located. This SESSEND has an internal format different from the SESSEND sent to the SSCP.

## UNBIND FAILURE (UNBINDF)

Flow: From PLU to CP (Normal)

UNBINDF informs the CP that an attempt of deactivate a session has failed, for the reason indicated in the UNBINDF. The UNBINDF indicates no-response requested.

For SSCP-mediated sessions, the PLU is located in a subarea node and sends UNBINDF to its SSCP. The UNBINDF identifies the LU-LU session that failed to be deactivated.

A session key containing the network addresses of the PLU and SLU is used for this purpose. Sense data identifying the error and a reason code for the error are included in the UNBINDF.

For PNCP-mediated sessions, the PLU is located in a peripheral node. The PLU does not send UNBINDF to its PNCP.

Flow: From CP to LU and from LU to CP (Normal)

NOTIFY is used to send information from a CP to an LU, or from an LU to a CP. The NOTIFY indicates definite-response requested.

NOTIFY is used for SSCP-mediated sessions, only. An LU in either a subarea node or peripheral node can send or receive NOTIFY pertaining to an SSCP-mediated session. NOTIFY is not used for PNCP-mediated sessions.

NOTIFY carries information in the form of a (vector key, vector data) pair:

- Vector key X'03'—ILU|TLU notification: Sent in NOTIFY from the SSCP to the ILU or TLU in order to notify the LU of a session-initiation or -termination failure after a positive response has been returned to the INIT-SELF or TERM-SELF. For a session-initiation failure, the NOTIFY indicates a setup procedure error; for a session-termination failure, it indicates a takedown procedure error. The NOTIFY also includes the reason for the error and the sense data identifying the error.

  The URC from the INIT-SELF or TERM-SELF is carried in the NOTIFY to allow the ILU or TLU to correlate the NOTIFY with the INIT-SELF or TERM-SELF.

- Vector key X'0C'—LU-LU session-services capabilities: Sent in NOTIFY from an LU to an SSCP to convey changes in the LU's current LU-LU session-services capabilities.

  The parameters of the LU-LU session-services capabilities include the LU's session count and limit, its capability to act as a PLU or SLU, and its capability to support parallel sessions. Its capability to act as a PLU or SLU is indicated as:

  - Enabled—sessions can be started
  - Disabled—sessions can be queued but not started
  - Inhibited—sessions can be neither queued nor started

  Whenever an event occurs during an active CP-LU session causing a change in an LU's session-services capabilities, the LU sends NOTIFY to the SSCP to convey its new session-services capabilities. (At CP-LU session activation time, the LU conveys its session-services capabilities to the SSCP by means of control vector X'0C' carried in the LU's response to ACTLU.)

  The session-services-capabilities parameters determine whether a DLU is available for initiation of an LU-LU session. In terms of these parameters, a DLU is available for session initiation when all of the following conditions are met:

  - The DLU's session count is less than its session limit.
  - It is enabled for PLU or SLU capability, as requested in the INIT-SELF request.
  - It supports parallel sessions with the OLU (this condition applies when one session between the OLU and DLU is already active).

  Otherwise, the DLU is unavailable for session initiation.

  The parameters specifying the LU's session count and limit, and its PLU or SLU capability, are used to determine whether to queue an INIT-SELF request, as follows:

  - When an INIT-SELF designates a DLU that is currently unavailable—because its session count equals its session limit or because its PLU or SLU capability as requested in the INIT-SELF is disabled—and the INIT-SELF specifies initiate/queue and the SSCP supports queuing of INIT-SELF, the INIT-SELF is queued.
  - When an INIT-SELF designates a DLU that is currently unavailable and either (1) the INIT-SELF specifies initiate only, (2) the SSCP does not support queuing of INIT-SELF, (3) the DLU's PLU or SLU capability as requested in the INIT-SELF is inhibited, or (4) the DLU does not support parallel sessions and a session between the OLU and DLU is already active, the INIT-SELF request is rejected (a negative response is returned).
  - When the DLU sends a NOTIFY indicating it has become available, the SSCP dequeues INIT-SELF requests (up to the session limit) for that DLU, resuming the session-initiation process.
  - When INIT-SELF designates a DLU that is available (and other necessary conditions are met), the session is initiated.

The defined (vector key, vector data) pairs are specified in Appendix E.

## SESSION-CONTROL RU'S

This section describes the session-control requests and extended responses that LNS sends and receives. Preceding the individual descriptions is a list of the RUs, grouped according to their use. Listed with each RU is the number of the page on which the description of the RU begins. In addition, Figure 4-3 on page 4-16 shows the RH formats for the session-control requests and responses that LNS sends and receives.

Each RU description includes the RU flow and a discussion of the function and use of the RU. Refer to "Appendix E. Request-Response Unit (RU) Formats" for specifications of the RU formats.

Session control RUs pertaining to CP-LU session activation and deactivation are:

| RU | Page |
|---|---|
| ACTIVATE LOGICAL UNIT (ACTLU) | 4-17 |
| RSP(ACTLU) | 4-17 |
| DEACTIVATE LOGICAL UNIT (DACTLU) | 4-19 |

Session-control RUs pertaining to LU-LU session activation and deactivation are:

| RU | Page |
|---|---|
| BIND SESSION (BIND) | 4-19 |
| RSP(BIND) | 4-25 |
| UNBIND SESSION (UNBIND) | 4-28 |

| Session Control RU ——> Header Indicators | | ACTLU DACTLU BIND UNBIND | |
|---|---|---|---|
| TH | EFI | Expedited | A |
| RH Byte 0 Bit 0 | RRI | RQ | |
| Bits 1-2 | RU_CTGY | SC | |
| Bit 3 | reserved | 0 | |
| Bit 4 | FI | 1 | |
| Bit 5 | SDI | *SD | |
| Bit 6 | BCI | BC | |
| Bit 7 | ECI | EC | |
| RH Byte 1 Bit 0 | DR1I | DR1 | Request |
| Bit 1 | reserved | 0 | |
| Bit 2 | DR2I | ¬DR2 | |
| Bit 3 | ERI | ¬ER | |
| Bits 4-5 | reserved | 00 | |
| Bit 6 | QRI | ¬QR | |
| Bit 7 | PI | ¬PAC | |
| RH Byte 2 Bit 0 | BBI | ¬BB | |
| Bit 1 | EBI | ¬EB | |
| Bit 2 | CDI | ¬CD | |
| Bits 3-6 | reserved | 0000 | |
| Bit 7 | CEB | ¬CEB | V |
| TH | EFI | Expedited | A |
| RH Byte 0 Bit 0 | RRI | RSP | |
| Bits 1-2 | RU_CTGY | SC | |
| Bit 3 | reserved | 0 | |
| Bit 4 | FI | 1 | |
| Bit 5 | SDI | *SD | |
| Bit 6 | BCI | BC | |
| Bit 7 | ECI | EC | Response |
| RH Byte 1 Bit 0 | DR1I | DR1 | |
| Bit 1 | reserved | 0 | |
| Bit 2 | DR2I | ¬DR2 | |
| Bit 3 | RTI | ±RSP | |
| Bits 4-5 | reserved | 00 | |
| Bit 6 | QRI | ¬QR | |
| Bit 7 | PI | *PAC | |
| RH Byte 2 Bits 0-7 | reserved | 00000000 | V |

Notes:
1. *XX means either XX or ¬XX.
2. See Appendix D for complete RH descriptions.
3. The TH formats are not described in this book.

Figure 4-3. Session-Control RH Formats

## ACTIVATE LOGICAL UNIT (ACTLU)

Flow: From CP to LU (Expedited)

ACTLU requests that the LU activate a CP-LU session between itself and the CP that sent the ACTLU request. The CP assumes the role of primary NAU, while the LU assumes the role of secondary. The ACTLU indicates definite-response requested.

The LU sends back either a positive or negative response, depending on the parameters of the ACTLU request. In addition, if the format of the ACTLU request is in error, or the LU already has a CP-LU session with the CP that sent the request, it sends back a negative response.

A description of the parameters in the ACTLU request follows.

Type: This specifies the type of CP-LU session activation requested. Type ERP (error recovery procedure) is always specified; no other type is defined for the ACTLU request. The LU sends back a negative response if the ACTLU request specifies a type other than ERP.

Type ERP is used to activate the CP-LU session without affecting any active LU-LU sessions. The type of session activation that the LU actually performs is indicated on the response. The LU may perform either ERP or Cold session activation.

FM Profile: This specifies the FM profile to be used for the CP-LU session. The FM profile indicated in the ACTLU request can be 0 or 6. The FM profile actually used for the CP-LU session is indicated on the response.

For LUs located in subarea nodes, FM profile 6 is always used. If the ACTLU request indicates FM profile 0, the LU negotiates it to 6.

For LUs located in peripheral nodes, either FM profile 0 or 6 may be used. If the LU implementation supports FM Profile 6, then 6 is used; if the ACTLU request indicates 0, the LU negotiates it to 6. If the LU implementation does not support FM profile 6, then FM profile 0 is used; if the ACTLU request indicates 6, the LU rejects the request and sends back a negative response.

TS Profile: This specifies the TS profile to be used for the CP-LU session. TS profile 1 is the only one defined. The LU sends back a negative response if the ACTLU request specifies a TS profile other than 1.

## RSP(ACTLU)

Flow: From LU to CP (Expedited)

A positive response to ACTLU completes activation of a CP-LU session between the LU and the CP that sent the ACTLU request. The ACTLU response also informs the CP of the CP-LU session capabilities and the LU-LU session services capabilities of the LU.

The positive ACTLU response has an extended format. If the ACTLU request is acceptable to the LU, it sends back a positive ACTLU response that specifies the parameters for the CP-LU session and for the LU's session services capabilities.

A description of the parameters in the ACTLU response follows.

Type: This specifies the type of CP-LU session activation that the LU is performing. The type of session activation may be either Cold or ERP (error recovery procedure).

The LU specifies type Cold when it has no active or pending-active LU-LU sessions. Otherwise, the LU specifies type ERP.

FM Profile: This specifies the FM profile to be used for the CP-LU session. The FM profile may be 0 or 6. For LUs located in subarea nodes, FM profile 6 is always used and indicated in the ACTLU response.

For LUs located in peripheral nodes, either FM profile 0 or 6 may be used. If the LU implementation supports FM Profile 6, then 6 is used and indicated in the response. If the LU implementation does not support FM profile 6 and FM profile 0 is indicated in the ACTLU request, then FM profile 0 is used and indicated in the response.

TS Profile: This specifies the TS profile to be used for the CP-LU session. TS profile 1 is the only one defined.

Control Vector X'00'—CP-LU Session Capabilities: This specifies the LU's capabilities for the CP-LU session with the CP that sent the ACTLU request. The CP-LU session capabilities are installation-specified for each

CP, and may be different for sessions with different CPs. The LU specifies the CP-LU session capabilities by means of parameters on the control vector. Details of the parameters follow.

- <u>Key:</u> This specifies the control vector key, X'00'.

- <u>Maximum RU Size:</u> This specifies the maximum RU size that either half-session may send. This parameter may specify a specific maximum RU size or no maximum RU size.

- <u>Character-Coded Capability:</u> This specifies whether the CP is permitted to send unsolicited character-coded requests to the LU. The capability to receive unsolicited character-coded requests on the CP-LU session is implementation-dependent.

- <u>Field-Formatted Capability:</u> This specifies whether the CP is permitted to send unsolicited field-formatted requests to the LU. The capability to receive unsolicited field-formatted requests on the CP-LU session is implementation-dependent.

<u>Control Vector X'0C'—LU-LU Session-Services Capabilities:</u> This specifies the LU's capabilities for LU-LU sessions for which the CP that sent the ACTLU request will be the mediator. The LU session-services capabilities are installation-specified for each CP, and may be different for sessions with different CPs. The LU specifies the LU session services capabilities by means of parameters on the control vector. Details of the parameters follow.

- <u>Key:</u> This specifies the control vector key, X'0C'.

- <u>Length of Vector Data:</u> This specifies the length of the remainder of the control vector.

- <u>Primary LU Capability:</u> This specifies whether the LU is currently available as a PLU for LU-LU session initiation. The LU's current PLU capability is specified as enabled, disabled, or inhibited.

  - <u>Enabled:</u> This LU can activate sessions for which it is the PLU, provided its LU-LU session count is less then its LU-LU session limit.

  - <u>Disabled:</u> This LU cannot activate sessions for which it is the PLU, but the CP may queue session-initiation requests that specify (1) this LU is the PLU for the session and (2) queue if this LU is currently unable to comply with the PLU/SLU specification.

  - <u>Inhibited:</u> This LU cannot activate sessions for which it is the PLU, and the CP cannot queue session-initiation requests that specify this LU is the PLU for the session.

  LUs located in peripheral nodes are able to activate SSCP-mediated sessions only as SLUs. Therefore, these LUs always specify their PLU capability as inhibited when the CP is an SSCP.

- <u>Secondary LU Capability:</u> This specifies whether the LU is currently available as an SLU for LU-LU session initiation. The LU's current SLU capability is specified as enabled, disabled, or inhibited.

  - <u>Enabled:</u> This LU can activate sessions for which it is the SLU, provided its LU-LU session count is less then its LU-LU session limit.

  - <u>Disabled:</u> This LU cannot activate sessions for which it is the SLU, but the CP may queue session-initiation requests that specify (1) this LU is the SLU for the session and (2) queue if this LU is currently unable to comply with the PLU/SLU specification.

  - <u>Inhibited:</u> This LU cannot activate sessions for which it is the SLU, and the CP cannot queue session-initiation requests that specify this LU is the SLU for the session.

- <u>LU-LU Session Limit:</u> This specifies the LU's current session limit for initiating sessions for which the CP is the mediator. Initiation of sessions for which the CP is not the mediator are not constrained by this session limit. A specification of 0 means the LU has no session limit.

  If the LU is in a peripheral node and the CP is an SSCP, the LU always specifies its session limit as 1.

- <u>LU-LU Session Count:</u> This specifies the LU's current session count of active sessions for which the CP is the mediator. Active sessions for which the CP is not the mediator are not included in this session count.

  If the session limit is not 0 and it is greater than the session count, the LU is available for LU-LU session initiation. If the session limit is not 0 and it equals the session count, the LU is unavailable for LU-LU session initiation and the CP may queue session-initiation requests that specify queue if the session limit is exceeded.

- <u>Parallel-Session Capability:</u> This specifies the LU's capability to support parallel-session protocols for sessions for which the CP is the mediator. If the LU is in a peripheral node and the CP is an SSCP, the LU always specifies parallel sessions are not supported.

- SESSST Capability: This specifies whether the LU, as an SLU, will send a SESSST request to an SSCP. If the LU is in a peripheral node, the LU always specifies it will not send SESSST. If the LU is in a subarea node, the LU always specifies it will send SESSST.

## DEACTIVATE LOGICAL UNIT (DACTLU)

Flow: From CP to LU (Expedited)

DACTLU requests the LU to deactivate the CP-LU session between itself and the CP that sent the DACTLU request. The DACTLU indicates definite-response requested.

The LU sends back either a positive or negative response, depending on the parameters of the DACTLU request. If the format of the DACTLU request is in error, or the type parameter specifies a type other than normal or SON, the LU sends back a negative response. Otherwise, the LU sends back a positive response, even if it has no CP-LU half-session to which it can correlate the DACTLU request.

A description of the parameters in the DACTLU request follows.

Type: This specifies the type of CP-LU session deactivation requested. The type of deactivation is either normal or session outage notification (SON).

- Normal: This specifies that the LU is to deactivate the CP-LU session and reset its half-sessions for all of the LU-LU sessions for which the CP is the mediator.
- Session-Outage Notification: This specifies that the LU is to deactivate the CP-LU session but not reset its LU-LU half-sessions. The DACTLU request includes a specification of the cause of the SON deactivation. See "Session Outage and Session Reinitiation" on page 4-4 for more information on SON. See the definition of the DACTLU request in Appendix E for a list of the cause codes and a description of the causes.

Receipt of DACTLU does not cause the LU to terminate any LU-LU sessions.

## BIND SESSION (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 request carries the PLU's suggested parameters for the session. The specifications of the BIND parameters are based on the PLU's implementation-dependent support, on the installation-specified values currently in effect for the parameters, or on the CINIT request for the session, depending on the particular parameter.

The SLU uses the BIND parameters to help determine whether it will send back a positive or negative response to BIND. In addition, if the format of the BIND request is in error, the SLU sends back a negative response. 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 request after sending a BIND request, 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 in one of the following ways, with one LU being the winner of the race (its BIND is accepted) and the other being the loser (its BIND is rejected):

- If the Fully Qualified PLU Network Name subfield of the user data is omitted from both BINDs, the winner of the race is the LU that sent BIND with the OAF'-DAF' Assignor indicator (ODAI) set to 1. The ODAI is carried in byte 0, bit 6 of the transmission header (TH) of BIND. The ODAI and TH are not further described in this book.

- If the Fully Qualified PLU Network Name subfield is present in only one of the BINDs, the winner is the LU that sent the BIND containing the subfield.

- If the Fully Qualified PLU Network Name subfield is present in both BINDs, the winner is determined by comparing the fully qualified PLU network names in the BINDs. Fully qualified PLU network 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 fully qualified PLU network names is the winner.

  The comparison is made by comparing the two names as EBCDIC character strings, left-justified, and filled on the right with space characters, if necessary. Fully qualified LU network names contain no leading or embedded space characters.

The LU that is the winner of the BIND race sends back a negative response to the BIND it received. The other LU sends back a positive response, unless the BIND is not acceptable for other reasons, such as invalid format.

The BIND request and its response do not have an ERP type, in contrast to other session-activation requests and their responses, such as ACTLU. The distinction between simple activation and resynchronizing reactivation following a failure is made after the session has been activated. In some cases, change-number-of-sessions protocols are used; in others, end-user protocols are invoked.

The SLU does not reject the BIND because of any incompatibility it may have with the BIND parameters. Rather, if the BIND request 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 sent 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 PLU receives the positive BIND response and checks the parameter specifications. If they are acceptable, then these specifications are used for the activated session. Otherwise, the PLU sends UNBIND.

A description of the parameters in the BIND request follows.

Format: This specifies the format of the BIND request. Only one format is defined: Format 0.

Type: This specifies the type of BIND request. The type is always specified as negotiable. The positive response to BIND has the same general format as the BIND request. The negotiable type of BIND request 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 LU-LU 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 request.

TS Profile: This specifies the TS profile to be used for the LU-LU 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 request.

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)—Session Segmenting: This specifies whether the PLU supports receiving segmented BIUs on the session. Support for session-level segmenting of BIUs is implementation-dependent. When both the PLU and SLU specify in the BIND request and BIND response, respectively, that they support session segmenting, then RUs can be segmented on the session; otherwise, segmenting of RUs will not occur. Session segmenting affects the specifications of the maximum-size RUs sent by the PLU and SLU. For more details, see the descriptions of the TS usage parameters, Maximum-Size RU Sent by PLU and Maximum-Size RU Sent by SLU.

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 Response Queuing: This specifies whether the SLU is permitted to queue (hold) the BIND response for an indefinite period. Whether the PLU permits the SLU to queue the BIND response is implementation-dependent. If the PLU does, then the permission 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.

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 a parallel or single session, as indicated by the PS usage parameter, Parallel Session Support, in the BIND request.

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, the contention winner (PLU or SLU) for the session is installation-specified.

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. The specification is taken from the CINIT request for the session. See "Chapter 6.2. Transmission Control" for details on session-level pacing.

TS Usage—Secondary TC's Send Window Size: This specifies whether pacing of normal-flow requests sent by the SLU will occur. If one-stage pacing from the SLU to the PLU is specified for the session, this specification is the same as that for the primary TC's receive window size. Otherwise, this specification is taken from the CINIT request for the session.

TS Usage—Secondary TC's Receive Window Size: This specifies whether pacing of normal-flow requests received by the SLU will occur. The specification is taken from the CINIT request for the session.

TS Usage—Maximum-Size RU Sent by SLU: This specifies the maximum-size RU that the SLU may send to the PLU on the normal flow. The PLU sets this value to the maximum size it allows for received RUs. All implementations permit the specification of a maximum-size RU of 256.

The specification of the maximum-size RU is between a lower bound and an upper bound, which are installation-specified. The lower and upper bounds can range between 8 and 491420, with the lower bound less than or equal to the upper bound. The particular values allowed for the lower bound and upper bound is implementation-dependent, except that minimum lower bound is less than or equal to 256 and the maximum upper bound is greater than or equal to 256.

If session segmenting can occur for the session, the upper bound used is the installation-specified value. Otherwise, the upper bound used is the minimum of (1) path control's maximum-size RU for the PLU node and (2) the installation-specified value. The lower bound used is always the installation-specified value.

Based on the lower and upper bounds and on the CINIT for the session, the PLU sets the value for the maximum-size RU sent by SLU as follows:

- If the value specified in CINIT is between the lower and upper bounds, the PLU copies the value from CINIT into BIND.

- If the value specified in CINIT is less than the lower bound, the PLU sets the value in BIND to the lower bound.

- If the value specified in CINIT is greater than the upper bound, or the value in CINIT is not specified, the PLU sets the value in BIND to the upper bound.

TS Usage—Maximum-Size RU Sent by PLU: This specifies the maximum-size RU that the PLU may send to the SLU on the normal flow. The PLU sets this value to the maximum-size RU it can send. The algorithm used for determining the maximum-size RU sent by the PLU is the same as that used for determining the maximum-size RU sent by the SLU.

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. The specification is taken from the CINIT request for the session.

TS Usage—Primary TC's Send Window Size: This specifies whether pacing of normal-flow requests sent by the PLU will occur. If one-stage pacing from the PLU to the SLU is specified for the session, this specification is the same as that for the secondary TC's receive window size. Otherwise, this specification is taken from the CINIT request for the session.

TS Usage—Primary TC's Receive Window Size: This specifies whether pacing of normal-flow requests received by the PLU will occur. The specification is based on the CINIT request for the session and an installation-specified value, as follows:

- If the CINIT for the session specifies a primary TC's receive window size of 0, this specification is taken from the installation-specified value.

- If CINIT specifies a window size other than 0 and the installation-specified value is 0, this specification is taken from CINIT.

- If CINIT specifies a window size other than 0 and the installation-specified value is also other than 0, this specification is taken from the minimum of the value in CINIT and the installation-specified value.

A window size of 0 means the PLU will receive RUs unpaced.

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—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), 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 controlled.
2. Primary half-session will reinitiate.
3. Secondary half-session will reinitiate.
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 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 LNS informs the resources manager that the current session has been deactivated.

PS Usage—Parallel-Session Support: This specifies whether parallel sessions are supported between the PLU and SLU. Support for parallel sessions is implementation-dependent. If the PLU implementation supports it, 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.

PS Usage—Change-Number-Of-Sessions Support: This specifies whether the PLU and SLU support the change-number-of-sessions (CNOS) protocols, 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, then 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 PLU implementation supports it, the indication of support versus no support is installation-specified for each mode name for the session, and also depends on whether the CINIT for the session specified session-level mandatory cryptography. If both the mode name and the CINIT for the session indicate 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. The PLU name is always specified for an SSCP-mediated session. Whether it is specified for a PNCP-mediated session is implementation-dependent. The PLU omits the PLU name by specifying 0 for the length of the PLU name (applicable only to PNCP-mediated sessions).

The PLU name is taken from the CINIT for the session. When the SLU initiates the session, the PLU name is the uninterpreted name from the INIT-SELF. When the PLU or a third-party LU initiates the session, the PLU name is the network PLU name derived by the CP.

This parameter is not used by LNS. Instead, LNS uses the fully-qualified PLU network 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. LNS makes use of the user data in the BIND request and response, only; LNS does not supply or examine the user data in the session-services RUs.

Figure 4-4 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.

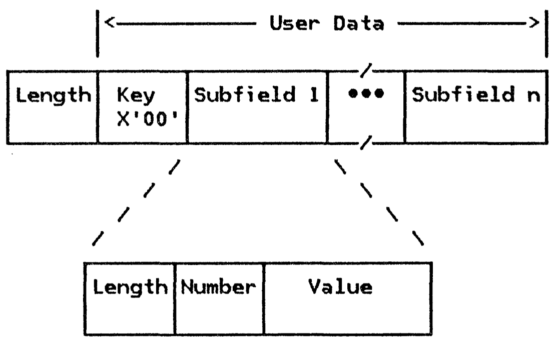


Figure 4-4. Format of User Data

Each subfield includes a length and is identified by a subfield number 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:

| Number | Name |
|---|---|
| X'00' | Unformatted Data |
| X'02' | Mode Name |
| X'03' | Session-Instance ID |
| X'04' | Fully Qualified PLU Network Name |

The PLU may omit one or more subfields; whether it does omit a subfield is implementation-dependent. If it does, the entire subfield, including its length, are omitted.

A T2.1-node implementation that contains a single LU and a single link connection, that does not support parallel sessions and CNOS, and that does not support the synchronization level for Sync point and Backout, 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.

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. For a given session, the PLU uses the mode name from CINIT for the mode name in the Mode Name subfield. 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 multiple sessions between the PLU and SLU. 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 used to differentiate the IDs generated by one LU from those generated by the other LU; this ensures uniqueness of all the IDs used between two LUs. The value of the first byte is either X'F0' or X'00', depending on which LU has the greater fully qualified LU network name. The IDs generated by the LU with the greater fully qualified LU network name have a first byte of X'F0'. The appropriate value (X'F0' or X'00') of the first byte is determined by the SLU and sent in the BIND response.

  The PLU specifies the session-instance ID when parallel sessions and CNOS are supported, when the synchronization level for the session permits Sync point and Backout, or when the session-instance ID is used as part of an implementation-dependent function. Otherwise, the PLU omits the Session-Instance Identifier subfield.

- Subfield X'04'—Fully Qualified PLU Network Name: The fully qualified PLU network name allows the PLU to identify itself to the SLU. The fully qualified PLU network name is installation-specified at both the PLU and SLU.

  An LU resolves BIND-race conditions by comparing the fully qualified PLU network name it sent in the BIND request with the fully qualified PLU network name it received in a BIND request sent by the partner LU. BIND race conditions are discussed in more detail in the first part of this description of the BIND request.

  A T2.1-node implementation that contains a single LU and a single link connection, that does not support parallel sessions and CNOS, and that does not support the synchronization level for Sync point and Backout, may have no fully qualified PLU network name. In this case, the PLU omits the Fully Qualified PLU Network Name subfield from the BIND request.

User Request Correlation: This specifies the user request correlation (URC) value for the session when the SLU initiates the session (SLU = ILU). The SLU uses the URC to correlate the BIND with the INIT-SELF it sent. When the SLU does not initiate the session, the PLU omits the URC from BIND. The PLU omits the URC by specifying 0 for the length of the URC.

Secondary LU Name: This specifies the SLU name used to route the BIND to the intended SLU for the session. For PNCP-mediated sessions, the PU uses the SLU name to route the BIND to the appropriate LU in its node. For SSCP-mediated sessions, the PU uses the destination address in the TH, instead of the SLU name, to route the BIND request to the appropriate LU in its node.

A T2.1-node implementation that contains a single LU and a single link connection, that does not support parallel sessions and CNOS, and that 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.

RSP(BIND)

Flow: From SLU to PLU (Expedited)

A positive response to BIND is sent from an SLU to a PLU to complete activation of a session between the LUs. The positive BIND response has an extended format that is the same as the BIND request.

When the SLU receives a BIND request that is acceptable (for example, there are no format errors and the SLU's session limit is not exceeded), the SLU sends back a positive BIND response 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 request. 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 request, depending on the particular parameter.

The following description of the BIND-response 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 request 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 multilple-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 multilple-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)—Session Segmenting: The SLU specifies whether it supports receiving segmented RUs on the session.

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 Response Queuing: Taken from the BIND request.

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 BIND response. For a parallel session, the specification is taken from the BIND request—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 SLU is to be the installation-specified contention winner; otherwise, the specification is taken from the BIND request.

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: Taken from the BIND request.

TS Usage—Secondary TC's Send Window Size: Taken from the BIND request, as follows: If the BIND request specifies one-stage pacing from the SLU to the PLU, this specification is taken from the primary TC's receive window size; otherwise, this specification is taken directly from the secondary TC's send window size.

TS Usage—Secondary TC's Receive Window Size: This specification is based on the BIND request for the session and an installation-specified value associated with the mode name, as follows:

- If the BIND request 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 and the installation-specified value is 0, this specification is taken directly from BIND.
- If BIND specifies a window size other than 0 and the installation-specified value is also 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 SLU specifies a value between a lower bound and an upper bound, as follows:

- If the value specified in the BIND request is between the lower and upper bounds, the value in the BIND response is taken from the BIND request.
- If the value specified in BIND is less than the lower bound, the SLU sets the value in the BIND response to the lower bound.
- If the value specified in BIND is greater than the upper bound, the SLU sets the value in the BIND response to the upper bound.

TS Usage—Maximum-Size RU Sent by PLU: The SLU specifies a value between a lower bound and an upper bound, as described above for the maximum-size RU sent by the SLU.

TS Usage—Staging for Primary CPMGR to Secondary CPMGR: Taken from the BIND request.

TS Usage—Primary TC's Send Window Size: Taken from the BIND request, as follows: If the BIND request specifies one-stage pacing from the PLU to the SLU, this specification is taken from the seconcary TC's receive window size; otherwise, this specification is taken directly from the secondary TC's send window size.

TS Usage—Primary TC's Receive Window Size: Taken from the BIND request.

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—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 request 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 request 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 request for the session.

The matrix in Figure 4-5 shows how the SLU derives the specification for the BIND response. The rows of the matrix give the installation-specified responsibility and the columns give the responsibility specified in the BIND request. The cells of the matrix give the responsibility that the SLU specifies for the BIND response.

Rows indicate installation-specified responsibility.

Columns indicate responsibility received in BIND request.

■ Cells indicate responsibility sent in BIND response.

| | Oper-ator | Pri-mary | Sec-ondary | Either |
|---|---|---|---|---|
| Operator | Oper-ator | Oper-ator | Oper-ator | Oper-ator |
| Primary | Oper-ator | Pri-mary | Either | Pri-mary |
| Secondary | Oper-ator | Either | Sec-ondary | Sec-ondary |
| Either | Oper-ator | Pri-mary | Sec-ondary | Either |

Figure 4-5. Reinitiation Responsibility

<u>PS Usage—Parallel-Session Support:</u> 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 request 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 request specifies parallel sessions are not supported, the SLU specifies parallel sessions are not supported.

<u>PS Usage—Change-Number-Of-Sessions Support:</u> 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 request 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 request specifies CNOS is not supported, the SLU specifies CNOS is not supported.

<u>Cryptography Options:</u> Taken from the BIND request.

<u>Primary LU Name:</u> Taken from the BIND request.

<u>User Data:</u> The SLU specifies further parameters for the session, by means of the User Data structured subfields. If the SLU receives a BIND request containing a subfield it does not recognize, it ignores the subfield and does not send it in the BIND response.

The User Data subfields that the SLU sends in the BIND response are:

| Number | Name |
|---|---|
| X'00' | Unformatted Data |
| X'02' | Mode Name |
| X'03' | Session-Instance ID |
| X'05' | Fully-Qualified SLU Network Name |

The SLU may omit one or more subfields; whether it does omit a subfield is implementation-dependent. If it does, the entire subfield, including its length, is omitted.

A T2.1-node implementation that contains a single LU and a single link connection, that does not support parallel sessions and CNOS, and that does not support the synchronization level for Sync point and Backout, may omit all User Data subfields.

Details of each subfield follow.

- <u>Subfield X'00'—Unformatted Data:</u> This subfield carries installation-specified data. Support for this subfield is implementation-dependent.
- <u>Subfield X'02'—Mode Name:</u> Taken from the BIND request.
- <u>Subfield X'03'—Session Instance Identifier:</u> Taken from the BIND request, 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 fully qualified LU network name is greater than its own. Otherwise, it sets the first byte to X'00'.
- <u>Subfield X'05'—Fully Qualified SLU Network Name:</u> The fully qualified SLU network name allows the SLU to confirm its identify to the PLU. The fully qualified SLU network name is installation-specified at both the SLU and PLU.

  All T2.1-node products can receive a BIND request with the fully qualified PLU network name subfield omitted. If the SLU receives such a BIND request, it uses a unique default fully qualified PLU network name in order to locally identify the PLU.

A T2.1-node implementation that contains a single LU and a single link connection, that does not support parallel sessions and CNOS, and that does not support the synchronization level for Sync point and Backout, may have no fully qualified SLU network name. In this case, the SLU omits the Fully Qualified SLU Network Name subfield from the BIND response.

User Request Correlation Field: Taken from the BIND request.

Secondary LU Name: Taken from the BIND request.

## UNBIND SESSION (UNBIND)

Flow: From LU to LU (Expedited)

UNBIND requests the partner LU to deactivate the LU-LU session. The UNBIND indicates definite-response requested.

The LU can send an UNBIND request as a result of an action at the LU (one that its CP does not initiate), or as a result of receiving a CTERM or CLEANUP request from its CP. Sending UNBIND as a result of local action is the normal case for terminating SSCP-mediated sessions and the only case for terminating PNCP-mediated sessions.

Sending UNBIND as a result of receiving a CTERM or CLEANUP request occurs when an LU other than one of the session partners requests termination of the LU-LU session, or when one of the session partners sends its SSCP a TERM-SELF request to terminate a pending-active or queued session and the SSCP for the PLU has already sent CINIT to the PLU.

The LU receiving the UNBIND request can send back a positive or negative response to the UNBIND. If the response is positive, both LUs send their respective CPs a SESSEND request. If the response is negative, the session was SSCP-mediated, and the PLU sent the UNBIND request, the PLU sends the SSCP an UNBINDF request.

The LU sends back a negative response if the format of the UNBIND request is in error. Otherwise, the LU sends back a positive response, even if it has no LU-LU half-session to which it can correlate the UNBIND request.

A description of the parameter in the UNBIND request follows.

Type: This specifies the type of LU-LU 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.

The other types of session deactivation are associated with error conditions. Some of these types of session deactivation are caused by session outage notification (SON). See "Session Outage and Session Reinitiation" on page 4-4 for more information about SON.

One of the other types indicates a format or protocol error. When this type is specified, sense data is also included in the UNBIND request. The sense data identifies the reason for the format or protocol error.

## MAINTENANCE-SERVICES RU'S

This section describes the maintenance-services requests that LNS sends and receives. These RUs belong to the FM-data category of network-services RUs.

Preceding the individual descriptions is a list of the RUs. Listed with each RU is the number of the page on which the description of the RU begins. In addition, Figure 4-6 on page 4-30 shows the RH formats for the maintenance-services requests and responses that LNS sends and receives.

Each RU description includes the RU flow and a discussion of the function and use of the RU. Refer to Appendix E for specifications of the RU formats.

The maintenance-services RUs listed below permit an LU to send test data on a CP-LU session and receive a copy of the data from the CP.

| RU | Page |
|---|---|
| ECHO TEST (ECHOTEST) | 4-31 |
| REQUEST ECHO TEST (REQECHO) | 4-31 |

| Header Indicators | MS RU ——> | REQECHO ECHOTEST | |
|---|---|---|---|
| TH | EFI | NORMAL | A |
| RH Byte 0 BIT 0 | RRI | RQ | |
| BITS 1-2 | RU_CTGY | FMD | |
| BIT 3 | reserved | 0 | |
| BIT 4 | FI | 1 | |
| BIT 5 | SDI | *SD | |
| BIT 6 | BCI | BC | |
| BIT 7 | ECI | EC | |
| RH Byte 1 BIT 0 | DR1I | DR1 | Request |
| BIT 1 | reserved | 0 | |
| BIT 2 | DR2I | ¬DR2 | |
| BIT 3 | ERI | ¬ER | |
| BITS 4-5 | reserved | 00 | |
| BIT 6 | QRI | ¬QR | |
| BIT 7 | PI | *PAC | |
| RH Byte 2 BIT 0 | BBI | ¬BB | |
| BIT 1 | EBI | ¬EB | |
| BIT 2 | CDI | ¬CD | |
| BITS 3-6 | reserved | 0000 | |
| BIT 7 | CEB | ¬CEB | V |
| TH | EFI | NORMAL | A |
| RH Byte 0 BIT 0 | RRI | RSP | |
| BITS 1-2 | RU_CTGY | FMD | |
| BIT 3 | reserved | 0 | |
| BIT 4 | FI | 1 | |
| BIT 5 | SDI | *SD | |
| BIT 6 | BCI | BC | |
| BIT 7 | ECI | EC | Response |
| RH Byte 1 BIT 0 | DR1I | DR1 | |
| BIT 1 | reserved | 0 | |
| BIT 2 | DR2I | ¬DR2 | |
| BIT 3 | RTI | ±RSP | |
| BITS 4-5 | reserved | 00 | |
| BIT 6 | QRI | ¬QR | |
| BIT 7 | PI | *PAC | |
| RH Byte 2 BITS 0-7 | reserved | 00000000 | V |

Notes:
1. *XX means either XX or ¬XX.
2. See Appendix D for complete RH descriptions.
3. The TH formats are not described in this book.

Figure 4-6. Maintenance Services RU Formats

## ECHO TEST (ECHOTEST)

**Flow:** From CP to LU (Expedited)

ECHOTEST carries test data to the LU; the test data is the same as that carried in the corresponding REQECHO request that the LU sent. The ECHOTEST indicates definite-response requested.

The number of ECHOTESTs that the CP sends back to the LU is specified by the repetition factor in the REQECHO request. The LU can prematurely terminate the CP's sending of ECHOTESTs by returning a negative response. Support for ECHOTEST is implementation-dependent.

## REQUEST ECHO TEST (REQECHO)

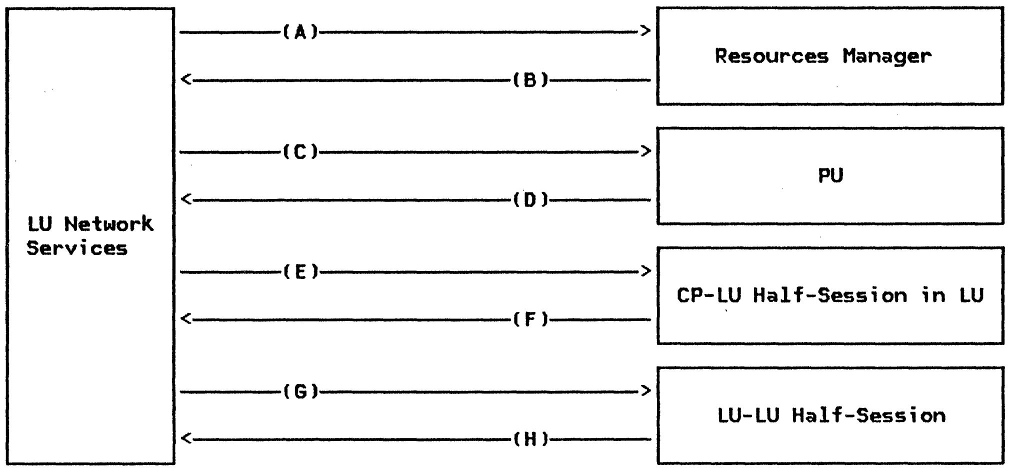
**Flow:** From LU to CP (Expedited)

REQECHO requests that the CP return in an ECHOTEST request the specified test data. The REQECHO indicates definite-response requested.

The repetition factor in the REQECHO request specifies the number of times the CP is to send back ECHOTEST requests, each carrying the same test data as carried in the REQECHO request. Support for REQECHO is implementation-dependent.

## LNS PROTOCOL BOUNDARIES

This section shows the protocol boundaries that LNS has with other components of the LU and with the PU. LNS interacts with other LU components and the PU by sending and receiving records at its protocol boundaries. Figure 4-7 on page 4-33 shows the protocol boundaries and lists the record names associated with these protocol boundaries. The FAPL procedures and finite-state machines (FSMs) of this chapter describe LNS's protocols for sending and receiving these records. See "Appendix A. Node Data Structures" for a definition of the formats of these records.



<u>Records that LNS sends:</u>

(A) ACTIVATE_SESSION_RSP
SESSION_ACTIVATED
SESSION_DEACTIVATED
CTERM_DEACTIVATE_SESSION

(C) BIND_RQ_SEND_RECORD
BIND_RSP_SEND_RECORD
UNBIND_RQ_SEND_RECORD
UNBIND_RSP_SEND_RECORD
ACTLU_RSP_SEND_RECORD
DACTLU_RSP_SEND_RECORD
PC_CONNECT
HIERARCHICAL_RESET_RSP
PC_HS_CONNECT
PC_HS_DISCONNECT

(E) INIT_HS
HS_SEND_RECORD (contains following RUs)
- RQCPID[1]
- INIT_SELF[2]
- TERM_SELF[2]
- NOTIFY_0C[3]
- REQECHO[3]
- ±RSP(CINIT)[2]
- ±RSP(CTERM)[3]
- ±RSP(CLEANUP)[3]
- ±RSP(NOTIFY 03)[3]
- ±RSP(ECHOTEST)[3]
- SESSST[3]
- SESSSTI[4]
- SESSEND[3]
- SESSENDI[4]
- BINDF[3]
- UNBINDF[3]

(G) INIT_HS

<u>Records that LNS receives:</u>

(B) ACTIVATE_SESSION
DEACTIVATE_SESSION

(D) BIND_RQ_RCV_RECORD
BIND_RSP_RCV_RECORD
UNBIND_RQ_RCV_RECORD
UNBIND_RSP_RCV_RECORD
ACTLU_RQ_RCV_RECORD
DACTLU_RQ_RCV_RECORD
PC_CONNECT_RSP
HIERARCHICAL_RESET
SESSION_ROUTE_INOP

(F) INIT_HS_RSP
HS_RCV_RECORD (contains following RUs)
- ±RSP(RQCPID)[1]
- ±RSP(INIT_SELF)[2]
- ±RSP(TERM_SELF)[2]
- ±RSP(NOTIFY 0C)[3]
- ±RSP(REQECHO)[3]
- CINIT[2]
- CTERM[3]
- CLEANUP[3]
- NOTIFY_03[3]
- ECHOTEST[3]

(H) INIT_HS_RSP
ABORT_HS

<u>Notes:</u>

[1] Sent to or received from the PNCP-LU half-session.
[2] Applies to both SSCP- and PNCP-mediated sessions.
[3] Applies only to SSCP-mediated sessions.
[4] Internal form of SESSST and SESSEND, sent to the PNCP-LU half-session.

Figure 4-7. Records Exchanged Between LNS and Other Components

This section shows examples of sequence flows that can occur between LNS and other components of the LU and other nodes. These flows, which are shown in the following pages, illustrate some examples of CP-LU session activation and deactivation, and LU-LU session initiation and termination.

Flows for an LU in a peripheral node are shown in Figure 4-8 on page 4-35 through Figure 4-16 on page 4-40. Flows for an LU in a subarea node are shown in Figure 4-17 on page 4-41 through Figure 4-24 on page 4-45. The names shown on the flows represent the records listed in "LNS Protocol Boundaries" on page 4-32.

The subject LU in the illustrations is referred to as the local LU. Components of the local LU, with which LNS interacts, are shown. Except for the PNCP-LU half-session, the components of the PNCP are not shown in detail. The PNCP-LU half-session is shown simply for clarity of the PNCP-LU session flows within the peripheral node.

The following legend applies to these figures:

| | |
|---|---|
| o————>o | intercomponent flow |
| o——o——>o | intercomponent flow with intermediate-component processing |
| LNS | LU network services |
| LU | logical unit |
| LU-LU HS | LU-LU half-session |
| PC | path control |
| PNCP | peripheral node control point |
| PNCP-LU HS | PNCP-LU half-session |
| PU | physical unit |
| RM | resources manager |
| SSCP | system services control point |
| SSCP-LU HS | SSCP-LU half-session |

**FLOWS FOR A PERIPHERAL LU**

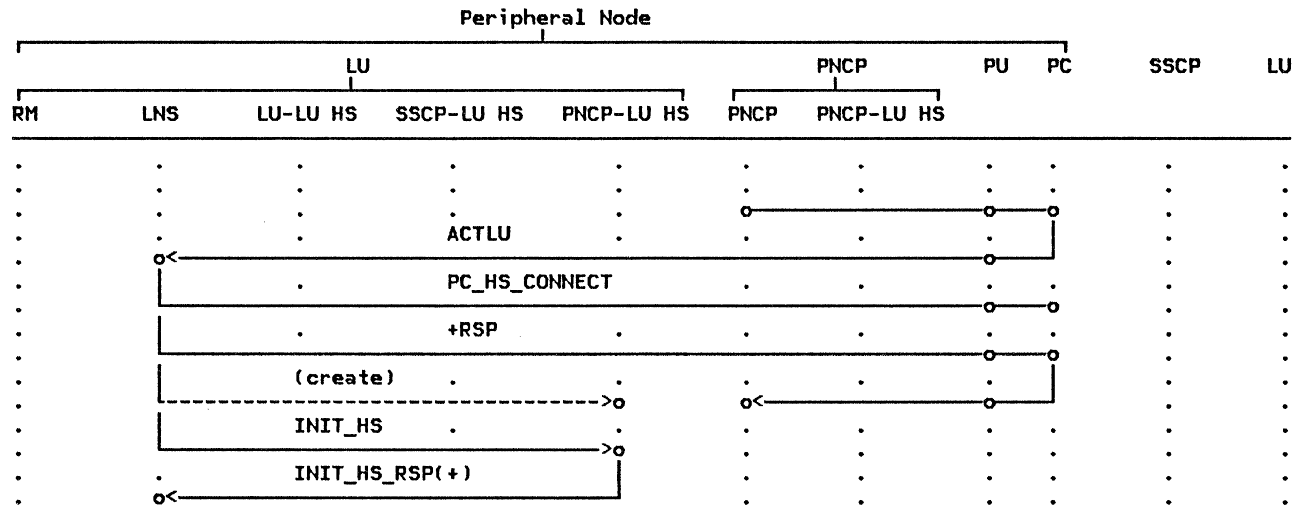


Figure 4-8. PNCP-LU Session Activation

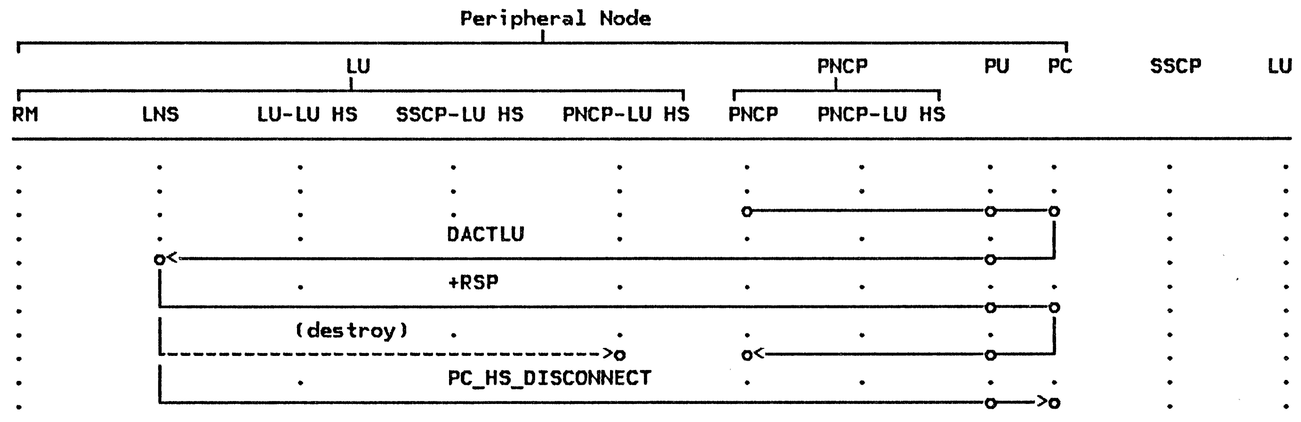


Figure 4-9. PNCP-LU Session Deactivation

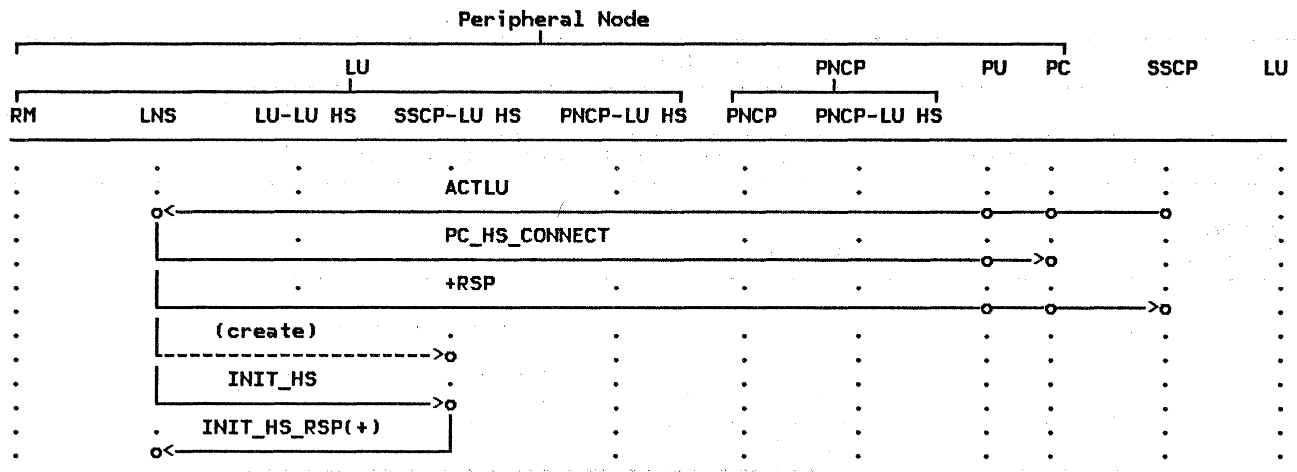


Figure 4-10. SSCP-LU Session Activation to an LU in a Peripheral Node

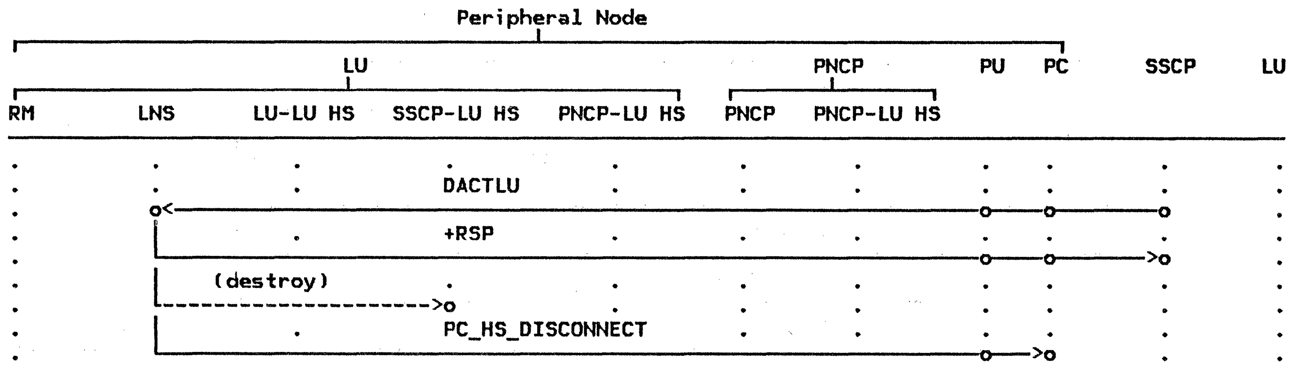
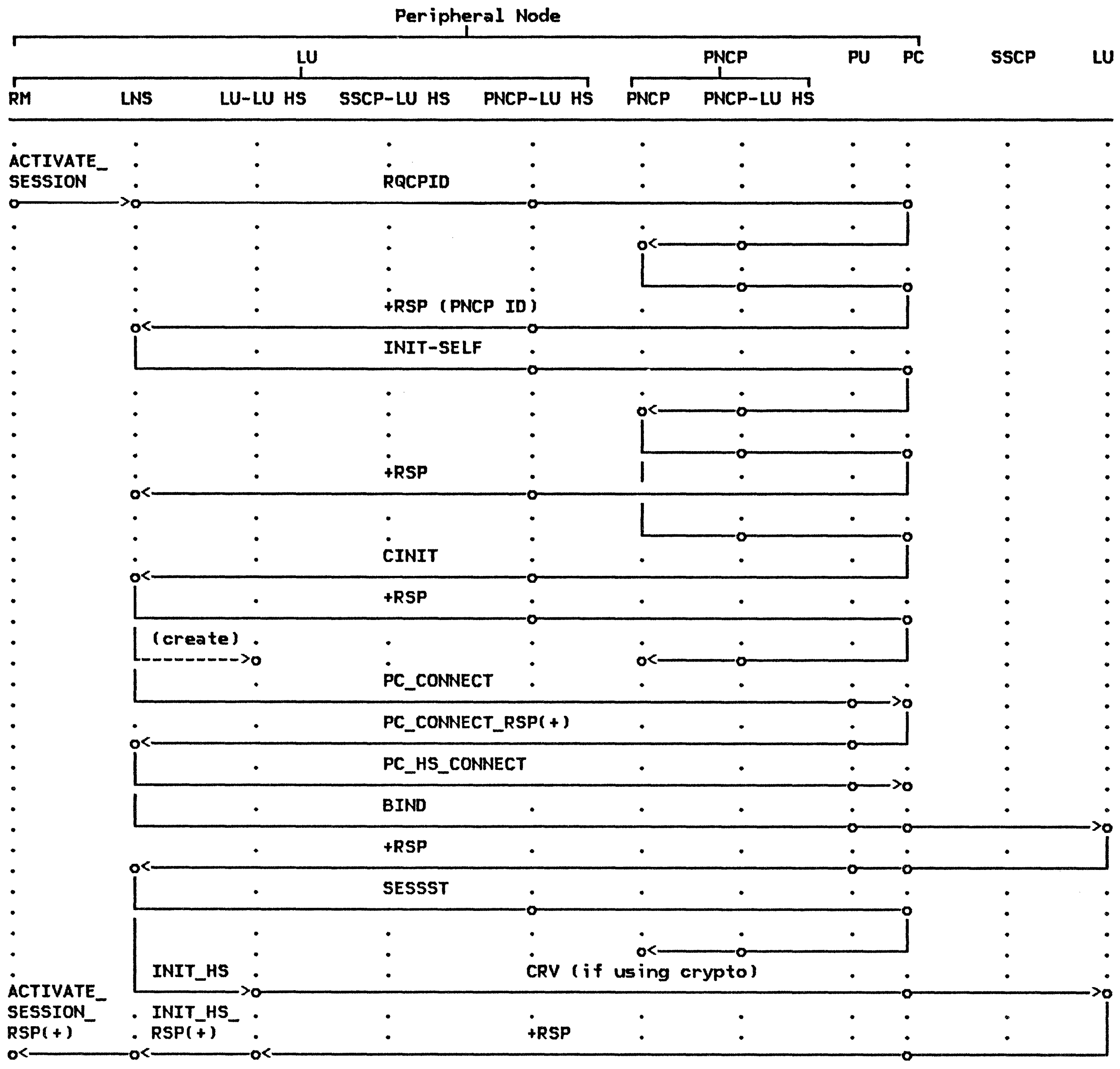
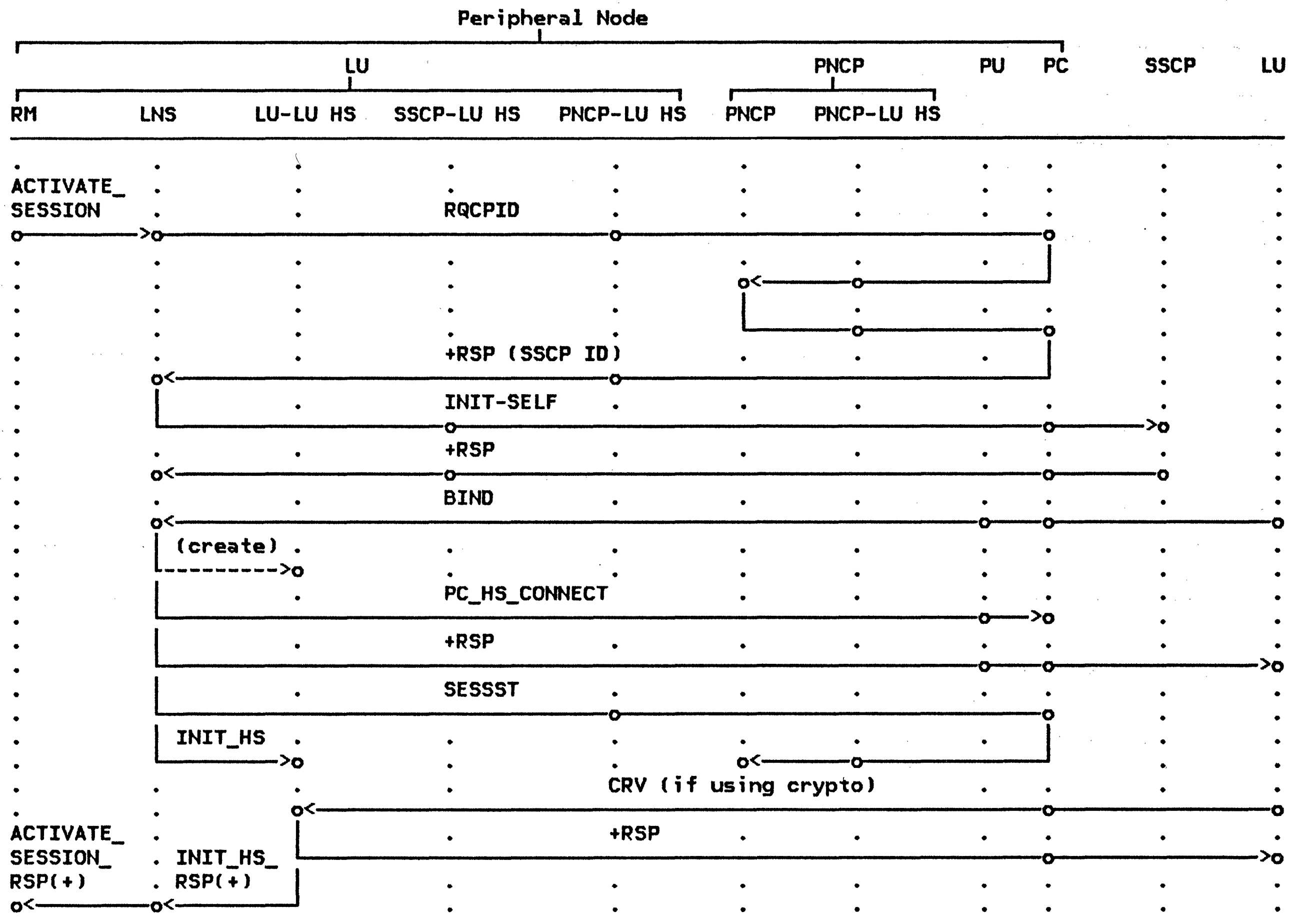


Figure 4-11. SSCP-LU Session Deactivation to an LU in a Peripheral Node



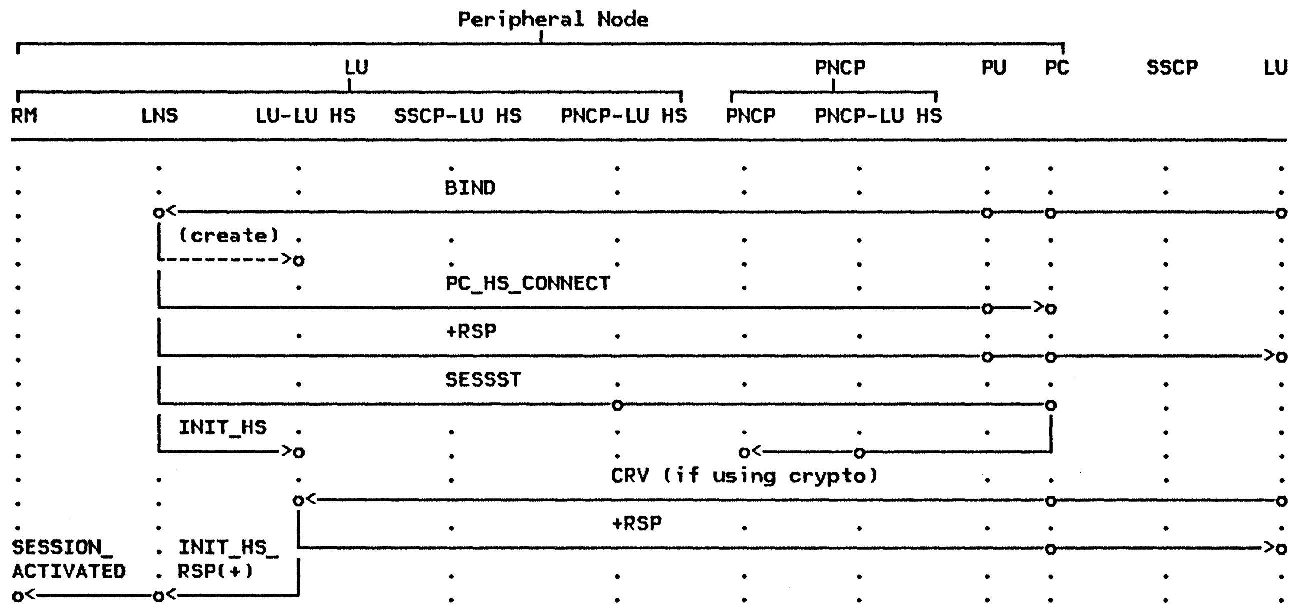
Note: This figure applies only to PNCP-mediated sessions.

Figure 4-12. LU-LU Session Initiation by Local PLU in a Peripheral Node



Note: This figure applies only to SSCP-mediated sessions.

Figure 4-13. LU-LU Session Initiation by Local SLU in a Peripheral Node



Note: This figure applies to both PNCP- and SSCP-mediated sessions.

Figure 4-14. LU-LU Session Initiation by Remote LU to Local LU in a Peripheral Node

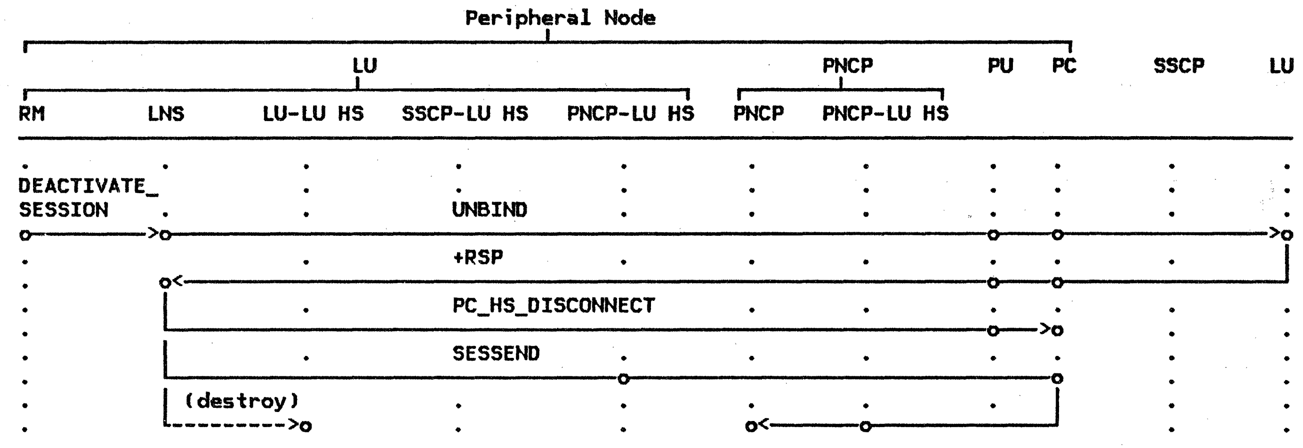


Figure 4-15. LU-LU Session Termination by Local LU in a Peripheral Node

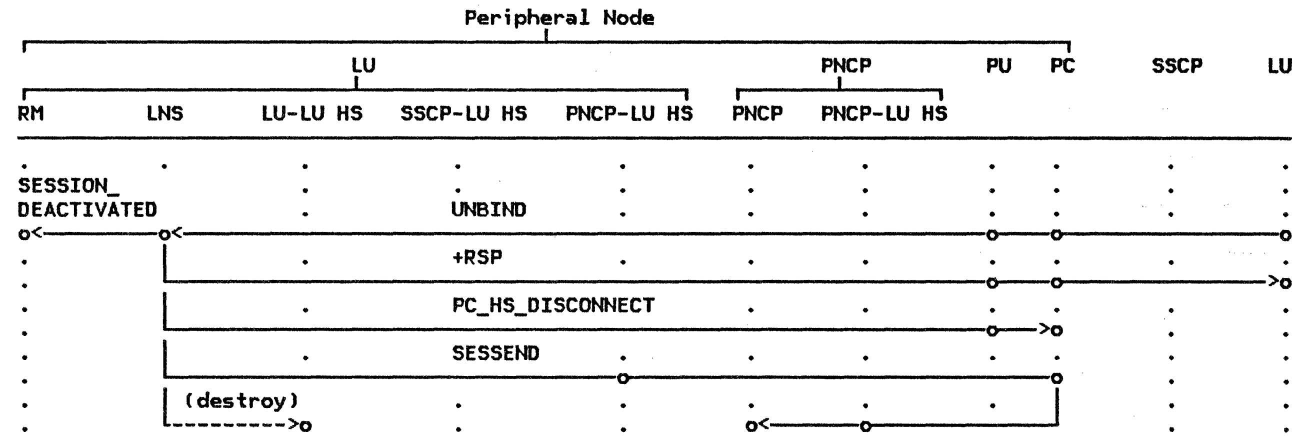


Figure 4-16. LU-LU Session Termination by Remote LU to Local LU in a Peripheral Node

**FLOWS FOR A SUBAREA LU**

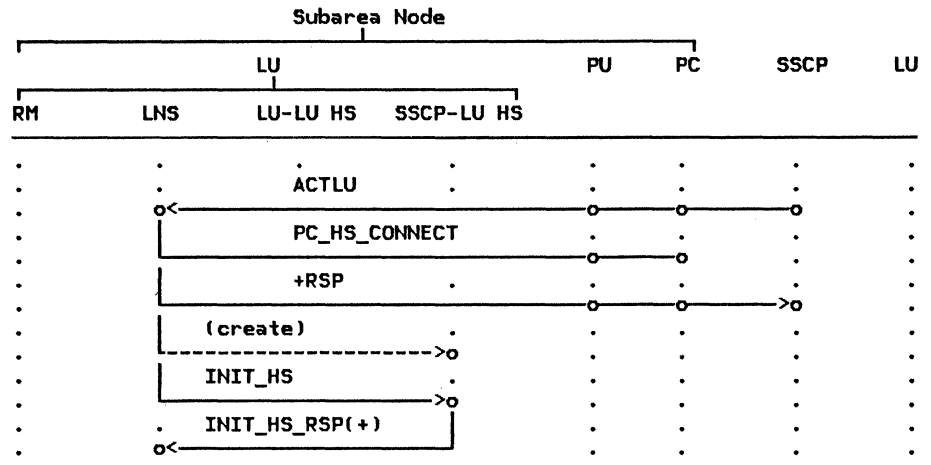


Figure 4-17. SSCP-LU Session Activation to an LU in a Subarea Node

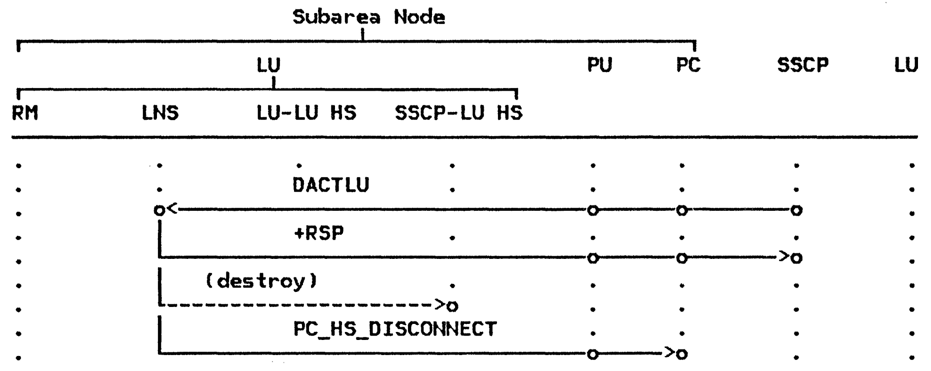


Figure 4-18. SSCP-LU Session Deactivation to an LU in a Subarea Node

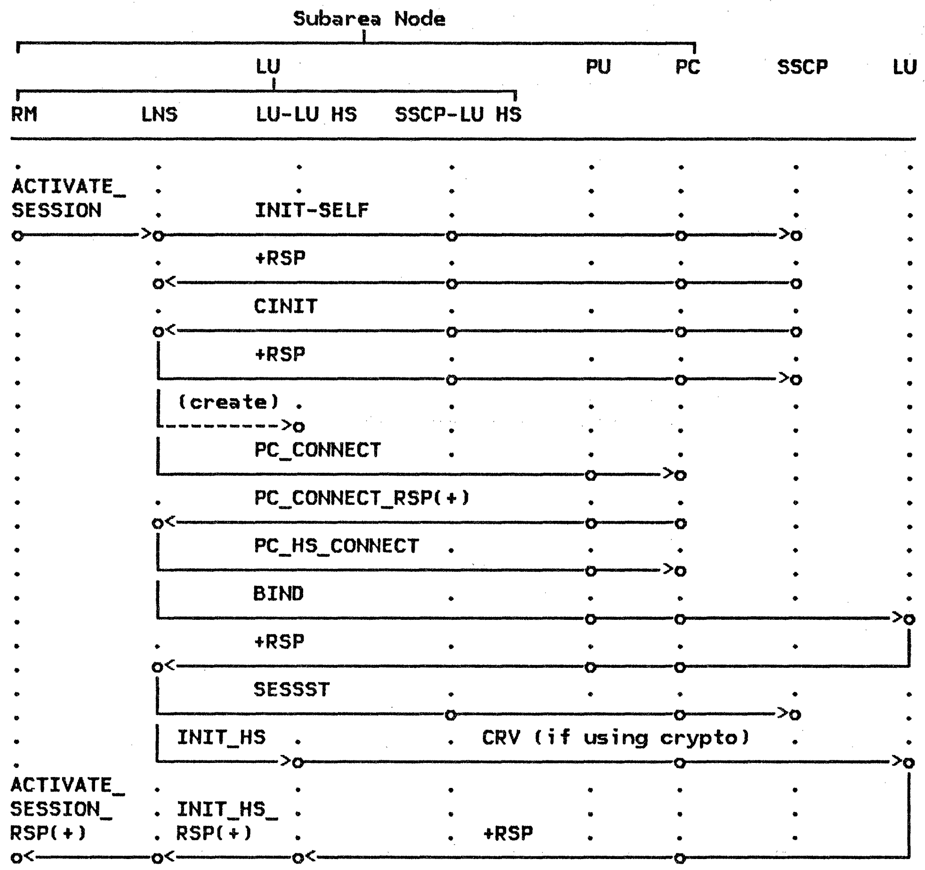


Figure 4-19. LU-LU Session Initiation by Local PLU in a Subarea Node

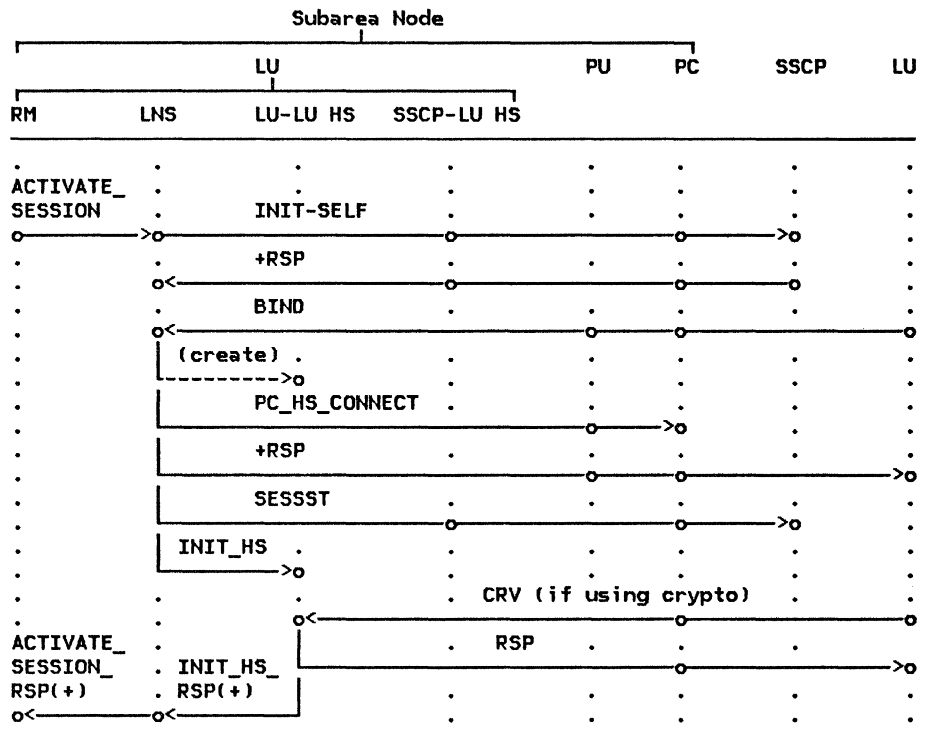


Figure 4-20. LU-LU Session Initiation by Local SLU in a Subarea Node

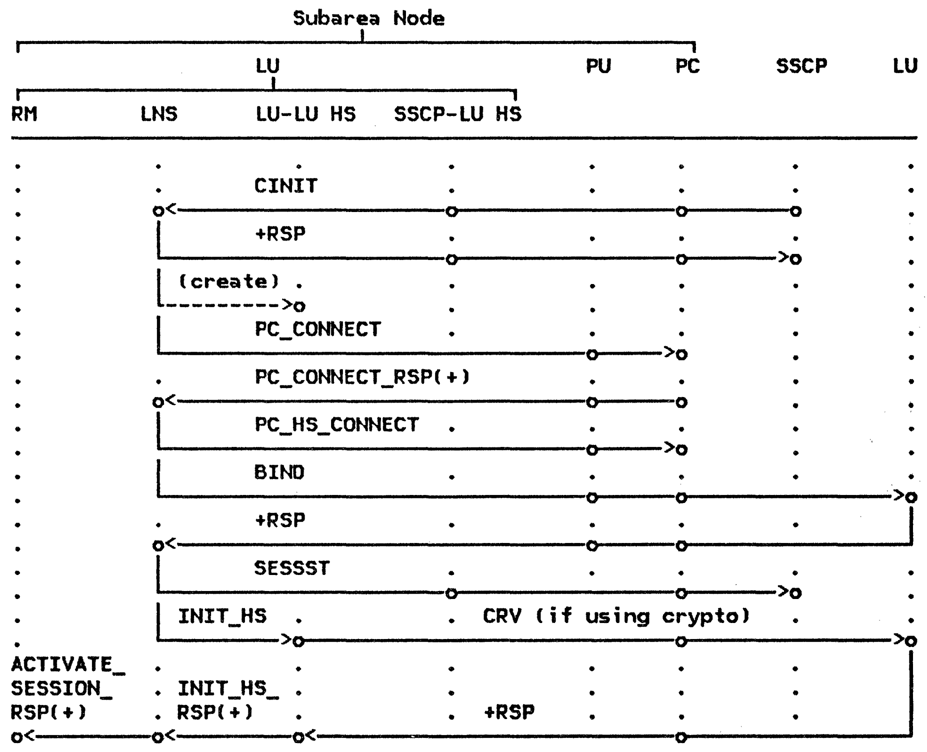


Figure 4-21. LU-LU Session Initiation by Remote PLU to Local SLU in a Subarea Node

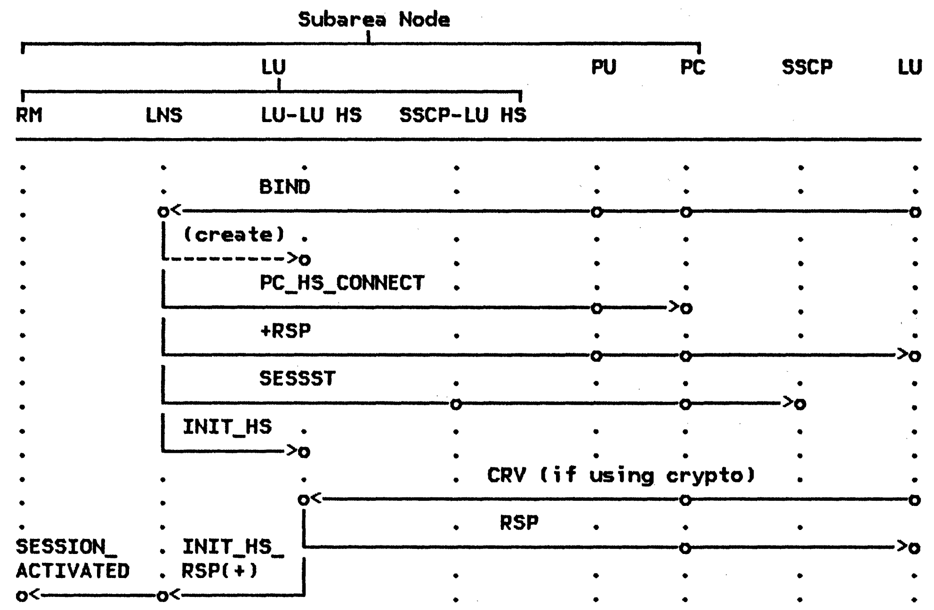


Figure 4-22. LU-LU Session Initiation by Remote PLU to Local SLU in a Subarea Node

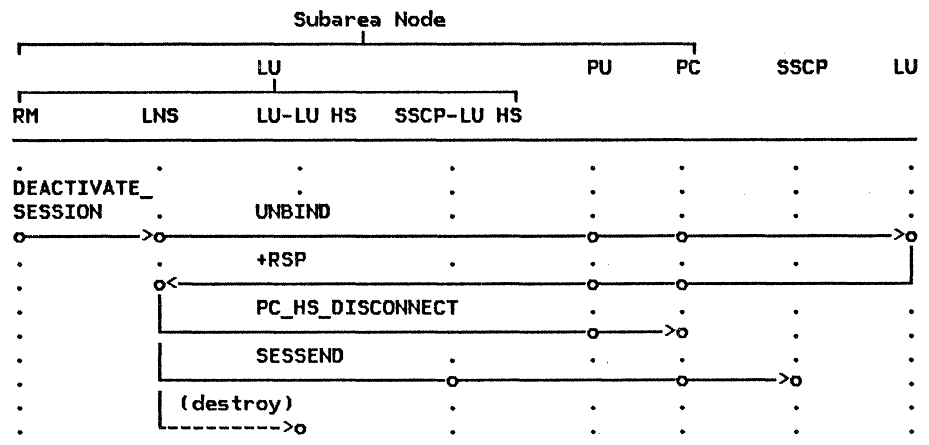


Figure 4-23. LU-LU Session Termination by Local LU

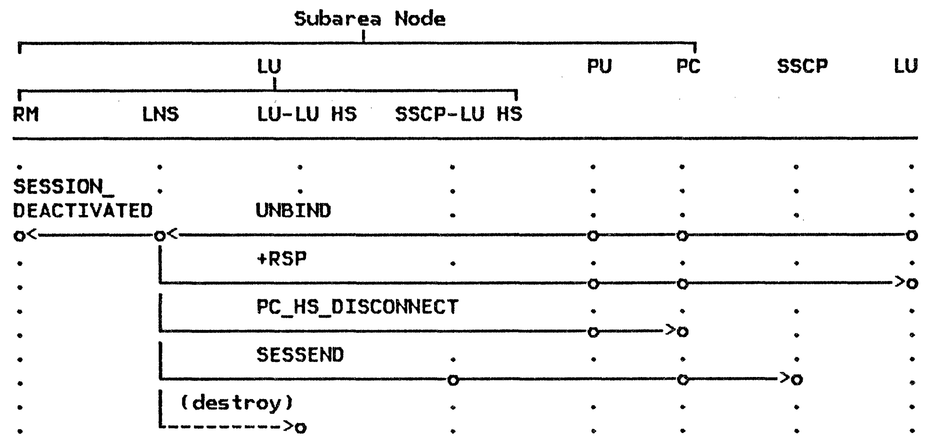


Figure 4-24. LU-LU Session Termination by Remote LU

The remaining pages of this chapter contain the formal description of LNS. This description consists of procedures, finite-state machines (FSMs), and data structures used only by LNS. The procedures are divided into two sections: High-level and low-level. The high-level procedures are organized hierarchically. The highest level is the root procedure of the calling tree, named LNS (same as the overall component). The LNS root procedure calls the other high-level procedures.

The low-level procedures are arranged alphabetically by name. Some are called by the high-level procedures; the others are called by the low-level procedures.

Throughout this formal description, certain error checks are described. The error checks that all implementations make are identified as being required. The other error checks described herein are optional; implementations may make some, none, or all of these checks. These required and optional error checks are the only error checks that implementations make.

LNS

| | |
|---|---|
| FUNCTION: | LU network services (LNS) is responsible for activating and deactivating sessions between this LU and another LU or a control point (CP). There is one LNS process per LU in the node, and it is created (destroyed) when the LU is created (destroyed). LNS receives records from the resources manager (RM), half-session (HS), and nodal NAU manager (NNM) processes. When the records are received, they are routed to the appropriate procedures where they are processed. LNS uses process data (called LOCAL) that can be accessed by any procedure in the LNS process. |
| INPUT: | Records from RM, HS, and NNM |
| OUTPUT: | Received records routed to appropriate procedures in LNS |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PROCESS_RECORD_FROM_RM | page 4-48 |
| PROCESS_RECORD_FROM_HS | page 4-48 |
| PROCESS_RECORD_FROM_NNM | page 4-50 |
| RM_TO_LNS_RECORD | page A-30 |
| NNM_TO_LNS_RECORD | page A-21 |
| HS_TO_LNS_RECORD | page A-10 |
| LOCAL | page 4-99 |
| LUCB | page A-1 |

```
Set up addressability to the control blocks used by LNS.  The LNS
 process data (LOCAL) is a data area that may be referenced by any procedure
 or FSM in LNS.  LOCAL is referenced only within LNS.
 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 LNS.  The CP-LU control block (CPLU_CB in
 LOCAL.CPLU_LIST) and LU-LU control block (LULU_CB in LOCAL.LULU_CB_LIST) are
 created and used only by LNS.

Do until LNS process is destroyed.
   Select based on one of the following conditions:
      When record is received from RM
         Call PROCESS_RECORD_FROM_RM(RM_TO_LNS_RECORD) (page 4-48).
      When record is received from HS
         Call PROCESS_RECORD_FROM_HS(HS_TO_LNS_RECORD) (page 4-48).
      When record is received from NNM
         Call PROCESS_RECORD_FROM_NNM(NNM_TO_LNS_RECORD) (page 4-50).
```

## PROCESS_RECORD_FROM_RM

| | |
|---|---|
| FUNCTION: | Route records received from RM to appropriate procedures. |
| INPUT: | RM_TO_LNS_RECORD (contains a request to activate or deactivate a session) |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PROCESS_ACTIVATE_SESSION | page 4-78 |
| PROCESS_DEACTIVATE_SESSION | page 4-86 |
| RM_TO_LNS_RECORD | page A-30 |
| ACTIVATE_SESSION | page A-31 |
| DEACTIVATE_SESSION | page A-31 |

```
Select based on RM_TO_LNS_RECORD type:
   When type is ACTIVATE_SESSION
      Call PROCESS_ACTIVATE_SESSION(ACTIVATE_SESSION) (page 4-78).
   When type is DEACTIVATE_SESSION
      Call PROCESS_DEACTIVATE_SESSION(DEACTIVATE_SESSION) (page 4-86).
```

## PROCESS_RECORD_FROM_HS

| | |
|---|---|
| FUNCTION: | Route records received from the half-session (HS) process to the appropriate procedures. The HS process represents either an LU-LU or an LU-CP session. If the record cannot be routed, a negative response is sent or the error is logged. |
| INPUT: | HS_TO_LNS_RECORD (usually contains a network services BIU) |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PROCESS_INIT_HS_RSP | page 4-87 |
| PROCESS_ABORT_HS | page 4-78 |
| PROCESS_CINIT_RQ | page 4-81 |
| PROCESS_NOTIFY_RQ | page 4-88 |
| PROCESS_ECHOTEST_RQ | page 4-86 |
| PROCESS_CLEANUP_RQ | page 4-83 |
| PROCESS_CTERM_RQ | page 4-84 |
| PROCESS_INIT_SELF_RSP | page 4-87 |
| PROCESS_TERM_SELF_RSP | page 4-90 |
| PROCESS_REQECHO_RSP | page 4-89 |
| PROCESS_NOTIFY_RSP | page 4-88 |
| BUILD_AND_SEND_RSP_OR_LOG | page 4-66 |
| LOCAL | page 4-99 |
| HS_TO_LNS_RECORD | page A-10 |
| INIT_HS_RSP | page A-11 |
| ABORT_HS | page A-11 |
| HS_RCV_RECORD | page A-11 |

```
Initialize LOCAL.SENSE_CODE to X'00000000'.

Select based on HS_TO_LNS_RECORD type:
   When type is INIT_HS_RSP (This record is received only from LU-LU half-sessions.
    INIT_HS_RSP from CP-LU half-session is explicitly received elsewhere.)
      Call PROCESS_INIT_HS_RSP(INIT_HS_RSP) (page 4-87).

   When type is ABORT_HS (received only from LU-LU half-sessions)
      Call PROCESS_ABORT_HS(ABORT_HS) (page 4-78).

   When type is HS_RCV_RECORD (received only from CP-LU half-sessions)
    (HS_RCV_RECORD always contains an NS header)
      Optionally check the format of the RH (see Figure 4-2 on page 4-8 for correct
       RH formats).
      If there is an RH format error then
         Call BUILD_AND_SEND_RSP_OR_LOG(HS_RCV_RECORD) (page 4-66)
          to send a negative response or log the error.

      Else
         If HS_RCV_RECORD contains a request (RH.RRI=RQ) then
            Select based on the NS header (first 3 bytes) in HS_RCV_RECORD.RU:
               When CINIT
                  Call PROCESS_CINIT_RQ(HS_RCV_RECORD) (page 4-81).
               When NOTIFY
                  Call PROCESS_NOTIFY_RQ(HS_RCV_RECORD) (page 4-88).
               When ECHOTEST
                  Call PROCESS_ECHOTEST_RQ(HS_RCV_RECORD) (page 4-86).
               When CLEANUP and this node is a subarea node
                  Call PROCESS_CLEANUP_RQ(HS_RCV_RECORD) (page 4-83).
               When CTERM and this node is a subarea node
                  Call PROCESS_CTERM_RQ(HS_RCV_RECORD) (page 4-84).
               Otherwise
                  Set LOCAL.SENSE_CODE to X'10030000' (function not supported).
                  Call BUILD_AND_SEND_RSP_OR_LOG(HS_RCV_RECORD) (page 4-66)
                   to send a negative response or log the error.

         Else (HS_RCV_RECORD contains a response)
            Select based on the NS header (first 3 bytes for positive response; 3 bytes
             following sense data for negative response) in HS_RCV_RECORD.RU:
               When INIT_SELF
                  Call PROCESS_INIT_SELF_RSP(HS_RCV_RECORD) (page 4-87).
               When TERM_SELF
                  Call PROCESS_TERM_SELF_RSP(HS_RCV_RECORD) (page 4-90).
               When REQECHO
                  Call PROCESS_REQECHO_RSP(HS_RCV_RECORD) (page 4-89).
               When NOTIFY
                  Call PROCESS_NOTIFY_RSP(HS_RCV_RECORD) (page 4-88).
               Otherwise
                  Optionally log the error with sense code 1003 (function not supported).
```

## PROCESS_RECORD_FROM_NNM

| | |
|---|---|
| FUNCTION: | Route records received from the nodal NAU manager (NNM) to the appropriate procedures. |
| INPUT: | NNM_TO_LNS_RECORD (usually contains a session activation or session deactivation BIU) |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PROCESS_BIND_RQ | page 4-80 |
| PROCESS_BIND_RSP | page 4-81 |
| PROCESS_UNBIND_RQ | page 4-90 |
| PROCESS_UNBIND_RSP | page 4-91 |
| PROCESS_ACTLU_RQ | page 4-79 |
| PROCESS_DACTLU_RQ | page 4-85 |
| PROCESS_PC_CONNECT_RSP | page 4-89 |
| PROCESS_SESSION_ROUTE_INOP | page 4-89 |
| PROCESS_HIERARCHICAL_RESET | page 4-86 |
| NNM_TO_LNS_RECORD | page A-21 |
| BIND_RQ_RCV_RECORD | page A-21 |
| BIND_RSP_RCV_RECORD | page A-22 |
| UNBIND_RQ_RCV_RECORD | page A-23 |
| UNBIND_RSP_RCV_RECORD | page A-23 |
| ACTLU_RQ_RCV_RECORD | page A-21 |
| DACTLU_RQ_RCV_RECORD | page A-22 |
| PC_CONNECT_RSP | page A-22 |
| SESSION_ROUTE_INOP | page A-23 |
| HIERARCHICAL_RESET | page A-22 |
| LOCAL | page 4-99 |

```
Set LOCAL.SENSE_CODE to X'00000000'.

Select based on NNM_TO_LNS_RECORD type:
   When type is BIND_RQ_RCV_RECORD
      Call PROCESS_BIND_RQ(BIND_RQ_RCV_RECORD) (page 4-80).
   When type is BIND_RSP_RCV_RECORD
      Call PROCESS_BIND_RSP(BIND_RSP_RCV_RECORD) (page 4-81).
   When type is UNBIND_RQ_RCV_RECORD
      Call PROCESS_UNBIND_RQ(UNBIND_RQ_RCV_RECORD) (page 4-90).
   When type is UNBIND_RSP_RCV_RECORD
      Call PROCESS_UNBIND_RSP(UNBIND_RSP_RCV_RECORD) (page 4-91).
   When type is ACTLU_RQ_RCV_RECORD
      Call PROCESS_ACTLU_RQ(ACTLU_RQ_RCV_RECORD) (page 4-79).
   When type is DACTLU_RQ_RCV_RECORD
      Call PROCESS_DACTLU_RQ(DACTLU_RQ_RCV_RECORD) (page 4-85).
   When type is PC_CONNECT_RSP
      Call PROCESS_PC_CONNECT_RSP(PC_CONNECT_RSP) (page 4-89).
   When type is SESSION_ROUTE_INOP
      Call PROCESS_SESSION_ROUTE_INOP(SESSION_ROUTE_INOP) (page 4-89).
   When type is HIERARCHICAL_RESET
      Call PROCESS_HIERARCHICAL_RESET(HIERARCHICAL_RESET) (page 4-86).
```

ACTIVATE_SESSION_ERROR

| | |
|---|---|
| FUNCTION: | Perform error checking upon receipt of an activate-session request from RM. These error checks are required. |
| OUTPUT: | TRUE if error; otherwise, FALSE. When TRUE, ERROR_TYPE is set. When FALSE, CP_ID is set. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LU_MODE_SESSION_LIMIT_EXCEEDED | page 4-77 |
| ACTIVATE_SESSION | page A-31 |
| PARTNER_LU | page A-2 |
| MODE | page A-3 |
| CP_ID | page A-2 |
| ERROR_TYPE | page 4-99 |

```
If there is not sufficient storage available to start a new session then
   Return with a value of TRUE (ERROR_TYPE—RETRY).

Determine the control point identifier (CP_ID) to be associated with the new
 LU-LU session.  For LUs in a subarea node, the CP_ID is obtained from the
 control block associated with the active SSCP-LU session.  If the SSCP-LU
 session is not active, the CP_ID cannot be obtained.  For LUs in a peripheral
 node, a "request CP identifier (RQCPID)" record is sent to the local control
 point (PNCP).  The PNCP attempts to determine the control point to be used
 based on the (partner LU, modename) pair.  If determined successfully and an
 active session exists between this LU and that CP, a RQCPID response record
 is returned containing the correct CP_ID.  Otherwise, a negative RQCPID
 response is returned containing no CP_ID.
If the CP_ID cannot be obtained then
   Return with a value of TRUE (ERROR_TYPE—NO_RETRY).

Locate the PARTNER_LU and MODE control blocks using the partner LU and mode names
 from the passed ACTIVATE_SESSION record.
Call LU_MODE_SESSION_LIMIT_EXCEEDED(PARTNER_LU.FULLY_QUALIFIED_LU_NAME, MODE,
 ACTIVATE_SESSION.SESSION_TYPE, ACTIVE_AND_PENDING_ACTIVE) (page 4-77).
If the LU-LU session limit for the (partner LU, modename) pair will be
 exceeded then
   Return with a value of TRUE (ERROR_TYPE—RETRY).

If the LU-LU session limit associated with the control point will be exceeded then
 (The control point has a session limit for every LU it is mediating sessions for.
 This limit indicates the maximum number of sessions this LU may have with other
 LUs.)
   Return with a value of TRUE (ERROR_TYPE—RETRY).

If the new LU-LU session is to be PNCP-mediated and this LU is unable to act as
 a PLU then
   Return with a value of TRUE (ERROR_TYPE—NO_RETRY).

If this LU is unable to act as a PLU or SLU then
   Return with a value of TRUE (ERROR_TYPE—NO_RETRY).

Return with a value of FALSE (no error found).
```

BIND_RQ_STATE_ERROR

| | |
|---|---|
| FUNCTION: | Determine if there is a state error on receipt of a BIND request. Required checks are explicitly indicated. |
| INPUT: | BIND_RQ_RCV_RECORD |
| OUTPUT: | TRUE if error; otherwise, FALSE. If TRUE, LOCAL.SENSE_CODE is set to appropriate sense data. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| BIND_SESSION_LIMIT_EXCEEDED | page 4-56 |
| LOCAL | page 4-99 |
| BIND_RQ_RCV_RECORD | page A-21 |
| PARTNER_LU | page A-2 |
| MODE | page A-3 |
| SESSION_TYPE | page 4-99 |

```
Note: The following checks are optional except when specifically indicated
 as required.

If there is insufficient storage available to establish a new session then
   Set LOCAL.SENSE_CODE to X'08120000' (insufficient resources).
   Return with a value of TRUE (error).

If this LU is currently unable to act as an SLU then
   Set LOCAL.SENSE_CODE to X'083A0000' (LU not enabled).
   Return with a value of TRUE (error).

Locate the PARTNER_LU control block using the user data PLU name field in BIND.
Locate the MODE control block using the user data mode name field in BIND.
If either control block cannot be located then
   Set LOCAL.SENSE_CODE to X'0835xxxx' (xxxx is offset to PLU name or mode name).
   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 SESSION_TYPE to FIRST_SPEAKER.

Else (use value in BIND request)
   If BIND specifies the secondary as contention winner then
      Set SESSION_TYPE to FIRST_SPEAKER.
   Else
      Set SESSION_TYPE to BIDDER.

Call BIND_SESSION_LIMIT_EXCEEDED(PARTNER_LU.FULLY_QUALIFIED_LU_NAME, MODE,
 SESSION_TYPE, BIND_RQ_RCV_RECORD.ADDRESS) (page 4-56).
If the session limit will be exceeded then
   Return with a value of TRUE (LOCAL.SENSE_CODE is set by BIND_SESSION_LIMIT_EXCEEDED).

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 Appendix E).
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).

If this LU's cryptography support capability does not match that specified in BIND then
 (this check is required)
   Set LOCAL.SENSE_CODE to X'0835xxxx' (xxxx is offset to cryptography field).
   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 (this check is required)
   Set LOCAL.SENSE_CODE to X'08480000' (cryptography function inoperative).
   Return with a value of TRUE (error).

If a duplicate user data session instance identifier exists then (another active session
 is using the same identifier--use SESSION_ID field in LULU_CB)
   Set LOCAL.SENSE_CODE to X'08520001' (duplicate session-activation request).
   Return with a value of TRUE (error).
```

```
If the SLU supports sending segments, the PLU does not support receiving
 segments, and the lower bound of the maximum RU size (see the discussion
 in BIND [page 4-19]) sent for this (partner-LU, mode name) pair
 is greater than the maximum RU size for the link then
   Set LOCAL.SENSE_CODE to X'0835xxxx' (xxxx is offset to segmenting field).
   Return with a value of TRUE (error).
```

## BIND_RSP_STATE_ERROR

| | |
|---|---|
| FUNCTION: | Perform state error checking on a received BIND response. Required checks are explicitly indicated. |
| INPUT: | BIND_RSP_RCV_RECORD, LULU_CB |
| OUTPUT: | TRUE if error; otherwise, FALSE |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| BIND_RSP_RCV_RECORD | page A-22 |
| LULU_CB | page A-5 |

```
Note: The following checks are done only on positive response to BIND and
 are optional except where explicitly indicated as required.

If the BIND request specified that an alternate code set will not be used and
 the BIND response specifies that an alternate code set may be used then
   Return with a value of TRUE (error).
```

Pacing and maximum RU size checks

```
If the pacing staging indicators in the BIND response are not the same as
 those specified in the BIND request then
   Return with a value of TRUE (error).

If secondary-to-primary pacing is one-stage and the secondary send window size
 in the BIND response is not the same as that specified in the BIND request then
   Return with a value of TRUE (error).

If the secondary receive window size in the BIND response is greater than that
 specified in the BIND request or the primary send window size is greater than
 that specified in the BIND request then (a window size of 0 is treated as
 infinitely large for these comparisons)
   Return with a value of TRUE (error).

If the primary receive window size in the BIND response is not the same as
 that specified in the BIND request then
   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 segmenting, then the
 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
   Return with a value of TRUE (error).
```

```
PS usage checks

If there are other active sessions for this (partner-LU, mode name) pair and
 the values of the BIND response fields for synchronization level and session
 reinitiation do not equal those of the other active sessions then
   Return with a value of TRUE (consistency error).

Else (no other sessions active for this (partner-LU, mode name) pair)
   If the BIND response specifies a synchronization level of Confirm, Sync Point,
    and Backout and the BIND request specified only Confirm then
      Return with a value of TRUE (error).
   If the BIND response specifies session reinitiation responsibility as not
    operator controlled and the BIND request specified operator controlled then
      Return with a value of TRUE (error).

   If the BIND response specifies session reinitiation responsibility as
    secondary will reinitiate and the BIND request specified primary will
    reinitiate then
      Return with a value of TRUE (error).

   If the BIND response specifies session reinitiation responsibility as
    primary will reinitiate and the BIND request specified secondary will
    reinitiate then
      Return with a value of TRUE (error).

If the values of the BIND response fields for parallel sessions support and
 change number of sessions support are not the same as specified in the BIND
 request then
   Return with a value of TRUE (error).

Contention winner checks

If the BIND response specifies parallel sessions supported then
   If the value of the BIND response contention winner field is not the
    same as that specified in the BIND request then
      Return with a value of TRUE (error).

Else (parallel sessions not supported)
   If the BIND response contention winner is specified as the primary and the
    BIND request was specified as the secondary then
      Return with a value of TRUE (error).

Cryptography checks (these checks are required).

If the BIND response cryptography field values are not the same as those specified
 in the BIND request then
   Return with a value of TRUE (error).

PLU name (not in user data) checks

If the primary LU name (the one not in the User Data field) in the BIND
 response is not the same as that specified in the BIND request then
   Return with a value of TRUE (error).
```

```
        User data subfield checks

If the user-data mode name in the BIND response is not the same as that
 specified in the BIND request then
   Return with a value of TRUE (error).

If the user-data session-instance identifier in the BIND response is not
 specified correctly (see page E-16) then
   Return with a value of TRUE (error).

        URC checks

If the URC in the BIND response is not the same as that specified
 in the BIND request 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 request. |
| INPUT: | PARTNER_LU.FULLY_QUALIFIED_LUNAME, MODE, SESSION_TYPE (FIRST_SPEAKER or BIDDER), ADDRESS (TH address fields from the received BIND request). |
| OUTPUT: | TRUE if limits exceeded; otherwise, FALSE. If TRUE, LOCAL.SENSE_CODE is set to appropriate sense data. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LU_MODE_SESSION_LIMIT_EXCEEDED | page 4-77 |
| LOCAL | page 4-99 |
| MODE | page A-3 |
| ADDRESS | page A-33 |
| PARTNER_LU | page A-2 |

```
  Call LU_MODE_SESSION_LIMIT_EXCEEDED(PARTNER_LU.FULLY_QUALIFIED_LU_NAME, MODE, SESSION_TYPE, AC
TIVE)
   (page 4-77).
  If the session limit will be exceeded then
     Return with a value of TRUE (LOCAL.SENSE_CODE is set by
      LU_MODE_SESSION_LIMIT_EXCEEDED).

  The following handles BIND race conditions.
  Call LU_MODE_SESSION_LIMIT_EXCEEDED(PARTNER_LU.FULLY_QUALIFIED_LU_NAME,MODE,SESSION_TYPE,
   ACTIVE_AND_PENDING_ACTIVE) (page 4-77).
  If the session limit will be exceeded then (LOCAL.SENSE_CODE is set if the
   limit will be exceeded)
     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 sequ
ence.
      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 neither LU name is known, the result is equivalent to name equality in the
      comparison (this can occur only for LUs in peripheral nodes that do not know
      their names).  When this occurs, the winner is determined by using the ODAI
      field in the ADDRESS of the BIND request.  If the ODAI value is 0, then this
      LU is the winner; otherwise, the other LU is the winner.

     If this LU is the winner then
        Return with a value of TRUE.

     Else
        Reset LOCAL.SENSE_CODE to X'00000000'.
        Return with a value of FALSE.

  Else (session limit will not be exceeded)
     Return with a value of FALSE.
```

## 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 (indicates retry or no retry). |
| OUTPUT: | ACTIVATE_SESSION_RSP sent to RM |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| ACTIVATE_SESSION_RSP | page A-20 |
| ACTIVATE_SESSION | page A-31 |
| LULU_CB | page A-5 |
| ERROR_TYPE | page 4-99 |

Create 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 ACTIVATE_SESSION_RSP to RM.

## BUILD_AND_SEND_ACT_SESS_RSP_POS

| | |
|---|---|
| FUNCTION: | Build and send ACTIVATE_SESSION_RSP (positive) to RM. |
| INPUT: | LULU_CB |
| OUTPUT: | ACTIVATE_SESSION_RSP sent to RM |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LULU_CB | page A-5 |
| ACTIVATE_SESSION_RSP | page A-20 |

Create ACTIVATE_SESSION_RSP record.
Set ACTIVATE_SESSION_RSP.CORRELATOR to LULU_CB.CORRELATOR (enables RM to correlate this response with the original ACTIVATE_SESSION request).
Set ACTIVATE_SESSION_RSP.TYPE to POS (positive).
Set ACTIVATE_SESSION_RSP.HS_ID to LULU_CB.LU_LU.HS_ID (the identifier of the half-session just activated).
Set ACTIVATE_SESSION_RSP.SESSION_INFORMATION.HALF_SESSION_TYPE to LULU_CB.HALF_SESSION_TYPE (primary or secondary).
Set ACTIVATE_SESSION_RSP.SESSION_INFORMATION.BRACKET_TYPE to LULU_CB.SESSION_TYPE (first speaker or bidder).

Send ACTIVATE_SESSION_RSP to RM.

## BUILD_AND_SEND_ACTLU_RSP_NEG

| | |
|---|---|
| FUNCTION: | Build and send a negative ACTLU response. |
| INPUT: | ACTLU_RQ_RCV_RECORD |
| OUTPUT: | ACTLU_RSP_SEND_RECORD sent to NNM |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 4-99 |
| ACTLU_RQ_RCV_RECORD | page A-21 |
| ACTLU_RSP_SEND_RECORD | page A-17 |

```
Create ACTLU_RSP_SEND_RECORD.
Set ACTLU_RSP_SEND_RECORD.LU_ID to this LU's identifier.
Copy the PC_ID, ADDRESS, EFI, SNF, and RH from the ACTLU_RQ_RCV_RECORD to
 the ACTLU_RSP_SEND_RECORD.
Set ACTLU_RSP_SEND_RECORD.DCF to the appropriate value.
Set ACTLU_RSP_SEND_RECORD.RH to indicate negative response (SD, NEG, and
 byte 2 of RH set to all 0's).
Set ACTLU_RSP_SEND_RECORD.RU to the sense data (from LOCAL.SENSE_CODE) followed
 by the ACTLU request code.

Send the ACTLU_RSP_SEND_RECORD to the CP via the nodal NAU manager.
```

## BUILD_AND_SEND_ACTLU_RSP_POS

| | |
|---|---|
| FUNCTION: | Build and send a positive ACTLU response. Also, initialize the CP-LU half-session. |
| INPUT: | ACTLU_RQ_RCV_RECORD, INIT_HS_RSP |
| OUTPUT: | CP-LU half-session initialized and ACTLU_RSP_SEND_RECORD sent to NNM |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_STATUS | page 4-92 |
| LULU_CB | page A-5 |
| ACTLU_RQ_RCV_RECORD | page A-21 |
| ACTLU_RSP_SEND_RECORD | page A-17 |
| INIT_HS | page A-16 |
| INIT_HS_RSP | page A-11 |

```
Create ACTLU_RSP_SEND_RECORD.
Set ACTLU_RSP_SEND_RECORD.LU_ID to this LU's identifier.
Copy the PC_ID, ADDRESS, EFI, SNF, and RH fields from the ACTLU_RQ_RCV_RECORD
 into the ACTLU_RSP_SEND_RECORD.
Set ACTLU_RSP_SEND_RECORD.RH to indicate positive response (RSP, ¬SD, POS,
 and RH byte 2 set to all 0's).

The following builds the ACTLU response RU--see Appendix E description for
 field settings not explicitly shown here.
Find all LU-LU sessions (represented by LULU_CBs) being
 mediated by this CP (the CP_ID in ACTLU matches that in the LULU_CB).
If there are no active or pending active LU-LU sessions being mediated by this
 CP then
   Set ACTLU response type of activation to cold.
   Reset all LU-LU sessions being mediated by this CP by calling FSM_STATUS
    with a RESET_NORMAL signal (page 4-92).

Else
   Set ACTLU response type of activation to ERP.

If FM profile 6 is supported by this LU then
   Set the FM profile field in the ACTLU response to 6.

Build control vector X'00' (Appendix E).
Build control vector X'0C' (Appendix E).  Peripheral nodes always suppress
 sending the SESSST RU; subarea nodes always send SESSST.  The LU-LU session
 count is determined by counting all the LULU_CBs associated with (being
 mediated by) this CP.  For peripheral node sessions with an SSCP (as opposed
 to a PNCP) the primary LU capability is inhibited (not able ever to be a primary),
 the LU-LU session limit is 1, and parallel session capability is not supported.
Put the control vectors in the ACTLU response RU.
(End of building ACTLU response RU.)

Create the new CP-LU half-session process.
Create an INIT_HS record to be sent to the CP-LU half-session.
Set INIT_HS.PC_ID to ACTLU_RQ_RCV_RECORD.PC_ID.
Set INIT_HS.TYPE to secondary (LU is always secondary with respect to CP).
Set INIT_HS.DATA_TYPE to indicate this record contains an ACTLU_IMAGE.
Set INIT_HS.ACTLU_IMAGE.FM_PROFILE and TS_PROFILE to the corresponding field values
 from the ACTLU response RU.
Set INIT_HS.ACTLU_IMAGE.MAX_RU_SIZE to the maximum RU size allowed on this session
 (implementation-defined).

Send the ACTLU_RSP_SEND_RECORD to the CP via the nodal NAU manager (NNM).
Send the INIT_HS record to the CP-LU half-session.
Receive the INIT_HS_RSP from the CP-LU half-session (this response is always positive).
 This response is used just so CP-LU and LU-LU half-sessions operate in the same
 manner.
```

## BUILD_AND_SEND_BIND_RQ

| | |
|---|---|
| FUNCTION: | Build and send a BIND request. |
| INPUT: | LULU_CB |
| OUTPUT: | BIND_RQ_SEND_RECORD sent to NNM |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LULU_CB | page A-5 |
| BIND_RQ_SEND_RECORD | page A-17 |

```
Create BIND_RQ_SEND_RECORD to contain the BIND request.
Set BIND_RQ_SEND_RECORD.LU_ID to this LU's identifier.
Set BIND_RQ_SEND_RECORD.PC_ID to LULU_CB.LU_LU.PC_ID (identifies the path control
 that the BIND will flow through).
Set BIND_RQ_SEND_RECORD.ADDRESS to LULU_CB.LU_LU.ADDRESS (TH addresses).
Set BIND_RQ_SEND_RECORD.EFI to EXP (expedited).
Set BIND_RQ_SEND_RECORD.SNF to a unique identifier.  This identifier is
 also saved in LULU_CB.SENT_BIND_RQ.SNF for correlating the BIND response later.
Set BIND_RQ_SEND_RECORD.RH to the appropriate values (Figure 4-3 on page 4-16).
Set BIND_RQ_SEND_RECORD.RU to the appropriate values (see page 4-19).
Set BIND_RQ_SEND_RECORD.DCF to the appropriate value.
Save the BIND request in the LULU_CB for later uses (e.g., checking the BIND response).

Send BIND_RQ_SEND_RECORD to the other LU via the nodal NAU manager.
```

## BUILD_AND_SEND_BIND_RSP_NEG

| | |
|---|---|
| FUNCTION: | Build and send a negative BIND response. |
| INPUT: | BIND_RQ_RCV_RECORD |
| OUTPUT: | BIND_RSP_SEND_RECORD sent to NNM |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 4-99 |
| BIND_RQ_RCV_RECORD | page A-21 |
| BIND_RSP_SEND_RECORD | page A-17 |

```
Create the BIND_RSP_SEND_RECORD to contain the negative BIND response.
Set BIND_RSP_SEND_RECORD.LU_ID to this LU's identifier.
Copy the PC_ID, ADDRESS, EFI, SNF, and RH from the BIND_RQ_RCV_RECORD into
 the BIND_RSP_SEND_RECORD.
Indicate negative response in BIND_RSP_SEND_RECORD.RH (RSP, SD, NEG, and
 byte 2 of RH set to all 0's).
Set BIND_RSP_SEND_RECORD.RU to LOCAL.SENSE_CODE followed by BIND request code.
Set BIND_RSP_SEND_RECORD.DCF field to appropriate value.

Send BIND_RSP_SEND_RECORD to the LU via the nodal NAU manager.
```

## BUILD_AND_SEND_BIND_RSP_POS

| | |
|---|---|
| FUNCTION: | Build and send a positive BIND response. |
| INPUT: | BIND_RQ_RCV_RECORD, LULU_CB |
| OUTPUT: | BIND_RSP_SEND_RECORD sent to NNM and BIND image returned (i.e. BIND image from BIND response just sent) |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LULU_CB | page A-5 |
| BIND_RQ_RCV_RECORD | page A-21 |
| BIND_RSP_SEND_RECORD | page A-17 |
| PARTNER_LU | page A-2 |
| MODE | page A-3 |

```
Create BIND_RSP_SEND_RECORD to contain the positive BIND response.
Set BIND_RSP_SEND_RECORD.LU_ID to this LU's identifier.
Copy the PC_ID, ADDRESS, EFI, SNF, and RH from the BIND_RQ_RCV_RECORD into
 the BIND_RSP_SEND_RECORD.
Set BIND_RSP_SEND_RECORD.RH to indicate positive response (RSP, ¬SD, POS, and
 byte 2 of RH set to all 0's).
Set BIND_RSP_SEND_RECORD.RU to the appropriate values (see page 4-25).
 The PARTNER_LU, MODE, and LULU_CB are used in construction of the BIND RU.
Set BIND_RSP_SEND_RECORD.DCF to appropriate value.

Send BIND_RSP_SEND_RECORD to the other LU via the nodal NAU manager.
Return a copy of BIND image from the BIND response RU.
```

## BUILD_AND_SEND_BINDF_RQ

| | |
|---|---|
| FUNCTION: | Build and send BINDF (BIND failure) request to SSCP (via SSCP-LU half-session). This procedure is used only within subarea nodes. |
| INPUT: | Reason code (type of BINDF to send), LULU_CB, sense data |
| OUTPUT: | HS_SEND_RECORD (containing BINDF request) sent to SSCP-LU half-session |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LULU_CB | page A-5 |
| HS_SEND_RECORD | page A-16 |

```
If this node is a subarea node then (the BINDF request is sent only by subarea nodes)
   Create HS_SEND_RECORD to contain the BINDF request.
   Set HS_SEND_RECORD.EFI to NORMAL.
   Set HS_SEND_RECORD.SNF to a unique identifier.
   Set HS_SEND_RECORD.DCF to appropriate value.
   Set HS_SEND_RECORD.RH to appropriate values (Figure 4-2 on page 4-8).
   Set HS_SEND_RECORD.RU (see BINDF request in Appendix E).  Set the BINDF reason
    field in accordance with the passed reason code and the BINDF sense data to the
    passed sense data parameter.  The passed LULU_CB contains information (e.g., addresses)
    used in building the RU.

   Send HS_SEND_RECORD to the CP via the CP-LU half-session.
```

## BUILD_AND_SEND_CINIT_RSP

| | |
|---|---|
| FUNCTION: | Build and send a positive or negative CINIT response. |
| INPUT: | HS_RCV_RECORD containing CINIT request. LOCAL.SENSE_CODE indicates what type of response (positive or negative) to build. |
| OUTPUT: | HS_SEND_RECORD (containing CINIT response) sent to CP via CP-LU half-session |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 4-99 |
| HS_RCV_RECORD | page A-11 |
| HS_SEND_RECORD | page A-16 |

```
Create HS_SEND_RECORD to contain CINIT response.
Copy the EFI, SNF, and RH from the HS_RCV_RECORD to the HS_SEND_RECORD.

If LOCAL.SENSE_CODE = X'00000000' then (build a positive response)
   Set HS_SEND_RECORD.RH to indicate positive response (RSP, ¬SD, POS, and
    byte 2 of RH set to all 0's).
   Set HS_SEND_RECORD.RU to the CINIT request code.
   If there are any unknown control vectors in the CINIT request RU then
      Append control vector X'FE' to the CINIT response RU (see control vectors
       in Appendix E).

Else (build a negative response)
   Set HS_SEND_RECORD.RH to indicate negative response (RSP, SD, NEG, and
    byte 2 of RH set to all 0's).
   Set HS_SEND_RECORD.RU to the sense data (LOCAL.SENSE_CODE) followed by
    the CINIT request code.

Set the HS_SEND_RECORD.DCF to the appropriate value.
Send the HS_SEND_RECORD (containing the CINIT response) to the CP via the
 CP-LU half-session.
```

## BUILD_AND_SEND_DACTLU_RSP

| | |
|---|---|
| FUNCTION: | Build and send a positive or negative DACTLU response. |
| INPUT: | DACTLU_RQ_RCV_RECORD |
| OUTPUT: | DACTLU_RSP_SEND_RECORD sent to CP via nodal NAU manager |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 4-99 |
| DACTLU_RSP_SEND_RECORD | page A-17 |
| DACTLU_RQ_RCV_RECORD | page A-22 |

```
Create the DACTLU_RSP_SEND_RECORD to contain the DACTLU response.
Set DACTLU_RSP_SEND_RECORD.LU_ID to this LU's identifier.
Copy the PC_ID, ADDRESS, EFI, SNF, and RH from the DACTLU_RQ_RCV_RECORD into
 the DACTLU_RSP_SEND_RECORD.

If LOCAL.SENSE_CODE = X'00000000' then (build a positive response)
   Set DACTLU_RSP_SEND_RECORD.RH to indicate positive response (RSP, ¬SD, POS,
    and byte 2 of RH set to all 0's).
   Set DACTLU_RSP_SEND_RECORD.RU to DACTLU request code.
   Set DACTLU_RSP_SEND_RECORD.DCF to appropriate value.

Else (build a negative response)
   Set DACTLU_RSP_SEND_RECORD.RH to indicate negative response (RSP, SD, NEG,
    and byte 2 of RH set to all 0's).
   Set DACTLU_RSP_SEND_RECORD.RU to LOCAL.SENSE_CODE followed by the DACTLU
    request code.
   Set DACTLU_RSP_SEND_RECORD.DCF to appropriate value.

Send DACTLU_RSP_SEND_RECORD to the CP via the nodal NAU manager.
```

## BUILD_AND_SEND_DEACTIVATE_SESS

| | |
|---|---|
| FUNCTION: | Build and send CTERM_DEACTIVATE_SESSION to RM. This is sent to RM when a CTERM-ORDERLY is received for an active LU-LU session. LNS cannot deactivate a session in an orderly manner because it does not know when to send BIS. Therefore, it must tell RM to do it. |
| INPUT: | HS identifier of the half-session to be deactivated |
| OUTPUT: | CTERM_DEACTIVATE_SESSION sent to RM |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| CTERM_DEACTIVATE_SESSION | page A-20 |

```
Create CTERM_DEACTIVATE_SESSION record.
Set CTERM_DEACTIVATE_SESSION.HS_ID to passed HS identifier (identifies the
 half-session to be deactivated).

Send CTERM_DEACTIVATE_SESSION to RM.
```

## BUILD_AND_SEND_HIER_RESET_RSP

| | |
|---|---|
| FUNCTION: | Build and send a HIERARCHICAL_RESET response to the nodal NAU manager. |
| INPUT: | HIERARCHICAL_RESET |
| OUTPUT: | HIERARCHICAL_RESET_RSP sent to nodal NAU manager |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| HIERARCHICAL_RESET_RSP | page A-18 |
| HIERARCHICAL_RESET | page A-22 |

```
Create HIERARCHICAL_RESET_RSP record.
Set HIERARCHICAL_RESET_RSP.LU_ID to this LU's identifier.
Copy the PC_ID and CP_ID fields from HIERARCHICAL_RESET into HIERARCHICAL_RESET_RSP.

Send HIERARCHICAL_RESET_RSP to the nodal NAU manager.
```

## 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, BIND image, and half-session type (PRI or SEC) |
| OUTPUT: | INIT_HS sent to HS (LU-LU half-session) |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LULU_CB | page A-5 |
| INIT_HS | page A-16 |

```
Create INIT_HS record.
Set INIT_HS.PC_ID to LULU_CB.LU_LU.PC_ID (path control the LU-LU half-session will
 send to and receive from).
Set INIT_HS.TYPE to passed half-session type parameter (primary or secondary).
Set INIT_HS.DATA_TYPE to BIND image type (indicates data is a BIND image).
Set INIT_HS.DATA.BIND_IMAGE to the passed BIND image (half-session protocols are based on
 fields in the BIND image).

Send INIT_HS record to HS (the LU-LU half-session identified by LULU_CB.LU_LU.HS_ID).
```

## BUILD_AND_SEND_INIT_RQ

| | |
|---|---|
| FUNCTION: | Build and send an INIT-SELF request to the control point (SSCP or PNCP). |
| INPUT: | LULU_CB, DLU role (PLU or SLU) |
| OUTPUT: | HS_SEND_RECORD (containing INIT-SELF request) sent to CP-LU half-session |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LULU_CB | page A-5 |
| HS_SEND_RECORD | page A-16 |

```
Create HS_SEND_RECORD to contain INIT-SELF request.
Set HS_SEND_RECORD.EFI to NORMAL.
Set HS_SEND_RECORD.SNF to a unique identifier.
Set HS_SEND_RECORD.RH to appropriate values (Figure 4-2 on page 4-8).
Set HS_SEND_RECORD.RU to appropriate values (see INIT-SELF request in Appendix E).
 The choice of initiate type (I or I/Q) is installation defined.  The PLU/SLU
 specification is set according to the passed DLU role parameter.
Set HS_SEND_RECORD.DCF to the appropriate value.

(Save information from the INIT-SELF request.  This information is used to correlate
 with the INIT-SELF response [SNF] and with the CINIT or BIND request [URC].)
Set LULU_CB.SENT_INITIATE_RQ.SNF to HS_SEND_RECORD.SNF.
Set LULU_CB.SENT_INITIATE_RQ.URC to the URC field of the INIT-SELF RU.

Send HS_SEND_RECORD to the CP via the CP-LU half-session.
```

## BUILD_AND_SEND_RSP_OR_LOG

| | |
|---|---|
| FUNCTION: | Build and send a positive or negative response to passed HS_RCV_RECORD if possible. If an error has occurred and a negative response cannot be sent, the error is logged. |
| INPUT: | HS_RCV_RECORD (to be responded to), LOCAL.SENSE_CODE (has nonzero value if error occurred) |
| OUTPUT: | HS_SEND_RECORD (containing response) sent to HS (CP-LU half-session), or error is logged |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 4-99 |
| HS_SEND_RECORD | page A-16 |
| HS_RCV_RECORD | page A-11 |

```
If HS_RCV_RECORD contains a response or a request asking for no response then
   If LOCAL.SENSE_CODE is nonzero then
      Optionally log the error.

Else (request that requires a response)
   Create HS_SEND_RECORD to contain response.
   Set HS_SEND_RECORD.PIU to HS_RCV_RECORD.PIU (copy request PIU into response).
   Set HS_SEND_RECORD.RH to indicate response (RSP, BC, EC, ¬PAC, and byte 2
    of RH set to all 0's).
   Set HS_SEND_RECORD.RU with data other than sense data.  For formatted FMD
    requests use the 3-byte NS header; for any non-FMD request use the 1-byte
    request code; otherwise, use no data.

   If LOCAL.SENSE_CODE = X'00000000' then (build positive response)
      Set HS_SEND_RECORD.RH to indicate a positive response (¬SD, POS).

   Else (build negative response)
      Set HS_SEND_RECORD.RH to indicate a negative response (SD, NEG).
      Insert LOCAL.SENSE_CODE in HS_SEND_RECORD.RU (first 4 bytes of RU).

   Set HS_SEND_RECORD.DCF to appropriate value.

   Send HS_SEND_RECORD to the control point via the CP-LU half-session.
```

## BUILD_AND_SEND_PC_CONNECT

| | |
|---|---|
| FUNCTION: | Build and send a path control connect record. The purpose of this record is to obtain (via a response) the process ID (PC_ID) of the path control to which BIND will be sent and get path control characteristics necessary to build a BIND request. Also, for peripheral nodes only, this procedure obtains the address that will represent the LU-LU session being activated. For subarea nodes, this record may cause a virtual route to be activated. |
| INPUT: | LULU_CB |
| OUTPUT: | PC_CONNECT sent to nodal NAU manager |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LULU_CB | page A-5 |
| PC_CONNECT | page A-18 |

```
Create PC_CONNECT record.
Set PC_CONNECT.LU_ID to this LU's identifier.
Set PC_CONNECT.HS_ID to LULU_CB.LU_LU.HS_ID (half-session process identifier).

If this node is a peripheral node then
   Set PC_CONNECT.TYPE to PERIPHERAL.
   Set PC_CONNECT.ALS to LULU_CB.LU_LU.ALS (identifies adjacent link station
    to be used for this LU-LU session).

Else (subarea node)
   Set PC_CONNECT.TYPE to SUBAREA.
   Set PC_CONNECT.PATH_INFORMATION to the class-of-service and virtual-route-identifier-
    list from the control vector X'0D' of the CINIT request.  This information is used to
    select the virtual route.
   Set PC_CONNECT.SUBAREA_ADDRESS to the subarea portion of the address of the target LU.

Send PC_CONNECT to path control via the nodal NAU manager.
```

## BUILD_AND_SEND_PC_HS_CONNECT

| | |
|---|---|
| FUNCTION: | Build and send a path control half-session connect record. The purpose of this record is to tell path control a new half-session process has been started that uses the specified address. Path control needs this HS_ID/ADDRESS relationship in order to route incoming PIUs and build THs for outgoing PIUs. |
| INPUT: | Process identifier (PC ID) of the path control to which the PC_HS_CONNECT record is to be sent, process identifier (HS ID) of the half-session just activated, ADDRESS (address for the half-session) |
| OUTPUT: | PC_HS_CONNECT sent to path control via nodal NAU manager |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PC_HS_CONNECT | page A-18 |
| ADDRESS | page A-33 |

```
Create PC_HS_CONNECT record.
Set PC_HS_CONNECT.LU_ID to this LU's identifier.
Set PC_HS_CONNECT.PC_ID to passed path control identifier.
Set PC_HS_CONNECT.HS_ID to passed half-session identifier.
Set PC_HS_CONNECT.ADDRESS to passed ADDRESS (TH addresses).

Send PC_HS_CONNECT to path control via the nodal NAU manager.
```

## BUILD_AND_SEND_PC_HS_DISCONNECT

| | |
|---|---|
| FUNCTION: | Build and send a path control half-session disconnect record. This is to notify path control that a half-session is deactivated. |
| INPUT: | Half-session process identifier (HS ID) |
| OUTPUT: | PC_HS_DISCONNECT sent to path control via nodal NAU manager |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PC_HS_DISCONNECT | page A-18 |

```
Create PC_HS_DISCONNECT record.
Set PC_HS_DISCONNECT.LU_ID to this LU's identifier.
Set PC_HS_DISCONNECT.HS_ID to passed half-session identifier.

Send PC_HS_DISCONNECT to path control via nodal NAU manager.
```

## BUILD_AND_SEND_SESS_ACTIVATED

| | |
|---|---|
| FUNCTION: | Build and send SESSION_ACTIVATED to RM to indicate that a half-session has become active. It also indicates information about the half-session. |
| INPUT: | LULU_CB |
| OUTPUT: | SESSION_ACTIVATED sent to RM |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LULU_CB | page A-5 |
| SESSION_ACTIVATED | page A-20 |

```
Create SESSION_ACTIVATED record.
Set SESSION_ACTIVATED.HS_ID to LULU_CB.LU_LU.HS_ID (identifies half-session that
 has been activated).
Set SESSION_ACTIVATED.SESSION_INFORMATION.HALF_SESSION_TYPE to
 LULU_CB.HALF_SESSION_TYPE (indicates primary or secondary).
Set SESSION_ACTIVATED.SESSION_INFORMATION.BRACKET_TYPE to LULU_CB.SESSION_TYPE
 (indicates bidder or first speaker).
Set SESSION_ACTIVATED.LU_NAME to LULU_CB.LUNAME.LOCAL (locally known name of the
 target LU).
Set SESSION_ACTIVATED.MODE_NAME to LULU_CB.MODENAME.

Send SESSION_ACTIVATED TO RM (notify RM that an LU-LU session has been activated).
```

## BUILD_AND_SEND_SESS_DEACTIVATED

| | |
|---|---|
| FUNCTION: | Build and send SESSION_DEACTIVATED to RM to indicate that a session has been deactivated. |
| INPUT: | Process identifier (HS ID) of half-session deactivated, reason code (reason for deactivation) |
| OUTPUT: | SESSION_DEACTIVATED sent to RM |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SESSION_DEACTIVATED | page A-21 |

```
Create SESSION_DEACTIVATED record.
Set SESSION_DEACTIVATED.HS_ID to passed HS ID (identifies half-session that was
 deactivated).
Set SESSION_DEACTIVATED.REASON to passed reason code (indicates the reason the
 half-session was deactivated).

Send SESSION_DEACTIVATED to RM (notify RM that an LU-LU session has been deactivated).
```

## BUILD_AND_SEND_SESSEND_RQ

| | |
|---|---|
| FUNCTION: | Build and send SESSEND request to the control point (SSCP or PNCP) to indicate that a session has ended. |
| INPUT: | LULU_CB |
| OUTPUT: | HS_SEND_RECORD (containing SESSEND request) sent to CP-LU half-session |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LULU_CB | page A-5 |
| HS_SEND_RECORD | page A-16 |

```
Create HS_SEND_RECORD to contain SESSEND request.
Set HS_SEND_RECORD.EFI to NORMAL.
Set HS_SEND_RECORD.SNF to a unique identifier.
Set HS_SEND_RECORD.RH to appropriate values (Figure 4-2 on page 4-8).
Set HS_SEND_RECORD.RU as specified in Appendix E.  Fields from the LULU_CB
 (e.g., addresses) are used in building this RU.

Send HS_SEND_RECORD to the control point via the CP-LU half-session.
```

## BUILD_AND_SEND_SESSST_RQ

| | |
|---|---|
| FUNCTION: | Build and send SESSST request to the control point (SSCP or PNCP) to indicate that a session has been activated. |
| INPUT: | LULU_CB |
| OUTPUT: | HS_SEND_RECORD (containing SESSST request) sent to CP-LU half-session |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LULU_CB | page A-5 |
| HS_SEND_RECORD | page A-16 |

```
Create HS_SEND_RECORD to contain SESSST request.
Set HS_SEND_RECORD.EFI to NORMAL.
Set HS_SEND_RECORD.SNF to a unique identifier.
Set HS_SEND_RECORD.RH to appropriate values (Figure 4-2 on page 4-8).
Set HS_SEND_RECORD.RU as specified in Appendix E.  Fields from the LULU_CB
 (e.g., addresses) are used in building this RU.

Send HS_SEND_RECORD to the control point via the CP-LU half-session.
```

## BUILD_AND_SEND_TERM_RQ

| | |
|---|---|
| FUNCTION: | Build and send TERM-SELF request to the control point (SSCP or PNCP). |
| INPUT: | LULU_CB, DEACTIVATE_SESSION.TYPE (type of TERM-SELF to send) |
| OUTPUT: | HS_SEND_RECORD (containing TERM-SELF request) sent to HS (CP-LU half-session) |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| HS_SEND_RECORD | page A-16 |
| LULU_CB | page A-5 |
| DEACTIVATE_SESSION | page A-31 |

```
Create HS_SEND_RECORD to contain the TERM-SELF request.
Set HS_SEND_RECORD.EFI to NORMAL.
Set HS_SEND_RECORD.SNF to a unique identifier.
Set HS_SEND_RECORD.RH to appropriate values (Figure 4-2 on page 4-8).
Set HS_SEND_RECORD.RU to appropriate values (see TERM-SELF request in Appendix E).
 The termination reason field is set according to the passed DEACTIVATE_SESSION.TYPE.
 Fields from the LULU_CB (e.g., URC from the INIT-SELF request) are used in building
 this RU.
Set HS_SEND_RECORD.DCF to the appropriate value.

Send HS_SEND_RECORD to the CP via the CP-LU half-session.
```

BUILD_AND_SEND_UNBIND_RQ

| | |
|---|---|
| FUNCTION: | Build and send an UNBIND request. |
| INPUT: | LULU_CB (indicates the LU-LU session to UNBIND), UNBIND type code, sense data (used for format-or-protocol-error type UNBINDs only) |
| OUTPUT: | UNBIND_RQ_SEND_RECORD sent to nodal NAU manager |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LULU_CB | page A-5 |
| UNBIND_RQ_SEND_RECORD | page A-19 |

```
Create UNBIND_RQ_SEND_RECORD to contain UNBIND request.
Set UNBIND_RQ_SEND_RECORD.LU_ID to this LU's identifier.
Set UNBIND_RQ_SEND_RECORD.PC_ID to LULU_CB.LU_LU.PC_ID (identifies path control
 through which the UNBIND will flow).
Set UNBIND_RQ_SEND_RECORD.ADDRESS to LULU_CB.LU_LU.ADDRESS (to be used in the TH
 address field).
Set UNBIND_RQ_SEND_RECORD.EFI to EXP (expedited-flow).
Set UNBIND_RQ_SEND_RECORD.SNF to a unique identifier.  Also, save this identifier
 in LULU_CB.SENT_UNBIND_RQ.SNF (used to correlate UNBIND response).
Set UNBIND_RQ_SEND_RECORD.DCF to the appropriate value.
Set UNBIND_RQ_SEND_RECORD.RH to the appropriate values (Figure 4-3 on page 4-16).
Set UNBIND_RQ_SEND_RECORD.RU to the appropriate values (see UNBIND request in Appendix E).
 The UNBIND Type field is set according to the passed UNBIND type code.  If the type is
 X'FE' (format or protocol error) then the passed sense data is included in the UNBIND RU.

Send UNBIND_RQ_SEND_RECORD to the other LU via the nodal NAU manager.
```

BUILD_AND_SEND_UNBIND_RSP

| | |
|---|---|
| FUNCTION: | Build and send an UNBIND response. |
| INPUT: | UNBIND_RQ_RCV_RECORD, LOCAL.SENSE_CODE (indicates what type of response [positive or negative] to build) |
| OUTPUT: | UNBIND_RSP_SEND_RECORD sent to nodal NAU manager |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 4-99 |
| UNBIND_RQ_RCV_RECORD | page A-23 |
| UNBIND_RSP_SEND_RECORD | page A-19 |

```
Create an UNBIND_RSP_SEND_RECORD.
Set UNBIND_RSP_SEND_RECORD.LU_ID to this LU's identifier.
Set UNBIND_RSP_SEND_RECORD.PC_ID to UNBIND_RQ_RCV_RECORD.PC_ID.
Set UNBIND_RSP_SEND_RECORD.ADDRESS to UNBIND_RQ_RCV_RECORD.ADDRESS.
Set UNBIND_RSP_SEND_RECORD.EFI to EXP.
Set UNBIND_RSP_SEND_RECORD.SNF to UNBIND_RQ_RCV_RECORD.SNF.
Initialize UNBIND_RSP_SEND_RECORD.RH to UNBIND_RQ_RCV_RECORD.RH.
Indicate response RH (RSP and byte 2 of RH set to all 0's).

If LOCAL.SENSE_CODE = X'00000000' then
   Set UNBIND_RSP_SEND_RECORD.RH to indicate a positive response (¬SD, POS).

Else
   Set UNBIND_RSP_SEND_RECORD.RH to indicate a negative response (SD, NEG).

Build the UNBIND RU, including the sense data if necessary.
Set UNBIND_RSP_SEND_RECORD.DCF to appropriate value.

Send UNBIND_RSP_SEND_RECORD to the other LU via the nodal NAU manager.
```

## BUILD_AND_SEND_UNBINDF_RQ

| | |
|---|---|
| FUNCTION: | Build and send UNBINDF request to the SSCP (for subarea nodes only). |
| INPUT: | Sense data (from UNBIND negative response), LULU_CB |
| OUTPUT: | HS_SEND_RECORD (containing UNBIND request) sent to HS (SSCP-LU half-session) |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LULU_CB | page A-5 |
| HS_SEND_RECORD | page A-16 |

```
  If this node is a subarea node and this LU is primary then (UNBINDF is sent only by
   primary LUs in subarea nodes)
     Create HS_SEND_RECORD to contain UNBINDF request.
     Set HS_SEND_RECORD.EFI to NORMAL.
     Set HS_SEND_RECORD.SNF to a unique value.
     Set HS_SEND_RECORD.RH to the appropriate values (Figure 4-2 on page 4-8).
     Set HS_SEND_RECORD.RU to the appropriate values (see UNBINDF request in Appendix E).
      Set sense data in the RU to the passed sense data.  Indicate UNBIND error in
      reaching SLU as the reason.  Fields from the passed LULU_CB (e.g., addresses)
      are used in building this RU.

     Send HS_SEND_RECORD to the CP via the CP-LU half-session.
```

## CINIT_RQ_STATE_ERROR

| | |
|---|---|
| FUNCTION: | Perform state error checking on received CINIT request. These checks are optional. |
| INPUT: | HS_RCV_RECORD (containing CINIT request), LULU_CB pointer (if null, indicates unsolicited CINIT; otherwise, indicates solicited CINIT) |
| OUTPUT: | TRUE if error; otherwise, FALSE. If TRUE, LOCAL.SENSE_CODE is set to appropriate sense code. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LU_MODE_SESSION_LIMIT_EXCEEDED | page 4-77 |
| LOCAL | page 4-99 |
| HS_RCV_RECORD | page A-11 |
| PARTNER_LU | page A-2 |
| MODE | page A-3 |
| LULU_CB | page A-5 |
| SESSION_TYPE | page 4-99 |

```
If the passed LULU_CB pointer contains a null value (indicating unsolicited CINIT) then
   If there are insufficient resources (e.g., storage) to start a new LU-LU session then
      Set LOCAL.SENSE_CODE to X'08120000' (insufficient resources).
      Return with a value of TRUE (error).

   If this LU cannot currently act as a primary LU then
      Set LOCAL.SENSE_CODE to X'083A0000' (LU not enabled).
      Return with a value of TRUE (error).

   Locate the PARTNER_LU control block in which the
    PARTNER_LU.FULLY_QUALIFIED_LU_NAME matches the
    SLU name in the CINIT request.
   If unable to locate the PARTNER_LU control block then
      Set LOCAL.SENSE_CODE to X'0835xxxx' (xxxx is the offset to SLU name).
      Return with a value of TRUE (error).

   Locate the MODE control block in which MODE.NAME matches the
    mode name in the X'0D' control vector of the CINIT request.
   If unable to locate the MODE then
      Set LOCAL.SENSE_CODE to X'0835xxxx' (xxxx is the offset to mode name
       in CINIT control vector X'0D').
      Return with a value of TRUE (error).

   Note: Unsolicited CINIT requests occur only when not using parallel sessions.
   The following determines whether the local LU will be the bidder or first
    speaker for the session so that the proper session limit checks can be made.
   If MODE.MIN_CONLOSERS_LIMIT = 1 then
      Set SESSION_TYPE to BIDDER.
   Else
      Set SESSION_TYPE to FIRST_SPEAKER.

   Call LU_MODE_SESSION_LIMIT_EXCEEDED(PARTNER_LU.FULLY_QUALIFIED_LU_NAME, MODE,
    SESSION_TYPE, ACTIVE_AND_PENDING_ACTIVE) (page 4-77).
   If the session limit will be exceeded then
      LOCAL.SENSE_CODE was set to the correct sense code by LU_MODE_SESSION_LIMIT_EXCEEDED.
      Return with a value of TRUE (error).
   Else
      Return with a value of FALSE (no error).

Else (solicited CINIT request)
   If LULU_CB.MODENAME ≠ the mode name in control vector X'0D' of the CINIT request
    then (The mode name must be the same as was sent in the INIT-SELF request.)
      Set LOCAL.SENSE_CODE to X'0835xxxx' (xxxx is the offset to mode name in CINIT
       control vector X'0D').
      Return with a value of TRUE (error).

   Locate the PARTNER_LU and MODE control blocks using the partner-LU name and
    mode name from the LULU_CB.  These control blocks will always be found for
    solicited CINIT requests.

   Call LU_MODE_SESSION_LIMIT_EXCEEDED(PARTNER_LU.FULLY_QUALIFIED_LU_NAME, MODE,
    LULU_CB.SESSION_TYPE, ACTIVE_AND_PENDING_ACTIVE) (page 4-77).
   If the session limit will be exceeded then
      LOCAL.SENSE_CODE was set to the correct sense data by LU_MODE_SESSION_LIMIT_EXCEEDED.
      Return with a value of TRUE (error).
   Else
      Return with a value of FALSE (no error).
```

## CLEANUP_LU_LU_SESSION

| | |
|---|---|
| FUNCTION: | Clean up LU-LU session. This may include sending a SESSEND request to the CP and a PC_HS_DISCONNECT record to path control. |
| INPUT: | LULU_CB of session to be cleaned up |
| OUTPUT: | LU-LU session cleaned up |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| BUILD_AND_SEND_SESSEND_RQ | page 4-69 |
| BUILD_AND_SEND_PC_HS_DISCONNECT | page 4-68 |
| LULU_CB | page A-5 |

```
If SESSST has been sent to the control point then
   Call BUILD_AND_SEND_SESSEND_RQ(LULU_CB) (page 4-69).

If PC_HS_CONNECT has been sent to path control then
   Call BUILD_AND_SEND_PC_HS_DISCONNECT(LULU_CB.LU_LU.HS_ID) (page 4-68).

Release any resources (e.g., buffers) held by this LU-LU session.
```

## 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, LULU_CB (to be initialized), CP_ID (control point identifier associated with this session) |
| OUTPUT: | LULU_CB (initialized) |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| CP_ID | page A-2 |
| ACTIVATE_SESSION | page A-31 |
| LULU_CB | page A-5 |
| PARTNER_LU | page A-2 |

```
Set LULU_CB.CP_ID to passed control point identifier (CP_ID).

Determine whether the local LU is to indicate the primary or secondary role for
 itself in INIT-SELF. An LU in a peripheral node indicates primary for
 PNCP-mediated sessions; otherwise, it indicates secondary. An LU in a subarea
 node indicates primary whenever it is capable of acting as a primary;
 otherwise, it indicates secondary.
Set LULU_CB.HALF_SESSION_TYPE to PRI or SEC as determined above.
Set LULU_CB.CP_LU.HS_ID to the identifier of the CP-LU half-session.
Set LULU_CB.CORRELATOR to ACTIVATE_SESSION.CORRELATOR.

Locate the PARTNER_LU control block using ACTIVATE_SESSION.LU_NAME.

Set LULU_CB.LUNAME.FQ to PARTNER_LU.FULLY_QUALIFIED_LU_NAME.
Set LULU_CB.LUNAME.LOCAL to ACTIVATE_SESSION.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 an unsolicited BIND request. |
| INPUT: | BIND_RQ_RCV_RECORD, LULU_CB (to be initialized) |
| OUTPUT: | LULU_CB (initialized) |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| BIND_RQ_RCV_RECORD | page A-21 |
| PARTNER_LU | page A-2 |
| MODE | page A-3 |
| LULU_CB | page A-5 |

```
Set the identifier (LULU_CB.CP_ID) of the control point mediating this LU-LU
 session.  The mediating control point for peripheral nodes is identified by
 either the adjacent link station (ALS) for an SSCP or a special identifier
 for the PNCP.  The control point for subarea nodes is identified by its
 address.

Set the CP-LU half-session identifier (LULU_CB.CP_LU.HS_ID)
 (set to a null value if control point does not have an active session with
 this LU).

Locate the partner-LU control block (PARTNER_LU) using the user-data PLU name
 in BIND.

Set LULU_CB.LUNAME.LOCAL to PARTNER_LU.LOCAL_LU_NAME.
Set LULU_CB.LUNAME.FQ to user-data PLU name in BIND.
Set LULU_CB.MODENAME to user-data mode name in BIND.
Set LULU_CB.HALF_SESSION_TYPE to SEC (BIND receiver is secondary).

If parallel sessions are supported with the partner LU then
   If BIND specifies secondary as contention winner then
      Set LULU_CB.SESSION_TYPE to FIRST_SPEAKER.
   Else
      Set LULU_CB.SESSION_TYPE to BIDDER.

Else (parallel sessions not supported with the partner LU)
   If MODE.MIN_CONWINNERS_LIMIT = 1 then
      Set LULU_CB.SESSION_TYPE to FIRST_SPEAKER.
   Else
      Set LULU_CB.SESSION_TYPE to BIDDER.
```

## INITIALIZE_LULU_CB_CINIT

| | |
|---|---|
| FUNCTION: | Initialize an LULU_CB for an LU-LU session being activated as a result of receiving an unsolicited CINIT request. |
| INPUT: | HS_RCV_RECORD (containing CINIT request), LULU_CB (to be initialized) |
| OUTPUT: | LULU_CB (initialized) |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LULU_CB | page A-5 |
| PARTNER_LU | page A-2 |
| CPLU_CB | page A-1 |
| MODE | page A-3 |
| HS_RCV_RECORD | page A-11 |

```
Locate the CP-LU control block (CPLU_CB) using the half-session identifier
 (HS_ID) from HS_RCV_RECORD.  It will always be found.
Set LULU_CB.CP_ID to CPLU_CB.CP_ID (control point identifier).
Set LULU_CB.CP_LU.HS_ID to CPLU_CB.HS_ID (CP-LU half-session identifier).
Set LULU_CB.HALF_SESSION_TYPE to PRI (CINIT receiver is always primary).

Locate the PARTNER_LU control block using the SLU name in the CINIT request
 as a search argument.  It will always be found.
Set LULU_CB.LUNAME.LOCAL to PARTNER_LU.LOCAL_LU_NAME.
Set LULU_CB.LUNAME.FQ to the SLU name from CINIT.
Set LULU_CB.MODENAME to the mode name in control vector X'0D' of CINIT.

Note:  The CINIT request can be received only if parallel sessions are not supported.
Locate the MODE control block using LULU_CB.MODENAME
 as a search argument.  It will always be found.

If MODE.MIN_CONLOSERS_LIMIT = 1 then
   Set LULU_CB.SESSION_TYPE to BIDDER (the local LU is bidder).

Else
   Set LULU_CB.SESSION_TYPE to FIRST_SPEAKER (the local LU is first speaker).
```

## 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 conditions. |
| NOTE: | If parallel sessions are not supported with the partner LU and the total session limit will not be exceeded, then a session-activation request specifying this LU as first speaker is accepted. For example, a BIND request is received specifying the SEC as first speaker (contention winner). The SEC LU 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 modename specified in the BIND). Even though the MIN_CONWINNERS_LIMIT of 0 will be exceeded, the BIND is accepted. |
| INPUT: | PARTNER_LU.FULLY_QUALIFIED_LU_NAME, MODE, session type (in SESSION_TYPE, ACTIVATE_SESSION.SESSION_TYPE, or LULU_CB.SESSION_TYPE—FIRST_SPEAKER or BIDDER), state (ACTIVE or ACTIVE_AND_PENDING_ACTIVE) |
| OUTPUT: | TRUE if session limits exceeded; otherwise, FALSE. If TRUE, LOCAL.SENSE_CODE is set to appropriate sense data. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 4-99 |
| 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.

Else
   Set BIDDER_SESSION_COUNT to the number of active and pending active
    bidder sessions.
   Set FSP_SESSION_COUNT to the number of active and pending active first
    speaker sessions.

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 in prologue)
      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).
```

## PROCESS_ABORT_HS

| | |
|---|---|
| FUNCTION: | Process an ABORT_HS record received from LU-LU half-session. |
| INPUT: | ABORT_HS record |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_STATUS | page 4-92 |
| LOCAL | page 4-99 |
| ABORT_HS | page A-11 |
| LULU_CB | page A-5 |

```
Determine which LU-LU session is being aborted by searching through the LU-LU
 control block list (LOCAL.LULU_CB_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 4-92).
```

## PROCESS_ACTIVATE_SESSION

| | |
|---|---|
| FUNCTION: | Process an ACTIVATE_SESSION record received from RM. |
| INPUT: | ACTIVATE_SESSION record |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| ACTIVATE_SESSION_ERROR | page 4-51 |
| BUILD_AND_SEND_ACT_SESS_RSP_NEG | page 4-57 |
| INITIALIZE_LULU_CB_ACT_SESS | page 4-74 |
| FSM_STATUS | page 4-92 |
| ACTIVATE_SESSION | page A-31 |
| LULU_CB | page A-5 |
| CP_ID | page A-2 |
| ERROR_TYPE | page 4-99 |

```
Call ACTIVATE_SESSION_ERROR(ACTIVATE_SESSION, ERROR_TYPE, CP_ID) (page 4-51).
If there is an error then (ERROR_TYPE is returned if error)
   Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(ACTIVATE_SESSION.CORRELATOR, ERROR_TYPE)
    (page 4-57).

Else (control point identifier [CP_ID] is returned if no error)
   Create an LU-LU control block (LULU_CB) and initialize its fields.
   Call INITIALIZE_LULU_CB_ACT_SESS(ACTIVATE_SESSION, LULU_CB, CP_ID) (page 4-74).
   Call FSM_STATUS(ACTIVATE_SESSION record, LULU_CB) (page 4-92).
```

## PROCESS_ACTLU_RQ

| | |
|---|---|
| FUNCTION: | Process a received ACTLU request. |
| INPUT: | ACTLU_RQ_RCV_RECORD |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| BUILD_AND_SEND_PC_HS_CONNECT | page 4-67 |
| BUILD_AND_SEND_ACTLU_RSP_NEG | page 4-58 |
| BUILD_AND_SEND_ACTLU_RSP_POS | page 4-59 |
| LOCAL | page 4-99 |
| ACTLU_RQ_RCV_RECORD | page A-21 |
| CPLU_CB | page A-1 |

```
Optionally check the ACTLU request for format errors.  This includes checking the
 RH (see Figure 4-3 on page 4-16 for correct format of RH) and RU (Appendix E) for
 syntax errors.  Also, if the FM profile of ACTLU is specified as 6, and this LU
 does not support 6, then it is an error.
If there is a format error then
   Set LOCAL.SENSE_CODE to the appropriate value (Appendix G).
   Call BUILD_AND_SEND_ACTLU_RSP_NEG(ACTLU_RQ_RCV_RECORD) (page 4-58) to
    send a negative response to ACTLU.

Else (no format error)
   Determine if a CP-LU session already exists (search for a CP-LU half-session
    control block (CPLU_CB) with a CP_ID the same as ACTLU_RQ_RCV_RECORD.CP_ID).
   If a CP-LU session already exists then
      Set LOCAL.SENSE_CODE to X'08150000' (function already active).
      Call BUILD_AND_SEND_ACTLU_RSP_NEG(ACTLU_RQ_RCV_RECORD) (page 4-58) to
       send a negative response to ACTLU.

   Else (CP-LU session does not already exist)
      If resources (e.g., storage) are not available to create a new CP-LU
       session then
         Set LOCAL.SENSE_CODE to X'08120000' (insufficient resources).
         Call BUILD_AND_SEND_ACTLU_RSP_NEG(ACTLU_RQ_RCV_RECORD) (page 4-58).

      Else (resources are available)
         Create a CPLU_CB to represent a new CP-LU session and insert it
          in LOCAL.CPLU_CB_LIST.
         Copy the CP_ID and PC_ID fields into the CPLU_CB from the ACTLU_RQ_RCV_RECORD.
         Set CPLU_CB.HS_ID to a unique value for the CP-LU half-session process.
         Call BUILD_AND_SEND_PC_HS_CONNECT(CPLU_CB.PC_ID, CPLU_CB.HS_ID,
          ACTLU_RQ_RCV_RECORD.ADDRESS) (page 4-67) to indicate to path
          control that a new half-session has been activated.

         Call BUILD_AND_SEND_ACTLU_RSP_POS(ACTLU_RQ_RCV_RECORD) (page 4-59)
          to send a positive response to ACTLU and create the CP-LU half-session.
```

## PROCESS_BIND_RQ

| | |
|---|---|
| FUNCTION: | Process a received BIND request. |
| INPUT: | BIND_RQ_RCV_RECORD |
| NOTE: | It is possible for a BIND containing a URC to be received with no matching LULU_CB. This can occur when session outage (DACTLU-SON) occurs on the CP-LU session after the INIT has been sent but before the BIND is received. In this case, the BIND is accepted even though there is currently no active CP-LU session. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| BUILD_AND_SEND_BIND_RSP_NEG | page 4-60 |
| BIND_RQ_STATE_ERROR | page 4-52 |
| INITIALIZE_LULU_CB_BIND | page 4-75 |
| FSM_STATUS | page 4-92 |
| LOCAL | page 4-99 |
| BIND_RQ_RCV_RECORD | page A-21 |
| LULU_CB | page A-5 |

```
Check BIND request for basic syntax errors that would inhibit further processing
 of the BIND, including errors in the RH, the TH DCF and the BIND length fields
 (see Appendix E and Figure 4-3 on page 4-16).
 Syntax errors are format errors and, as such, are state-independent.  When a syntax
 error is found, LOCAL.SENSE_CODE is set to the appropriate sense data (X'1002' for
 overall length errors and X'0835' with offset for individual length field errors).
 Syntax error checking is required.

If a syntax error exists then
   Call BUILD_AND_SEND_BIND_RSP_NEG(BIND_RQ_RCV_RECORD) (page 4-60).
   Optionally log the error.

Else (no syntax error)
   Determine if a BIND request is solicited.
    A solicited BIND is one that the local LU solicited by having previously sent
    an INIT-SELF.  The LULU_CBs are searched for a match on either the ADDRESS
    field in BIND_RQ_RCV_RECORD (or ADDRESS and PC_ID for peripheral nodes)
    or the URC field of the BIND RU.  If a match is found on either field, the
    BIND is considered solicited.

   If the BIND is solicited then
      Optionally check BIND for semantic errors (Appendix E) and if an error exists,
       set LOCAL.SENSE_CODE with sense data reflecting error.  Semantic errors are
       field content errors (e.g., a field does not contain an allowable value).  Like
       syntax errors, these errors are format errors and are state-independent.
       If a semantic error is found, LOCAL.SENSE_CODE is set to the sense data X'0835'
       with the offset to the field in error.

      Call BIND_RQ_STATE_ERROR(BIND_RQ_RCV_RECORD) (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.

      Call FSM_STATUS(BIND_RQ_RCV_RECORD, LULU_CB) (page 4-92).

   Else (BIND is unsolicited--session was not initiated by this LU (see Note in prologue))
      Check the BIND for semantic and state errors as described above.
      If an error exists then
         Call BUILD_AND_SEND_BIND_RSP_NEG(BIND_RQ_RCV_RECORD) (page 4-60).

      Else
         Create an LU-LU half-session control block (LULU_CB) and initialize its fields.
         Call INITIALIZE_LULU_CB_BIND(BIND_RQ_RCV_RECORD, LULU_CB) (page 4-75).
         Call FSM_STATUS(BIND_RQ_RCV_RECORD, LULU_CB) (page 4-92).
```

## PROCESS_BIND_RSP

| | |
|---|---|
| FUNCTION: | Process a received BIND response. |
| INPUT: | BIND_RSP_RCV_RECORD |
| NOTE: | The LOCAL.SENSE_CODE is not always set by the BIND response error checking procedures. FSM_STATUS determines whether or not an error has occurred by checking for a nonzero value in the LOCAL.SENSE_CODE field. Therefore, the LOCAL.SENSE_CODE is set to a dummy nonzero value X'FFFFFFFF'. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| BIND_RSP_STATE_ERROR | page 4-53 |
| FSM_STATUS | page 4-92 |
| LOCAL | page 4-99 |
| BIND_RSP_RCV_RECORD | page A-22 |
| LULU_CB | page A-5 |

```
Attempt to correlate the BIND response with a previously sent BIND request.
 A search is made for an LULU_CB  in which LULU_CB.PC_ID =
 BIND_RSP_RCV_RECORD.PC_ID and LULU_CB.SENT_BIND_RQ.SNF = BIND_RSP_RCV_RECORD.SNF.
If the correlation is successful then (an LULU_CB has been found)
   Check the BIND response for basic syntax errors that would inhibit further
    processing, including errors in the RH, the TH DCF and the BIND length
    fields (see Appendix E and Figure 4-3 on page 4-16).  Syntax errors are format
    errors and, as such, are state-independent.  These error checks are required.

   Optionally check the BIND response for semantic errors (Appendix E).  Semantic
    errors are field content errors (e.g., a field does not contain an allowable
    value).  Like syntax errors, these errors are format errors and are state-
    independent.

   Optionally call BIND_RSP_STATE_ERROR(BIND_RSP_RCV_RECORD, LULU_CB) (page 4-53)
    to check for state errors.

   If either a syntax, semantic, or state error is detected then
      Set LOCAL.SENSE_CODE to the value X'FFFFFFFF' (see Note in prologue).

   Call FSM_STATUS(BIND_RSP_RCV_RECORD, LULU_CB) (page 4-92).

Else (unable to correlate the BIND response)
   Set LOCAL.SENSE_CODE to X'200E0000' (response correlation error).
   Optionally log the error.
```

## PROCESS_CINIT_RQ

| | |
|---|---|
| FUNCTION: | Process a received CINIT request. |
| INPUT: | HS_RCV_RECORD containing CINIT request |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| BUILD_AND_SEND_CINIT_RSP | page 4-62 |
| CINIT_RQ_STATE_ERROR | page 4-72 |
| INITIALIZE_LULU_CB_CINIT | page 4-76 |
| FSM_STATUS | page 4-92 |
| LOCAL | page 4-99 |
| HS_RCV_RECORD | page A-11 |
| LULU_CB | page A-5 |

```
Note: Peripheral nodes receive CINIT from the PNCP only when initiating a session
 with a peer T2.1 LU (i.e., the session is PNCP-mediated).
If the local node is a peripheral node and the CINIT request has been received from
 an SSCP (as opposed to a PNCP) then
   Set LOCAL.SENSE_CODE to X'10030000' (function not supported).
   Call BUILD_AND_SEND_CINIT_RSP(HS_RCV_RECORD) (page 4-62)
    to send a negative response to CINIT.
   Optionally log the error.

Else
   Optionally check the CINIT request for syntax errors.  This includes checking
    the TH DCF field and length fields within the CINIT RU.  If the DCF is
    incorrect, sense data X'10020000' is used; otherwise, X'0835xxxx' is used
    (xxxx is the offset to the field in error).  An additional check is made to
    determine whether the URC field (within the BIND image in CINIT) is present
    if required.  See CINIT request in Appendix E for correct format.
   If there is a syntax error then
      Set LOCAL.SENSE_CODE to appropriate value.
      Call BUILD_AND_SEND_CINIT_RSP(HS_RCV_RECORD) (page 4-62)
       to send a negative response to CINIT.
      Optionally log the error.

   Else (no syntax error)
      If the CINIT request indicates either third-party-initiated (INITIATE origin
       specifies ILU is not OLU) or secondary-LU-initiated (SLU is OLU) then
       (unsolicited CINIT processing)
         Optionally check the CINIT request for semantic errors.  This includes checking
          that the proper session keys and control vectors are included.  The sense
          data X'0835xxxx' is used to indicate fields in error (xxxx is the offset to the
          field in error).  See CINIT request in Appendix E for correct RU values.
          Optionally perform CINIT state checks by calling CINIT_RQ_STATE_ERROR
          (HS_RCV_RECORD, LULU_CB pointer) (page 4-72).  If any errors are
          found LOCAL.SENSE_CODE is set to the appropriate sense data.
         If there is a semantic or state error then
            Call BUILD_AND_SEND_CINIT_RSP(HS_RCV_RECORD) (page 4-62)
             to send a negative response to CINIT.

         Else (no errors)
            Create and initialize an LU-LU half-session control block (LULU_CB).
            Call INITIALIZE_LULU_CB_CINIT(HS_RCV_RECORD, LULU_CB) (page 4-76).
            Call FSM_STATUS(HS_RCV_RECORD, LULU_CB) (page 4-92).

      Else (not unsolicited CINIT)
         Attempt to correlate this CINIT request to a previously sent INIT-SELF
          request to the same CP.  Search for an LU-LU half-session control
          block (LULU_CB) where LULU_CB.SENT_INITIATE_RQ.URC = the URC field in
          the BIND image of the CINIT request.
         If the CINIT request is correlated successfully then (solicited CINIT
          processing)
            Check for CINIT request semantic and state errors as described above.
             If an error is found, LOCAL.SENSE_CODE is set.
            Call FSM_STATUS(HS_RCV_RECORD, LULU_CB) (page 4-92).

         Else (unable to correlate CINIT)
            Set LOCAL.SENSE_CODE to X'081E0000' (session reference error).
            Call BUILD_AND_SEND_CINIT_RSP(HS_RCV_RECORD) (page 4-62)
             to send a negative response to CINIT.
            Optionally log the error.
```

## PROCESS_CLEANUP_RQ

| | |
|---|---|
| FUNCTION: | Process a CLEANUP request received by a subarea LU. |
| INPUT: | HS_RCV_RECORD containing CLEANUP request |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| BUILD_AND_SEND_RSP_OR_LOG | page 4-66 |
| FSM_STATUS | page 4-92 |
| LOCAL | page 4-99 |
| LULU_CB | page A-5 |
| HS_RCV_RECORD | page A-11 |

```
Optionally check the CLEANUP request for format errors.  This includes
 checking the TH DCF field for RU length errors (X'10020000'), checking for
 format 0 (X'10030000'), and checking for valid session keys (X'0835xxxx').
 See CLEANUP request in Appendix E for correct format.
If there is a format error then
   Set LOCAL.SENSE_CODE to the appropriate value.
   Call BUILD_AND_SEND_RSP_OR_LOG(HS_RCV_RECORD) (page 4-66)
    to send a negative response to CLEANUP.

Else (no format error)
   Determine the LU-LU session (mediated by the CP that sent the CLEANUP)
    to be cleaned up by searching for an LU-LU half-session control
    block (LULU_CB) that has an address pair (LULU_CB.ADDRESS)
    matching the address pair specified in the CLEANUP RU.  (The addresses
    of the address pair in CLEANUP may be specified in any order.)
   If an LULU_CB is found then
      Call BUILD_AND_SEND_RSP_OR_LOG(HS_RCV_RECORD) (page 4-66)
       to send a positive response to CLEANUP.
      Call FSM_STATUS(HS_RCV_RECORD, LULU_CB) (page 4-92).

   Else (unable to determine which LU-LU session to clean up)
      Set LOCAL.SENSE_CODE to X'081E0000' (session reference error).
      Call BUILD_AND_SEND_RSP_OR_LOG(HS_RCV_RECORD) (page 4-66)
       to send a negative response to CLEANUP.
```

## PROCESS_CTERM_RQ

| | |
|---|---|
| FUNCTION: | Process a CTERM request received by a subarea LU. |
| INPUT: | HS_RCV_RECORD containing CTERM request |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| BUILD_AND_SEND_RSP_OR_LOG | page 4-66 |
| FSM_STATUS | page 4-92 |
| LOCAL | page 4-99 |
| LULU_CB | page A-5 |
| HS_RCV_RECORD | page A-11 |

```
Optionally check the CTERM for format errors.  This includes checking
 the TH DCF field (X'10020000'), the Format and Type fields (X'10030000'),
 and the Session Key field (X'0835xxxx'). See CTERM request in Appendix E
 for correct format.
If there is a format error then
   Call BUILD_AND_SEND_RSP_OR_LOG(HS_RCV_RECORD) (page 4-66)
    to send a negative response to CTERM.

Else (no format error)
   Determine the LU-LU session (mediated by the CP that sent the CTERM)
    to be cleaned up by searching for an LU-LU half-session control
    block (LULU_CB) that has an address pair (LULU_CB.ADDRESS)
    matching the address pair specified in the CTERM RU.  (The addresses
    of the address pair in CTERM may be specified in any order.)
   If an LULU_CB if found then
      Call BUILD_AND_SEND_RSP_OR_LOG(HS_RCV_RECORD) (page 4-66)
       to send a positive response to CTERM.
      Call FSM_STATUS(HS_RCV_RECORD, LULU_CB) (page 4-92).

   Else (unable to determine which LU-LU session to clean up)
      Set LOCAL.SENSE_CODE to X'081E0000' (session reference error).
      Call BUILD_AND_SEND_RSP_OR_LOG(HS_RCV_RECORD) (page 4-66)
       to send a negative response to CTERM.
```

## PROCESS_DACTLU_RQ

| | |
|---|---|
| FUNCTION: | Process a received DACTLU request. |
| INPUT: | DACTLU_RQ_RCV_RECORD |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| BUILD_AND_SEND_DACTLU_RSP | page 4-63 |
| BUILD_AND_SEND_PC_HS_DISCONNECT | page 4-68 |
| FSM_STATUS | page 4-92 |
| LOCAL | page 4-99 |
| DACTLU_RQ_RCV_RECORD | page A-22 |
| CPLU_CB | page A-1 |

```
Optionally check the DACTLU request for RH format errors (see Figure 4-3 on page 4-16 for
 correct RH format).
If an error is found then
   Set LOCAL.SENSE_CODE to the appropriate value (Appendix G).
   Call BUILD_AND_SEND_DACTLU_RSP(DACTLU_RQ_RCV_RECORD) (page 4-63)
    to send a negative response to the DACTLU request.

Else (no error found)
   Call BUILD_AND_SEND_DACTLU_RSP(DACTLU_RQ_RCV_RECORD) (page 4-63)
    to send a positive response to the DACTLU request.
   Determine if a CP-LU session is active by searching for a CPLU_CB (in
    LOCAL.CPLU_CB_LIST) to determine whether DACTLU_RQ_RCV_RECORD and CPLU_CB have
    matching control point identifiers.
   If a CP-LU session is active then
      If the DACTLU RU length is less than 3 or the DACTLU deactivation type is
       normal then
         Reset all active and pending active LU-LU sessions mediated by this
          CP by calling FSM_STATUS (page 4-92) with a
          RESET_NORMAL signal for each LU-LU session.

      Else (must be DACTLU with type SON)
         Reset all LU-LU sessions that have not become active or pending active by
          calling FSM_STATUS (page 4-92) with a RESET_SON
          signal for each LU-LU session.

      Destroy the CP-LU half-session process.
      Call BUILD_AND_SEND_PC_HS_DISCONNECT(CPLU_CB.HS_ID) (page 4-68)
       to notify path control that a half-session has been deactivated.
```

## PROCESS_DEACTIVATE_SESSION

| | |
|---|---|
| FUNCTION: | Process a DEACTIVATE_SESSION record received from RM. |
| INPUT: | DEACTIVATE_SESSION record |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_STATUS | page 4-92 |
| DEACTIVATE_SESSION | page A-31 |
| LULU_CB | page A-5 |

```
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 4-92).
```

## PROCESS_ECHOTEST_RQ

| | |
|---|---|
| FUNCTION: | Process a received ECHOTEST request in an implementation-defined way. |
| INPUT: | HS_RCV_RECORD containing ECHOTEST request |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| HS_RCV_RECORD | page A-11 |

See page 4-31.

## PROCESS_HIERARCHICAL_RESET

| | |
|---|---|
| FUNCTION: | Process a HIERARCHICAL_RESET record received from the nodal NAU manager. This record is generated as a result of a DACTPU. |
| INPUT: | HIERARCHICAL_RESET record |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| BUILD_AND_SEND_HIER_RESET_RSP | page 4-64 |
| FSM_STATUS | page 4-92 |
| HIERARCHICAL_RESET | page A-22 |
| CPLU_CB | page A-1 |

```
Attempt to locate the CP-LU session control block by searching for a CPLU_CB
 with a control point identifier matching that in the HIERARCHICAL_RESET record.
If a CPLU_CB is located then
   Reset all LU-LU sessions mediated by this CP by calling FSM_STATUS
    (page 4-92) with a RESET_NORMAL signal for each LU-LU session.
   Destroy the CP-LU half-session.

Call BUILD_AND_SEND_HIER_RESET_RSP(HIERARCHICAL_RESET) (page 4-64).
```

PROCESS_INIT_HS_RSP

| | |
|---|---|
| FUNCTION: | Process an INIT_HS_RSP record received from an LU-LU half-session. |
| INPUT: | INIT_HS_RSP record |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_STATUS | page 4-92 |
| INIT_HS_RSP | page A-11 |
| LULU_CB | page A-5 |

```
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 4-92).
```

PROCESS_INIT_SELF_RSP

| | |
|---|---|
| FUNCTION: | Process a received INIT-SELF response. |
| INPUT: | HS_RCV_RECORD containing INIT-SELF response |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_STATUS | page 4-92 |
| HS_RCV_RECORD | page A-11 |
| LULU_CB | page A-5 |

```
Attempt to correlate the INIT-SELF response with a sent INIT-SELF request.
 Search for an LU-LU control block (LULU_CB) where LULU_CB.CP_LU.HS_ID =
 HS_RCV_RECORD.HS_ID and LULU_CB.SENT_INITIATE_RQ.SNF = HS_RCV_RECORD.SNF.
If the response is correlated successfully then
   Call FSM_STATUS(HS_RCV_RECORD, LULU_CB) (page 4-92).

Else
   Optionally log the error using sense data X'200E0000'.
```

## PROCESS_NOTIFY_RQ

| | |
|---|---|
| FUNCTION: | Process a received NOTIFY request. |
| INPUT: | HS_RCV_RECORD containing NOTIFY request |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| BUILD_AND_SEND_RSP_OR_LOG | page 4-66 |
| FSM_STATUS | page 4-92 |
| LOCAL | page 4-99 |
| HS_RCV_RECORD | page A-11 |
| LULU_CB | page A-5 |

```
Optionally check the NOTIFY request for format errors.  This includes
 checking the TH DCF and length fields (X'10020000'), the vector type
 ('0835xxxx'), and the session keys (X'0835xxxx') (where xxxx is an offset
 in each case).  See NOTIFY request in Appendix E for correct format.
If there is a format error then
   Set LOCAL.SENSE_CODE to the appropriate value.
   Call BUILD_AND_SEND_RSP_OR_LOG(HS_RCV_RECORD) (page 4-66)
    to send a negative response to NOTIFY.

Else (no format error)
   Attempt to correlate this NOTIFY request with a previously sent INIT-SELF
    request (for the same CP).  Search for an LU-LU half-session control block
    (LULU_CB) where LULU_CB.SENT_INITIATE_RQ.URC matches the URC field in the
    NOTIFY request RU.
   If an LULU_CB is found then
      Call FSM_STATUS(HS_RCV_RECORD, LULU_CB) (page 4-92).

   Else (unable to correlate the NOTIFY request)
      Set LOCAL.SENSE_CODE to X'081E0000' (session reference error).
      Call BUILD_AND_SEND_RSP_OR_LOG(HS_RCV_RECORD) (page 4-66)
       to send a negative response to NOTIFY.
```

## PROCESS_NOTIFY_RSP

| | |
|---|---|
| FUNCTION: | Process a received NOTIFY response. |
| INPUT: | HS_RCV_RECORD containing received NOTIFY response. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| HS_RCV_RECORD | page A-11 |

See page 4-13 for a general discussion; details are not formally defined.

## PROCESS_PC_CONNECT_RSP

| | |
|---|---|
| FUNCTION: | Process a path control connect response (PC_CONNECT_RSP) received from NNM. |
| INPUT: | PC_CONNECT_RSP |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_STATUS | page 4-92 |
| LULU_CB | page A-5 |
| PC_CONNECT_RSP | page A-22 |

```
Attempt to locate the LU-LU half-session control block (LULU_CB) in which the
 half-session identifier (HS_ID) matches that in the PC_CONNECT_RSP record.
If an LULU_CB is located then
  Call FSM_STATUS(PC_CONNECT_RSP, LULU_CB) (page 4-92).
```

## PROCESS_REQECHO_RSP

| | |
|---|---|
| FUNCTION: | Process a received REQECHO response in an implementation-defined way. |
| INPUT: | HS_RCV_RECORD containing received REQECHO response |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| HS_RCV_RECORD | page A-11 |

See page 4-31.

## PROCESS_SESSION_ROUTE_INOP

| | |
|---|---|
| FUNCTION: | Process a SESSION_ROUTE_INOP record received from NNM. |
| INPUT: | SESSION_ROUTE_INOP |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_STATUS | page 4-92 |
| SESSION_ROUTE_INOP | page A-23 |
| LULU_CB | page A-5 |
| CPLU_CB | page A-1 |

```
Reset all CP-LU sessions that are using the path control process that failed.
 This is done by locating all the CP-LU session control blocks (CPLU_CBs)
 that have a path control identifier (PC_ID) matching that of the path
 control process that failed.  Each CPLU_CB located is then destroyed.

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,
 FSM_STATUS (page 4-92) is called with a RESET_NORMAL signal
 to reset that session.
```

## PROCESS_TERM_SELF_RSP

| | |
|---|---|
| FUNCTION: | Process a received TERM-SELF response. Nothing is done for a TERM-SELF response because the LU-LU session awareness is cleaned up when the TERM-SELF request is sent. The TERM-SELF response is simply discarded. |
| INPUT: | HS_RCV_RECORD containing TERM-SELF response |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| HS_RCV_RECORD | page A-11 |

No processing is done for a TERM-SELF response.

## PROCESS_UNBIND_RQ

| | |
|---|---|
| FUNCTION: | Process a received UNBIND request. |
| INPUT: | UNBIND_RQ_RCV_RECORD |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| BUILD_AND_SEND_UNBIND_RSP | page 4-71 |
| FSM_STATUS | page 4-92 |
| LOCAL | page 4-99 |
| UNBIND_RQ_RCV_RECORD | page A-23 |
| LULU_CB | page A-5 |

```
Optionally check the UNBIND request for syntax errors.  Syntax errors include checking
 the RH (see Figure 4-3 on page 4-16 for correct format) and checking the length
 (DCF) for being too short (see UNBIND request in Appendix E).
If there is a syntax error then
   Set LOCAL.SENSE_CODE to appropriate value.
   Call BUILD_AND_SEND_UNBIND_RSP(UNBIND_RQ_RCV_RECORD)
    (page 4-71) to send a negative response to UNBIND.
   Optionally log the error.

Else (no syntax error)
   Attempt to correlate the UNBIND with an existing LU-LU session by locating an LULU_CB
    where LULU_CB.LU_LU.PC_ID = UNBIND_RQ_RCV_RECORD.PC_ID and
    LULU_CB.LU_LU.ADDRESS = UNBIND_RQ_RCV_RECORD.ADDRESS.
   If the UNBIND correlates to an existing session (an LULU_CB is found) then
      Call FSM_STATUS(UNBIND_RQ_RCV_RECORD, LULU_CB) (page 4-92).

   Else (UNBIND does not correlate to an existing session)
      Call BUILD_AND_SEND_UNBIND_RSP(UNBIND_RQ_RCV_RECORD)
       (page 4-71) to send positive response to UNBIND.
```

PROCESS_UNBIND_RSP

| | |
|---|---|
| FUNCTION: | Process a received UNBIND response. |
| INPUT: | UNBIND_RSP_RCV_RECORD |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_STATUS | page 4-92 |
| LULU_CB | page A-5 |
| UNBIND_RSP_RCV_RECORD | page A-23 |

```
Optionally check the UNBIND response RH for format errors
 (see Figure 4-3 on page 4-16 for correct format of RH).
If there is an RH error then
   Optionally log the error.

Else (no RH error)
   Correlate this UNBIND response with a sent UNBIND request.  Search for an
    LU-LU half-session control block (LULU_CB) where LULU_CB.LU_LU.PC_ID =
    UNBIND_RSP_RCV_RECORD.PC_ID and LULU_CB.SENT_UNBIND_RQ.SNF =
    UNBIND_RSP_RCV_RECORD.SNF.
   If the UNBIND response is correlated successfully then
      Call FSM_STATUS(UNBIND_RSP_RCV_RECORD, LULU_CB) (page 4-92).

   Else (unable to correlate UNBIND response)
      Optionally log the error with sense data X'200E0000' (response correlation error).
```

FSM_STATUS

FUNCTION: This FSM maintains the state of an LU-LU session from initiation through termination. State name abbreviations are as follows:

- RES = reset
- PND INI RSP PLU = pending receipt of INIT-SELF response where INIT-SELF was sent specifying this LU as PLU
- PND INI RSP SLU = pending receipt of INIT-SELF response where INIT-SELF was sent specifying this LU as SLU
- PND CIN = pending receipt of CINIT request
- PND BIN = pending receipt of BIND request
- PND PC RSP THI = pending receipt of PC_CONNECT_RSP for a session this LU initiated (sent INIT-SELF for)
- PND BIN RSP THI = pending receipt of a BIND response for a session this LU initiated (sent INIT-SELF for)
- PND INI HS RSP THI = pending receipt of INIT_HS_RSP for a session this LU initiated (sent INIT-SELF for)
- PND PC RSP OTH = pending receipt of PC_CONNECT_RSP for a session the other LU initiated (sent INIT-SELF for)
- PND BIN RSP OTH = pending receipt of a BIND response for a session the other LU initiated (sent INIT-SELF for)
- PND INI HS RSP OTH = pending receipt of INIT_HS_RSP for a session the other LU initiated (sent INIT-SELF for)
- ACT = active
- PND UNB RSP = pending UNBIND response

The state of the LU-LU session may be considered "active" or "pending active." States 8, 11, 12, and 13 are considered "active." States 6, 7, 9, and 10 are considered "pending active." All other states are considered neither "active" nor "pending active."

NOTE: Error type is "retry" if the first 2 bytes of LOCAL.SENSE_CODE are 0812 or 0805; otherwise, error type is "no retry."

INPUT: The record to be processed and the LU-LU half-session control block (LULU_CB). The input record is used as an input to this FSM. These inputs denote RUs (Appendix E), interprocess records (i.e., from HS, RM, or NNM [Appendix A]), results of earlier sense data settings (LOCAL.SENSE_CODE set to NG or OK), and session roles (PLU or SLU) of the local LU (as ILU).

**Referenced procedures, FSMs, and data structures:**

| | |
|---|---|
| BUILD_AND_SEND_DEACTIVATE_SESS | page 4-63 |
| BUILD_AND_SEND_UNBINDF_RQ | page 4-72 |
| BUILD_AND_SEND_BIND_RSP_NEG | page 4-60 |
| BUILD_AND_SEND_BIND_RSP_POS | page 4-61 |
| BUILD_AND_SEND_UNBIND_RQ | page 4-71 |
| BUILD_AND_SEND_INIT_HS | page 4-64 |
| BUILD_AND_SEND_ACT_SESS_RSP_POS | page 4-57 |
| BUILD_AND_SEND_SESS_ACTIVATED | page 4-68 |
| CLEANUP_LU_LU_SESSION | page 4-74 |
| BUILD_AND_SEND_BINDF_RQ | page 4-61 |
| BUILD_AND_SEND_TERM_RQ | page 4-70 |
| BUILD_AND_SEND_UNBIND_RSP | page 4-71 |
| BUILD_AND_SEND_INIT_RQ | page 4-65 |
| BUILD_AND_SEND_ACT_SESS_RSP_NEG | page 4-57 |
| BUILD_AND_SEND_PC_HS_CONNECT | page 4-67 |
| BUILD_AND_SEND_PC_CONNECT | page 4-67 |
| BUILD_AND_SEND_PC_HS_DISCONNECT | page 4-68 |
| BUILD_AND_SEND_RSP_OR_LOG | page 4-66 |
| BUILD_AND_SEND_SESSST_RQ | page 4-70 |
| BUILD_AND_SEND_BIND_RQ | page 4-60 |
| BUILD_AND_SEND_CINIT_RSP | page 4-62 |
| BUILD_AND_SEND_SESS_DEACTIVATED | page 4-69 |
| DEACTIVATE_SESSION | page A-31 |
| BIND_RQ_RCV_RECORD | page A-21 |
| UNBIND_RQ_RCV_RECORD | page A-23 |
| UNBIND_RSP_RCV_RECORD | page A-23 |
| PC_CONNECT_RSP | page A-22 |
| INIT_HS_RSP | page A-11 |
| ABORT_HS | page A-11 |
| HS_RCV_RECORD | page A-11 |
| LULU_CB | page A-5 |
| LOCAL | page 4-99 |

| STATE NAMES----> | RES | PND INI RSP PLU | PND INI RSP SLU | PND CIN | PND BIN | PND PC RSP THI | PND BIN RSP THI | PND INI HS RSP THI | PND PC RSP OTH | PND BIN RSP OTH | PND INI HS RSP OTH | ACT | PND UNB RSP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| INPUTS STATE NUMBERS--> | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 010 | 011 | 012 | 013 |
| ACT_SESS,ILU=PLU | 2U | / | / | / | / | / | / | / | / | / | / | / | / |
| ACT_SESS,ILU=SLU | 3V | / | / | / | / | / | / | / | / | / | / | / | / |
| +RSP(INIT_SELF) | / | 4 | 5 | / | / | / | / | - | / | / | / | - | - |
| -RSP(INIT_SELF) | / | 1AA | 1AA | / | / | / | / | / | / | / | / | / | / |
| NOTIFY_03 | / | 1W | 1W | 1W | 1W | -R | -R | -R | -R | -R | -R | -R | -R |
| CINIT,OK | 9BB | 6BB | -Y | 6BB | -Y | -Y | -Y | -Y | -Y | -Y | -Y | -Y | -Y |
| CINIT,NG | / | 1JJ | -Y | 1JJ | -Y | -Y | -Y | -Y | -Y | -Y | -Y | -Y | -Y |
| +PC_CONNECT_RSP | / | / | / | / | / | 7F | / | / | 10F | / | / | / | / |
| -PC_CONNECT_RSP | / | / | / | / | / | 1LL | / | / | 1MM | / | / | / | / |
| +RSP(BIND),OK | / | / | / | / | / | / | 8C | / | / | 11C | / | / | - |
| +RSP(BIND),NG | / | / | / | / | / | / | 13X | / | / | 13Q | / | / | - |
| -RSP(BIND),OK | / | / | / | / | / | / | 1FF | / | / | 1UU | / | / | 1I |
| -RSP(BIND),NG | / | / | / | / | / | / | 1NN | / | / | 1N | / | / | 1N |
| +INIT_HS_RSP | / | / | / | / | / | / | / | 12D | / | / | 12G | / | - |
| -INIT_HS_RSP | / | / | / | / | / | / | / | 13J | / | / | 13H | / | - |
| BIND,OK | 11E | -EE | 8E | -EE | 8E | -EE | -EE | -EE | / | / | / | -EE | -EE |
| BIND,NG | / | -EE | 1TT | -EE | 1TT | -EE | -EE | -EE | / | / | / | -EE | -EE |
| DEACT_SESS_PEND | / | 1K | 1K | 1K | 1K | 1A | 13B | 13B | / | / | / | 13B | - |
| DEACT_SESS_ACT | / | / | / | / | / | / | / | / | / | / | / | 13B | - |
| DEACT_SESS_ACT_CU | / | / | / | / | / | / | / | / | / | / | / | 1T | 1T |
| ABORT_HS | / | / | / | / | / | / | / | / | / | / | / | 13M | - |
| +RSP(UNBIND) | / | / | / | / | / | / | / | / | / | / | / | / | 1I |
| -RSP(UNBIND) | / | / | / | / | / | / | / | / | / | / | / | / | 1Z |
| UNBIND | / | / | / | / | / | / | 1HH | 1HH | / | 1P | 1P | 1S | 1P |
| CLEANUP | / | / | / | / | / | 1CC | 1DD | 1DD | 1A | 1T | 1T | 1KK | 1T |
| CTERM_ORDERLY | / | / | / | / | / | 1CC | 13PP | 13PP | 1A | 13QQ | 13QQ | -RR | - |
| CTERM_FORCED | / | / | / | / | / | 1CC | 13PP | 13PP | 1A | 13QQ | 13QQ | 13SS | - |
| SIGNAL(RESET_SON) | - | 1L | 1L | 1L | - | - | - | - | - | - | - | - | - |
| SIGNAL(RESET_NORMAL) | - | 1L | 1L | 1L | 1L | 1CC | 1L | 1L | 1I | 1I | 1I | 1GG | 1I |

| OUTPUT CODE | FUNCTION |
|---|---|
| A | Call BUILD_AND_SEND_PC_HS_DISCONNECT(LULU_CB.LU_LU.HS_ID) (page 4-68).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| B | If DEACTIVATE_SESSION.TYPE = NORMAL then<br>Call BUILD_AND_SEND_UNBIND_RQ(LULU_CB, NORMAL, X'00000000') (page 4-71).<br>Else<br>Call BUILD_AND_SEND_UNBIND_RQ(LULU_CB, FORMAT_OR_PROTOCOL_ERROR, DEACTIVATE_SESSION.SENSE_CODE) (page 4-71). |
| C | If either node's path control does not support segmenting and the primary send maximum RU size in the BIND response is greater than the link segment size then<br>Set the maximum RU size to the link maximum RU segment size.<br><br>If the BIND response specifies the primary as contention winner then<br>Set LULU_CB.SESSION_TYPE to FIRST_SPEAKER.<br>Else<br>Set LULU_CB.SESSION_TYPE to BIDDER.<br><br>Call BUILD_AND_SEND_SESSST_RQ(LULU_CB) (page 4-70).<br>Call BUILD_AND_SEND_INIT_HS(LULU_CB, BIND image from BIND response RU, PRI) (page 4-64). |
| D | Call BUILD_AND_SEND_ACT_SESS_RSP_POS(LULU_CB) (page 4-57). |
| E | Set fields in LULU_CB from BIND request record (ADDRESS, ALS [peripheral nodes only], PC_ID, and user-data session-instance identifier).<br>Create LU-LU half-session with unique identifier (save identifier in LULU_CB.LU_LU.HS_ID).<br>Call BUILD_AND_SEND_PC_HS_CONNECT(LULU_CB.LU_LU.PC_ID, LULU_CB.LU_LU.HS_ID, LULU_CB.LU_LU.ADDRESS) (page 4-67).<br>Call BUILD_AND_SEND_BIND_RSP_POS(BIND_RQ_RCV_RECORD, LULU_CB) (negotiated BIND image returned, page 4-61).<br>Call BUILD_AND_SEND_SESSST_RQ(LULU_CB) (page 4-70).<br>Call BUILD_AND_SEND_INIT_HS(LULU_CB, negotiated BIND image, SEC) (page 4-64). |
| F | If this node is a peripheral node then<br>Set LULU_CB.LU_LU.ADDRESS to PC_CONNECT_RSP.ADDRESS to save the assigned address for later use. Subarea nodes obtain the address from the CINIT request.<br>Call BUILD_AND_SEND_PC_HS_CONNECT(LULU_CB.LU_LU.PC_ID, LULU_CB.LU_LU.HS_ID, LULU_CB.LU_LU.ADDRESS) (page 4-67).<br>Call BUILD_AND_SEND_BIND_RQ(LULU_CB) (page 4-60). |
| G | Call BUILD_AND_SEND_SESS_ACTIVATED(LULU_CB) (page 4-68). |
| H | Call BUILD_AND_SEND_UNBIND_RQ(LULU_CB, FORMAT_OR_PROTOCOL_ERROR, INIT_HS_RSP.SENSE_CODE) (page 4-71). |
| I | Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| J | Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, NO_RETRY) (page 4-57).<br>Call BUILD_AND_SEND_UNBIND_RQ(LULU_CB, FORMAT_OR_PROTOCOL_ERROR, INIT_HS_RSP.SENSE_CODE) (page 4-71). |
| K | Call BUILD_AND_SEND_TERM_RQ(LULU_CB, DEACTIVATE_SESSION.TYPE) (page 4-70).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| L | Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, NO_RETRY) (page 4-57).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| M | Call BUILD_AND_SEND_UNBIND_RQ(LULU_CB, FORMAT_OR_PROTOCOL_ERROR, ABORT_HS.SENSE_CODE) (page 4-71).<br>Call BUILD_AND_SEND_SESS_DEACTIVATED(LULU_CB.LU_LU.HS_ID, ABNORMAL_NO_RETRY) (page 4-69). |

| | |
|---|---|
| N | Call BUILD_AND_SEND_BINDF_RQ(SETUP_REJECT_AT_SLU, LULU_CB, LOCAL.SENSE_CODE) (page 4-61).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| P | Call BUILD_AND_SEND_UNBIND_RSP(UNBIND_RQ_RCV_RECORD,LULU_CB_PTR) (page 4-71).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| Q | Call BUILD_AND_SEND_UNBIND_RQ(LULU_CB, INVALID_PARMS, X'00000000') (page 4-71). |
| R | If LOCAL.SENSE_CODE = X'00000000' then<br>   Set LOCAL.SENSE_CODE = X'08090000' (mode inconsistency).<br>Call BUILD_AND_SEND_RSP_OR_LOG(HS_RCV_RECORD) (send -RSP(NOTIFY), page 4-66). |
| S | Call BUILD_AND_SEND_UNBIND_RSP(UNBIND_RQ_RCV_RECORD,LULU_CB_PTR) (page 4-71).<br>Determine the reason for the session deactivation. If the UNBIND type is normal or BIND forthcoming, the reason is NORMAL. If the UNBIND type is virtual route inoperative or route extension inoperative, the reason is ABNORMAL_RETRY. For all other UNBIND types, the reason is ABNORMAL_NO_RETRY.<br>Call BUILD_AND_SEND_SESS_DEACTIVATED(LULU_CB.LU_LU.HS_ID, reason) (page 4-69).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| T | Call BUILD_AND_SEND_UNBIND_RQ(LULU_CB, CLEANUP, X'00000000') (page 4-71). (Send UNBIND(CLEANUP), page 4-71).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| U | Call BUILD_AND_SEND_INIT_RQ(LULU_CB, SLU) (page 4-65). |
| V | Call BUILD_AND_SEND_INIT_RQ(LULU_CB, PLU) (page 4-65). |
| W | Call BUILD_AND_SEND_RSP_OR_LOG(HS_RCV_RECORD) (send ±RSP(NOTIFY), page 4-66).<br>Determine the error type by examining the sense data in the NOTIFY request (see Note in prologue).<br>Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, error type) (page 4-57).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| X | Call BUILD_AND_SEND_UNBIND_RQ(LULU_CB, INVALID_PARMS, X'00000000') (page 4-71).<br>Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, NO_RETRY) (page 4-57). |
| Y | If LOCAL.SENSE_CODE = X'00000000' then<br>   Set LOCAL.SENSE_CODE to X'08150000' (function already active).<br>Call BUILD_AND_SEND_CINIT_RSP(HS_RCV_RECORD) (page 4-62). |
| Z | Call BUILD_AND_SEND_UNBINDF_RQ(sense data from UNBIND_RSP_RCV_RECORD, LULU_CB) (page 4-72).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| AA | Determine the error type by examining the sense data in the INIT-SELF response (see Note in prologue).<br>Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, error type) (page 4-57).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| BB | Call BUILD_AND_SEND_CINIT_RSP(HS_RCV_RECORD) (page 4-62) to send +RSP(CINIT).<br>Create a new half-session (HS) process.<br>Call BUILD_AND_SEND_PC_CONNECT(LULU_CB) (page 4-67).<br>Save the CINIT request in the LULU_CB for later use in building the BIND request. |
| CC | Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, NO_RETRY) (page 4-57).<br>Call BUILD_AND_SEND_PC_HS_DISCONNECT(LULU_CB.LU_LU.HS_ID) (page 4-68).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |

| | |
|---|---|
| DD | Call BUILD_AND_SEND_UNBIND_RQ(LULU_CB, CLEANUP, X'00000000') (page 4-71)<br>Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, NO_RETRY) (page 4-57).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| EE | If LOCAL.SENSE_CODE = X'00000000' then<br>   Set LOCAL.SENSE_CODE = X'08150000' (function already active).<br>Call BUILD_AND_SEND_BIND_RSP_NEG(BIND_RQ_RCV_RECORD) (page 4-60). |
| FF | Call BUILD_AND_SEND_BINDF_RQ(SETUP_REJECT_AT_SLU, LULU_CB, sense data from the BIND response) (page 4-61).<br>Determine the error type by examining the sense code in the BIND response (see Note in prologue).<br>Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, error type) (page 4-57).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| GG | Call BUILD_AND_SEND_SESS_DEACTIVATED(LULU_CB.LU_LU.HS_ID, ABNORMAL_NO_RETRY) (page 4-69).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| HH | Call BUILD_AND_SEND_UNBIND_RSP(UNBIND_RQ_RCV_RECORD) (page 4-71).<br>Determine the error type by examining the Type field in the UNBIND request. If it is virtual route inoperative or route extension inoperative, then the error type is RETRY; otherwise, the error type is NO_RETRY.<br>Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, error type) (page 4-57).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| JJ | If LOCAL.SENSE_CODE = X'00000000' then<br>   Set LOCAL.SENSE_CODE = X'08150000' (function already active).<br>Call BUILD_AND_SEND_CINIT_RSP(HS_RCV_RECORD) (page 4-62) to send -RSP(CINIT).<br>Determine the error type by examining LOCAL.SENSE_CODE (see Note in prologue).<br>Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, error type) (page 4-57).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| KK | Call BUILD_AND_SEND_SESS_DEACTIVATED(LULU_CB.LU_LU.HS_ID, ABNORMAL_NO_RETRY) (page 4-69).<br>Call BUILD_AND_SEND_UNBIND_RQ(LULU_CB, CLEANUP, X'00000000') (page 4-71).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| LL | If LOCAL.SENSE_CODE = X'00000000' then<br>   Set LOCAL.SENSE_CODE to PC_CONNECT_RSP.SENSE_CODE.<br>Call BUILD_AND_SEND_BINDF_RQ(SETUP_REJECT_AT_PLU, LULU_CB, LOCAL.SENSE_CODE) (page 4-61).<br>Determine the error type by examining LOCAL.SENSE_CODE (see Note in prologue).<br>Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, error type) (page 4-57).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| MM | If LOCAL.SENSE_CODE = X'00000000' then<br>   Set LOCAL.SENSE_CODE to PC_CONNECT_RSP.SENSE_CODE.<br>Call BUILD_AND_SEND_BINDF_RQ(SETUP_REJECT_AT_PLU, LULU_CB, LOCAL.SENSE_CODE) (page 4-61).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| NN | Determine the error type by examining LOCAL.SENSE_CODE (see Note in prologue).<br>Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, error type) (page 4-57).<br>Call BUILD_AND_SEND_BINDF_RQ(SETUP_REJECT_AT_SLU, LULU_CB, LOCAL.SENSE_CODE) (page 4-61).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |

| | |
|---|---|
| PP | Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, NO_RETRY) (page 4-57).<br>Call BUILD_AND_SEND_UNBIND_RQ(LULU_CB, NORMAL, X'00000000') (page 4-71). |
| QQ | Call BUILD_AND_SEND_UNBIND_RQ(LULU_CB, NORMAL, X'00000000') (page 4-71). |
| RR | Call BUILD_AND_SEND_DEACTIVATE_SESS(LULU_CB.LU_LU.HS_ID) (page 4-63). |
| SS | Call BUILD_AND_SEND_UNBIND_RQ(LULU_CB, NORMAL, X'00000000') (page 4-71).<br>Call BUILD_AND_SEND_SESS_DEACTIVATED(LULU_CB.LU_LU.HS_ID, ABNORMAL_NO_RETRY) (page 4-69). |
| TT | Call BUILD_AND_SEND_BIND_RSP_NEG(BIND_RQ_RCV_RECORD) (page 4-60).<br>Call BUILD_AND_SEND_ACT_SESS_RSP_NEG(LULU_CB.CORRELATOR, error type (see Note in prologue)) (page 4-57).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |
| UU | Call BUILD_AND_SEND_BINDF_RQ(SETUP_REJECT_AT_SLU, LULU_CB, sense data from the BIND response) (page 4-61).<br>Call CLEANUP_LU_LU_SESSION(LULU_CB) (page 4-74). |

## LOCAL DATA STRUCTURES

LOCAL

```
LOCAL (this control block is accessible by any procedure in LNS)
  CPLU_CB_LIST list of CP-LU half-session control blocks (page A-1)
  LULU_CB_LIST list of LU-LU half-session control blocks (page A-5)
  SENSE_CODE (this field is set with a sense data value whenever an error is found)
```

ERROR_TYPE

ERROR_TYPE: possible values: RETRY, NO_RETRY

SESSION_TYPE

SESSION_TYPE: possible values: FIRST_SPEAKER, BIDDER

# 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 and returned 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 may be 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 on 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 and destroys the PS process, and assigns half-sessions to it for conversation traffic. PS exchanges data with these half-sessions in carrying out transaction program verb requests. PS also interacts with the transaction program; or, more precisely, PS contains, and is driven by, a transaction program execution instance (TP).

### 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.

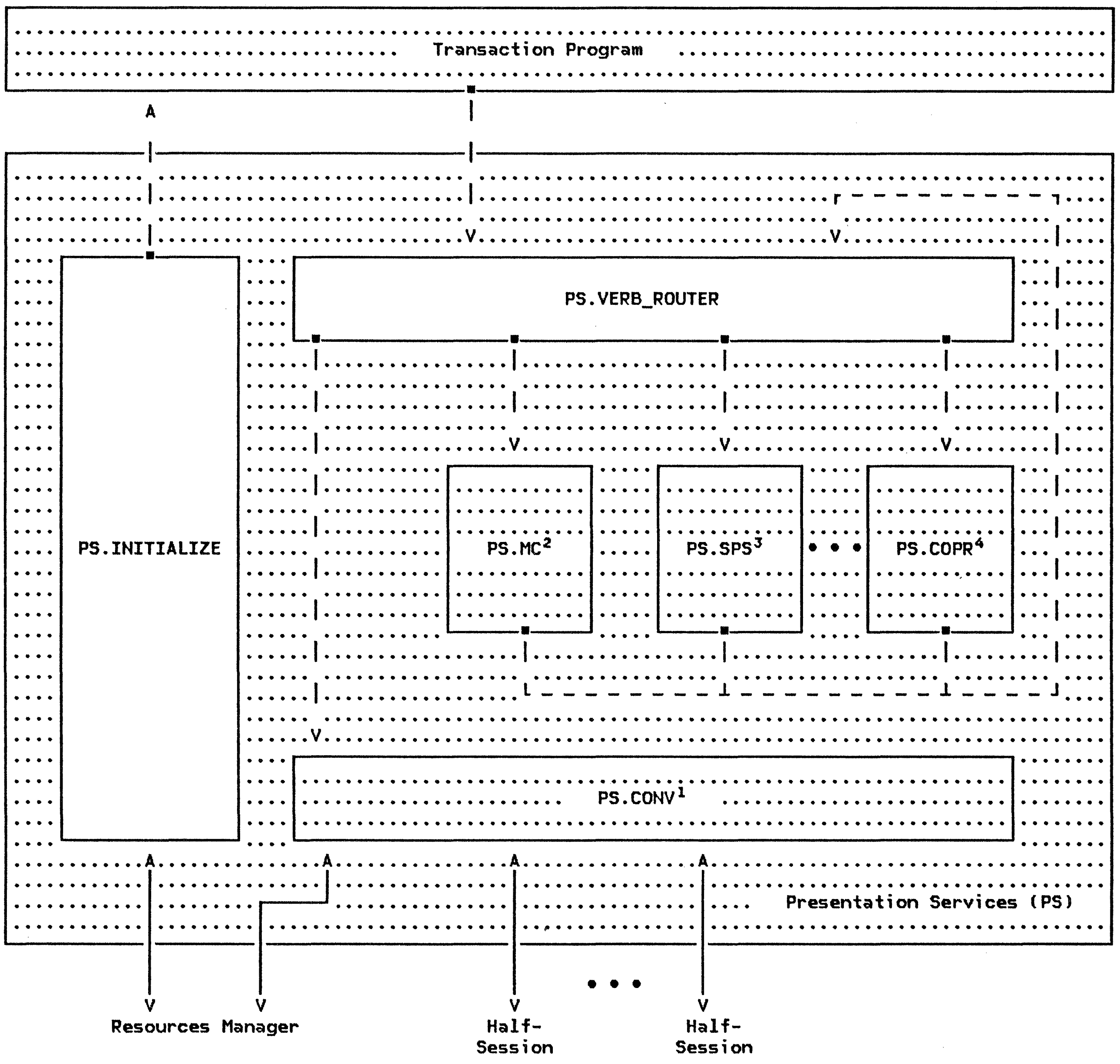
PS.CONV:

- 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 asynchronous inter-process 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



[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

components is described in detail in the corresponding chapters.

PS_PROCESS_DATA on page 5.0-20 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 with data passed from RM when it creates the PS process, and they remain unchanged thereafter.

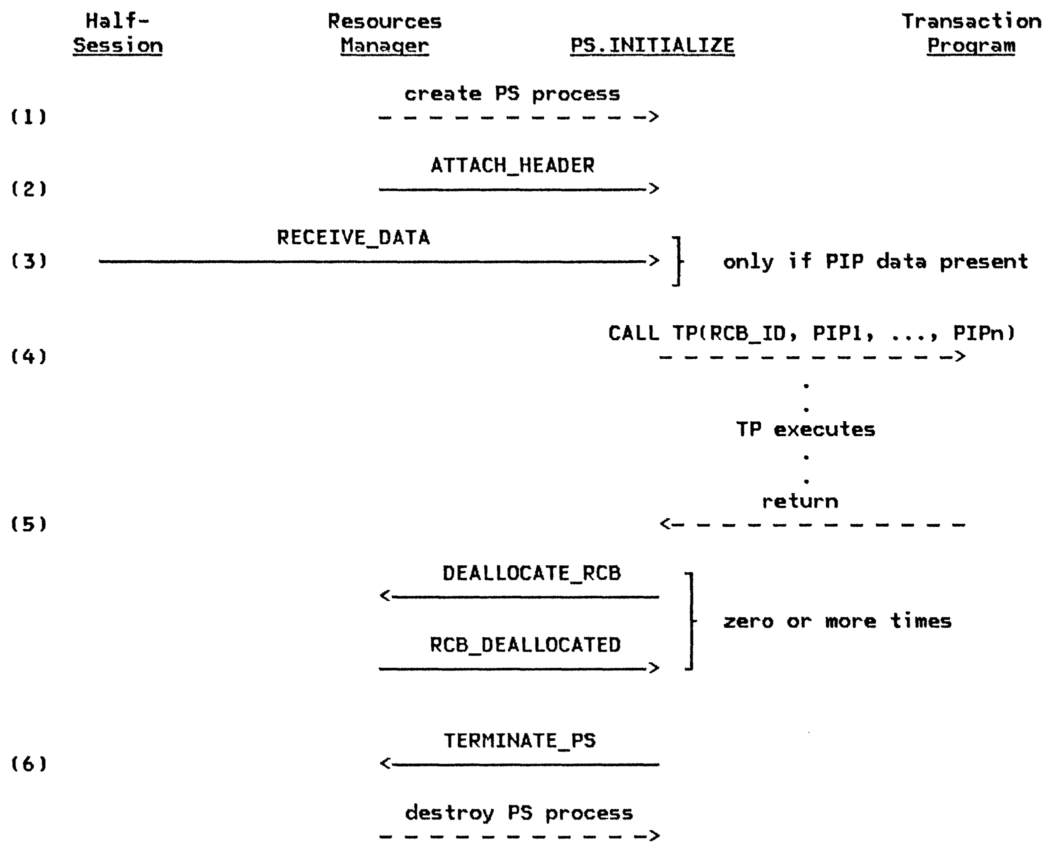


Figure 5.0-2. Initialization and Termination of Presentation Services and Transaction Program

The transaction control block (TCB, page A-10) contains information specific to the transaction program instance, such as the list of resources allocated to it. 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-7) 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. 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.

## INITIALIZATION AND TERMINATION (PS.INITIALIZE)

The PS.INITIALIZE component performs initialization and termination of PS and the TP.

Figure 5.0-2 shows the protocol boundary flows that are used by PS.INITIALIZE for initialization and termination of the PS process. The steps below correspond to the numbers in the figure.

1. The PS process is created by RM, which passes it several parameters, 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.

2. 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 is 0 (indicating no error was found by RM), PS.INITIALIZE performs additional checking of the Attach. This checking includes a check of the security fields and the transaction program's support of the conversation type and program initialization parameters (PIP data). If the Attach is in error (as determined by RM or PS.INITIALIZE) the conversation is terminated. Depending on the error detected, the session may be deactivated,

or the conversation ended with DEALLOCATE TYPE(ABEND_PROG).

3. 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.

4. 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).

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 record to the resources manager and waits to be terminated. On receipt of the TERMINATE_PS record, RM destroys the PS process.

## VERB PROCESSING (PS.VERB_ROUTER)

PS.VERB_ROUTER routes verbs to the appropriate PS verb-processing component. It also processes resource-independent verbs such as WAIT and GET_TYPE. The supplied RESOURCE parameter of most verbs identifies the conversation on which the verb is being issued. The value in the RESOURCE parameter must match one in TCB.RESOURCES_LIST, the list of resources allocated to the TP; if it does not, the TP is terminated abnormally.

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 TP. 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 is not processed by PS.CONV, because (unlike most verbs) it is not issued over a conversation. Instead, 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 activated 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.

## PS

| | |
|---|---|
| FUNCTION: | Presentation services (PS) provides verb-processing services to a transaction program execution instance (TP). PS invokes, terminates, contains, and is driven by the TP. |
| INPUT: | LUCB_LIST_PTR, TCB_LIST_PTR, and RCB_LIST_PTR, pointers to LUCB_LIST, TCB_LIST, and RCB_LIST, respectively; LU_ID, the ID of this LU; and TCB_ID, the ID of this PS process |
| OUTPUT: | Process data is initialized and PS.INITIALIZE is invoked. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS_INITIALIZE | page 5.0-6 |
| PS_PROCESS_DATA | page 5.0-20 |
| TCB | page A-10 |
| LUCB | page A-1 |
| LUCB_LIST_PTR | page 5.0-21 |
| RCB_LIST_PTR | page 5.0-21 |
| TCB_LIST_PTR | page 5.0-21 |
| LU_ID | page 5.0-21 |
| TCB_ID | page 3-69 |

```
Copy the input parameters into the fields of PS_PROCESS_DATA (page 5.0-20).
Set PS_PROCESS_DATA.LUCB_PTR to point to the LUCB for this LU (identified by LU_ID).
Set PS_PROCESS_DATA.TCB_PTR to point to the TCB for this transaction process
 (identified by TCB_ID).

Call PS.INITIALIZE (page 5.0-6).
```

## PS_INITIALIZE

| | |
|---|---|
| FUNCTION: | This procedure creates and invokes an instance of the transaction program named in a received FMH-5 (Attach).<br><br>After PS is created by RM, PS.INITIALIZE receives the Attach and other information from RM. As shown in "Function Management Header 5: Attach " on page H-6, the 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. PS.INITIALIZE receives the PIP data (if any) from the half-session, and validates fields of the Attach.<br><br>If the Attach is valid, PS invokes the transaction program named in the Attach. When the TP returns to PS.INITIALIZE, the PS process is destroyed (by calling DEALLOCATION_CLEANUP_PROC).<br><br>If the Attach contains an error, ATTACH_ERROR_PROC is called and the PS process is destroyed; no transaction program is invoked. |
| INPUT: | Attach information (from RM) |
| OUTPUT: | An execution instance of a transaction program is loaded and called. The TP is passed the RCB_ID representing the conversation between the attached program and the attaching program, and PIP data, if there is any. |
| NOTE: | Rather than invoking the transaction program immediately upon receipt of the Attach, PS may await the receipt of data indicating end-of-chain before dispatching the transaction program. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| INITIALIZE_ATTACHED_RCB | page 5.0-17 |
| RECEIVE_PIP_FIELD_FROM_HS | page 5.0-7 |
| PS_ATTACH_CHECK | page 5.0-8 |
| UPM_EXECUTE | page 5.0-18 |
| ATTACH_ERROR_PROC | page 5.0-10 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| PS_PROCESS_DATA | page 5.0-20 |
| TCB | page A-10 |
| RCB | page A-7 |
| PIP_FIELD | page 5.0-20 |
| PIP_LIST | page 5.0-21 |
| CODE, see SENSE_DATA | page 5.0-22 |

```
Receive Attach information from RM.
If the Attach (see page H-6 for format) indicates
 that PIP data is present, then
   Call RECEIVE_PIP_FIELD_FROM_HS(RCB,PIP_FIELD) (page 5.0-7).
Else
   Set PIP_FIELD to null.
Call PS_ATTACH_CHECK (page 5.0-8),
 and pass it PIP_FIELD and relevant Attach information.
Store the resulting sense data in CODE.
If the Attach is valid (CODE=0) then
   Set RCB to the RCB_LIST-element identified by ATTACH_RECEIVED.RCB_ID.
   Call INITIALIZE_ATTACHED_RCB(ATTACH_RECEIVED) (page 5.0-17).
   Copy the transaction program name and access security fields of the Attach
    into the TCB.
   Set TCB.CONTROLLING_COMPONENT to TP.

   Call RCB.FSM_CONVERSATION(R, ATTACH, RCB) (page 5.1-59).
   Call UPM_EXECUTE(TCB.TRANSACTION_PROGRAM_NAME, RCB.RCB_ID, PIP_LIST)
    (page 5.0-18) (see Note).
Else (Attach is not valid)
   Call ATTACH_ERROR_PROC(RCB, CODE) (page 5.0-10).

Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).
```

## RECEIVE_PIP_FIELD_FROM_HS

| | | |
|---|---|---|
| FUNCTION: | | During invocation of the transaction program, this procedure receives a program initialization parameter (PIP) field by issuing a RECEIVE_AND_WAIT verb. If this verb-issuance succeeds in receiving a complete logical record containing a PIP Data GDS variable, then the received PIP field is returned to PS.INITIALIZE. 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; a PIP Data GDS variable from the half-session |
| OUTPUT: | | A RECEIVE_AND_WAIT verb is issued in order to retrieve the expected PIP data, which is returned to PS.INITIALIZE via PIP_FIELD. |
| NOTES: | 1. | The RECEIVE_AND_WAIT structure that is created in this procedure cannot be destroyed at only the end of the procedure. The reason for this is that if a protocol violation is detected and DEALLOCATION_CLEANUP_PROC is invoked, this process ends without ever returning to this procedure (in this case, the RECEIVE_AND_WAIT would never be destroyed). |
| | 2. | This error occurs if the partner indicates in the Attach that PIP data follows, but either no data follows or the data that follows is not PIP data. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| PS_PROTOCOL_ERROR | page 5.0-16 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| PS_PROCESS_DATA | page 5.0-20 |
| RCB | page A-7 |
| PIP_FIELD | page 5.0-20 |

```
Issue a RECEIVE_AND_WAIT verb on this conversation, with
 a MAX_LENGTH of X'7FFF', a FILL of LL, and a DATA parameter of PIP_FIELD.
If PIP_FIELD does not now contain
 a complete PIP Data GDS variable (see page H-15 for format),
 then (optional receive check—see Note 2)
   Call PS_PROTOCOL_ERROR(RCB.HS_ID, RCB.RCB_ID, X'1008201D') (page 5.0-16).
   Call DEALLOCATION_CLEANUP_PROC  (page 5.0-14)
```

## PS_ATTACH_CHECK

| | |
|---|---|
| FUNCTION: | This procedure validates some fields of the received Attach (RM validates the other fields). |
| INPUT: | Attach information (from RM) and program initialization parameter (PIP) data from HS. |
| OUTPUT: | 0, if no invalid fields are found; the appropriate sense data, otherwise |
| NOTE: | If RM finds the Attach invalid, it is accompanied by sense data (in SENSE_CODE) with one of the following values: |

| | |
|---|---|
| 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, SENSE_CODE = X'00000000'.

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| PS_PROCESS_DATA | page 5.0-20 |
| TCB | page A-10 |
| LUCB | page A-1 |
| TRANSACTION_PROGRAM | page A-4 |
| PIP_FIELD | page 5.0-20 |
| SENSE_CODE, see SENSE_DATA | page 5.0-22 |

```
If SENSE_CODE > 0, then
   Return sense data set by RM.
Else (continue seeking Attach errors)
   Set TRANSACTION_PROGRAM to the LUCB.TRANSACTION_PROGRAM_LIST-element
    named in Attach.

Select, in order, based on
 the contents of the Attach (for format, see page H-6):

          ┌──────────────────────────────────────────────────────────────────────────────┐
          │               Errors that cause the session to be deactivated                │
          └──────────────────────────────────────────────────────────────────────────────┘

   When the Logical Unit of Work Identifier fields are incorrectly formatted
      Return X'10086011'.

          ┌──────────────────────────────────────────────────────────────────────────────┐
          │                 Errors that cause an FMH-7 to be generated                   │
          └──────────────────────────────────────────────────────────────────────────────┘

   When TRANSACTION_PROGRAM.NUMBER_OF_PIP_SUBFIELDS=0,
    but the Attach indicates that PIP data is present
      Return X'10086031' (PIP not allowed).

   When TRANSACTION_PROGRAM.NUMBER_OF_PIP_SUBFIELDS is positive, but
    the actual number of PIP subfields (in PIP_FIELD) differs from it,
    or PIP data is not indicated as present
      Return X'10086032' (PIP not specified correctly).

   When the Resource type is not supported by the transaction program
    (i.e., is not on the TRANSACTION_PROGRAM.RESOURCES_SUPPORTED_LIST)
      Return X'10086034' (conversation type mismatch).

   When the Access Security Information fields are not valid
    (according to page H-7)
      Return X'080F6051'.

   Otherwise (ATTACH is valid)
      Return X'00000000'.
```

## 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 requests that the session 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. PS also sends DEALLOCATE_RCB to RM, and then instructs RM to terminate the PS process. |
| INPUT: | The RCB corresponding to the conversation over which the invalid Attach was received, and sense data specifying the type of Attach error are received as parameters. |
| 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:

| | |
|---|---|
| PS | page 5.0-5 |
| | |
| PS_PROTOCOL_ERROR | page 5.0-16 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| GET_END_CHAIN_FROM_HS | page 5.1-34 |
| SEND_ERROR_TO_HS_PROC | page 5.1-52 |
| UPM_ATTACH_LOG | page 5.0-19 |
| SEND_DATA_BUFFER_MANAGEMENT | page 5.1-47 |
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| | |
| PS_PROCESS_DATA | page 5.0-20 |
| RCB | page A-7 |
| BUFFER_ELEMENT | page A-8 |
| SENSE_CODE, see SENSE_DATA | page 5.0-22 |

```
If SENSE_CODE is
 X'1008200E', X'10086000', X'10086005', X'10086009', X'10086011', or X'10086040',
 then (deactivate the session)
   Call PS_PROTOCOL_ERROR(RCB.HS_ID, RCB.RCB_ID, SENSE_CODE) (page 5.0-16).
   Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).
Else (generate an FMH-7)
   Call SEND_ERROR_TO_HS_PROC(RCB) (page 5.1-52).
   Call GET_END_CHAIN_FROM_HS(RCB) (page 5.1-34).
   Set BUFFER_ELEMENT to the last entry in RCB.HS_TO_PS_BUFFER_LIST.
   If the BUFFER_ELEMENT type is
    DEALLOCATE_CONFIRM, CONFIRM, PREPARE_TO_RCV_CONFIRM, or PREPARE_TO_RCV_FLUSH
    then
      Call UPM_ATTACH_LOG (page 5.0-19) to
       generate log data describing this Attach error.
      If this log data is non-null, then
         Log it in the local system error log.
         Put into the conversation's send buffer (RCB.PS_TO_HS_RECORD.DATA)
          an FMH-7 (page H-8) indicating that
          log data follows and sense data (from SENSE_CODE) is included.
         Append to the conversation's send buffer an
          Error Log GDS variable (page H-15) containing the log data.
          and append this GDS variable to the send buffer.
      Else
         Put into the conversation's send buffer (RCB.PS_TO_HS_RECORD.DATA)
          an FMH-7 (page H-8) indicating that
          no log data follows and sense data (from SENSE_CODE) is included.
         Call SEND_DATA_BUFFER_MANAGEMENT( null string, RCB ) (page 5.1-47).
         Set RCB.PS_TO_HS_RECORD.TYPE to DEALLOCATE_FLUSH.
         Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).
   Send a DEALLOCATE_RCB record (page A-26), derived from this RCB, to RM.
```

## 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: | | 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, and may 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. Thus, PS.MC may issue a basic conversation verb on a mapped conversation. PS.MC is allowed to do this because it is a "service" component that is part of PS; the transaction program is not. PS.VERB_ROUTER maintains knowledge, via the CONTROLLING_COMPONENT field in the TCB, of whether the verb currently being processed was issued by the transaction program or by a service component such as PS.MC. |
| | 2. | If the TP issues a verb that is incompatible with the specified resource, such as a mapped conversation verb specifying a basic conversation, then the TP has committed a protocol violation and is terminated abnormally. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| PS_CONV | page 5.1-10 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| WAIT_PROC | page 5.0-15 |
| PS_MC | page 5.2-20 |
| PS_COPR | page 5.4-32 |
| PS_SPS | page 5.3-20 |
| PS_PROCESS_DATA | page 5.0-20 |
| TCB | page A-10 |
| RCB | page A-7 |

Select based on TRANSACTION_PGM_VERB contents

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, REQUEST_TO_SEND, SEND_DATA, or SEND_ERROR
   If the supplied RESOURCE parameter of the verb
    fails to identify a conversation assigned to this transaction
    (i.e., does not occur on on TCB.RESOURCES_LIST), then
      Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).
   Find the RCB for the conversation identified by the supplied RESOURCE parameter.
   If RCB.CONVERSATION_TYPE≠BASIC_CONVERSATION and
    TCB.CONTROLLING_COMPONENT≠SERVICE_COMPONENT
    then (see Note 1)
      Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).
   Call PS_CONV(verb,parameters) (page 5.1-10).
```

| 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, or MC_SEND_ERROR
   If the verb's supplied RESOURCE parameter
    fails to identify a conversation assigned to this transaction
    (i.e., fails to occur on TCB.RESOURCES_LIST), then
      Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).
   Find the RCB for the conversation identified by RESOURCE.
   If RCB.CONVERSATION_TYPE is not MAPPED_CONVERSATION,
    then (it should be, because this verb is mapped)
      Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).
   Set TCB.CONTROLLING_COMPONENT to SERVICE_COMPONENT.
   Call PS_MC(verb,parameters) (page 5.2-20).
   Set TCB.CONTROLLING_COMPONENT back to TP.
```

| 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, SET_CPLU_CAPABILITY,
 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, GET_CPLU_CAPABILITY,
 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, LIST_CPLU_CAPABILITY,
 PROCESS_SESSION_LIMIT, ACTIVATE_SESSION, or DEACTIVATE_SESSION
   Set TCB.CONTROLLING_COMPONENT to SERVICE_COMPONENT,
   Call PS_COPR(verb parameters) (page 5.4-32), and
   Set TCB.CONTROLLING_COMPONENT  back to TP.
```

| Resource-Independent Verbs |
|---|

```
 When SYNCPT or BACKOUT
    Set TCB.CONTROLLING_COMPONENT to SERVICE_COMPONENT,
    Call PS_SPS (page 5.3-20),
    Set TCB.CONTROLLING_COMPONENT back to TP.

 When GET_TYPE
    If the verb's supplied RESOURCE parameter
     fails to identify a conversation assigned to this transaction, then
       Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).
    Find the RCB for the conversation identified by RESOURCE.
    Copy RCB.CONVERSATION_TYPE into the verb's returned TYPE parameter.

 When WAIT
    Set TCB.CONTROLLING_COMPONENT to SERVICE_COMPONENT,
    Call WAIT_PROC(verb,parameters)  (page 5.0-15),
    Set TCB.CONTROLLING_COMPONENT back to TP.

RETURN;
```

## DEALLOCATION_CLEANUP_PROC

| | |
|---|---|
| FUNCTION: | This procedure, which manages the destruction of this process, is invoked after the TP has ended (normally or abnormally) by returning to PS_INITIALIZE on page 5.0-6. It calls UPM_RETURN_PROCESSING on page 5.0-19 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 instructs RM (by sending a TERMINATE_PS record) to destroy the process. |
| INPUT: | TCB.RESOURCES_LIST |
| OUTPUT: | DEALLOCATE_RCB and TERMINATE_PS to RM. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| UPM_RETURN_PROCESSING | page 5.0-19 |
| PS_PROCESS_DATA | page 5.0-20 |
| TCB | page A-10 |
| RCB | page A-7 |
| TERMINATE_PS | page A-27 |
| RCB_ID | page 3-69 |

```
For each RCB_ID on TCB.RESOURCES_LIST, do the following:
   Find the RCB for the conversation identified by RCB_ID.
   If the conversation is not already in RESET or END_CONV state, then
      Call UPM_RETURN_PROCESSING(RESOURCE.RCB_ID) (page 5.0-19).
   Send a DEALLOCATE_RCB record (page A-26), derived from this RCB, to RM.

Send a TERMINATE_PS record to RM.
Wait to be destroyed by RM.
```

## 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 TEMPORARY_RESOURCE_LIST. This sublist contains only those resources from RESOURCE_LIST that are currently activated for posting. (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.) <br><br> After creating the TEMPORARY_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. (Any return code other than UNSUCCESSFUL or POSTING_NOT_ACTIVE indicates that a resource was posted and the WAIT completed successfully.) If the verb completed successfully, then RESOURCE_POSTED indicates which resource has been posted. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| TEST_FOR_RESOURCE_POSTED | page 5.0-18 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| PS_PROCESS_DATA | page 5.0-20 |
| TCB | page A-10 |
| RCB | page A-7 |
| RC, see RETURN_CODE | page 5.0-20 |

```
Check that all resources in the supplied RESOURCE_LIST parameter
 are validly allocated to this transaction (i.e., occur in TCB.RESOURCES_LIST),
 and that at least one of them has posting active.
If any resource is invalid, then
   Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).
If no resource has been posted, then
   Set the verb's primary RETURN_CODE to POSTING_NOT_ACTIVE, and return.
```

At this point (since all resources are valid and some have "posting active"), it is safe to wait for a resource to become posted. If some resource is already posted, though, then there is no need to wait.

```
For each resource that has posting active,
   Call TEST_FOR_RESOURCE_POSTED (page 5.0-18)
    on its RCB, and save the result in RC.
   If RC is not UNSUCCESSFUL, then
      Set the verb's RETURN_CODE to RC, and return.
```

Since no active resource is posted yet, wait until one is.

```
Initialize RC to UNSUCCESSFUL.
Do While RC remains UNSUCCESSFUL
   Suspend this process until an HS for a posting-active resource forwards received
    data to that resource.
   Set RCB to the RCB for the conversation on which the data has arrived.
   Call TEST_FOR_RESOURCE_POSTED(RCB) (page 5.0-18--see Note 4),
    and save the result in RC.
Set the returned RESOURCE_POSTED parameter to RCB.RCB_ID.
Set the verb's RETURN_CODE to RC.
```

PS_PROTOCOL_ERROR

| | |
|---|---|
| FUNCTION: | This procedure processes receive error conditions that require the session to be deactivated.<br><br>An UNBIND_PROTOCOL_ERROR record is sent to the resources manager to request deactivation of the session that committed the protocol violation. Then the procedure creates a CONVERSATION_FAILURE (PROTOCOL_VIOLATION) record and continues the conversation failure processing. |
| INPUT: | The HS ID for the half-session that committed the protocol violation, the RCB ID for the conversation that is using the session, 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:

| | |
|---|---|
| CONVERSATION_FAILURE_PROC | page 5.1-29 |
| | |
| PS_PROCESS_DATA | page 5.0-20 |
| UNBIND_PROTOCOL_ERROR | page A-28 |
| CONVERSATION_FAILURE | page A-32 |
| HS_ID | page 3-69 |
| RCB_ID | page 3-69 |
| SENSE_CODE, see SENSE_DATA | page 5.0-22 |

```
Create an UNBIND_PROTOCOL_ERROR record (page A-28)
 with this TCB_ID, HS_ID, and SENSE_CODE.
Send UNBIND_PROTOCOL_ERROR to RM.

Create a CONVERSATION_FAILURE record with RCB_ID for this conversation.
Set its REASON to PROTOCOL_VIOLATION.
Call CONVERSATION_FAILURE_PROC(CONVERSATION_FAILURE) (page 5.1-29).
```

## INITIALIZE_ATTACHED_RCB

| | |
|---|---|
| FUNCTION: | This procedure initializes fields in the RCB for the resource specified in the received Attach.<br><br>This procedure is invoked by PS.INITIALIZE when RM forwards Attach information to PS. |
| INPUT: | Attach information (from RM) (see "Function Management Header 5: Attach " on page H-6 for format) |
| OUTPUT: | Fields in the specified RCB are initialized. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| PS_PROCESS_DATA | page 5.0-20 |
| RCB | page A-7 |

```
Find the RCB identified in the received Attach.
Initialize this RCB's fields as follows:
 PS_TO_HS_RECORD.ALLOCATE to  NO
 PS_TO_HS_RECORD.FMH to NO
 PS_TO_HS_RECORD.TYPE to NOT_END_OF_DATA
 PS_TO_HS_RECORD.DATA to null string

 SEND_LL_REMAINDER to 0
 RECEIVE_LL_REMAINDER to 0
 MAX_BUFFER_LENGTH to an implementation-defined maximum
 RQ_TO_SEND_RCVD to NO
 LOCKS to SHORT
 POST_CONDITIONS.FILL to LL
 POST_CONDITIONS.MAX_LENGTH to 0
 SEND_LL_BYTE to NOT_PRESENT

 SYNC_LEVEL to the synchronization level specified in the Attach
 CONVERSATION_TYPE to the Resource type specified in the Attach

If RCB.CONVERSATION_TYPE = MAPPED_CONVERSATION then
   Initialize additional RCB fields as follows:
    MC_RQ_TO_SEND_RCVD to NO
    MC_POST to RESET
    MC_MAX_SEND_SIZE to an implementation-defined value
    MAPPER_SAVE_AREA according to an 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 calls either TEST_PROC on page 5.1-26 or MC_TEST_PROC on page 5.2-28 to test whether the resource has been po |
| INPUT: | The RCB for the resource whose posting status is to be determined |
| OUTPUT: | The return code returned from the TEST_PROC or MC_TEST_PROC call. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| TEST_PROC | page 5.1-26 |
| MC_TEST_PROC | page 5.2-28 |
| PS_PROCESS_DATA | page 5.0-20 |
| RCB | page A-7 |
| RETURN_CODE | page 5.0-20 |

```
Select based on RCB.CONVERSATION_TYPE:
   When basic:
      Call TEST_PROC(RCB,POSTED) (page 5.1-26).
   When mapped:
      Call MC_TEST_PROC(RCB,POSTED) (page 5.2-28).
Return the verb's RETURN_CODE.
```

## UNDEFINED PROTOCOL MACHINES

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_PROG) 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_PROG).

Regardless of what other actions are taken, this UPM causes FSM_CONVERSATION (page 5.1-59) to enter the reset state.

INPUT: The RCB_ID of the still-allocated resource

OUTPUT: See above.

Not defined by SNA

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 5.0-5) 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 |

PIP_FIELD

Program Initialization Parameter (PIP) data is sent as a GDS variable immediately following the FMH-5 if the FMH-5 indicated that PIP data follows.

NOTES: 1. The value in the LL field includes the length of the LL field itself.

2. PIP subfields are type G symbol strings. Minimum send and receive support for PIP_SUBFIELD.DATA is 64 characters.

PIP_FIELD:

| | |
|---|---|
| LL: | Length of this logical record (See Note 1.). |
| ID: | GDS ID for PIP Data GDS Variable (X'12F5'). |
| DATA: | Character string containing PIP data. |

RETURN_CODE

The primary and secondary return codes that may be returned on transaction program verbs are described in _SNA Transaction Programmer's Reference Manual for LU Type 6.2_

RETURN_CODE

| | |
|---|---|
| PRIMARY_CODE: | possible values: see _SNA Transaction Programmer's Reference Manual for LU Type 6.2_ |
| SECONDARY_CODE: | possible values: see _SNA Transaction Programmer's Reference Manual for LU Type 6.2_ |

| PIP_LIST |
|---|

PIP_LIST: List of PIP data subfields.

| LU_ID |
|---|

LU_ID: ID of this LU.

| 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

# CHAPTER 5.1. PRESENTATION SERVICES--CONVERSATION VERBS

## GENERAL DESCRIPTION

A PS process handles requests for LU services. A transaction program execution instance (TP) 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 protocol boundary to user-application transaction programming, while leaving the lower-level basic conversation protocol boundary "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 may be in different processes, this interaction may occur by means of asynchronous inter-process 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. Since the TP may have active conversations with several partners simultaneously, PS.CONV may be interacting with a number of different half-session processes.

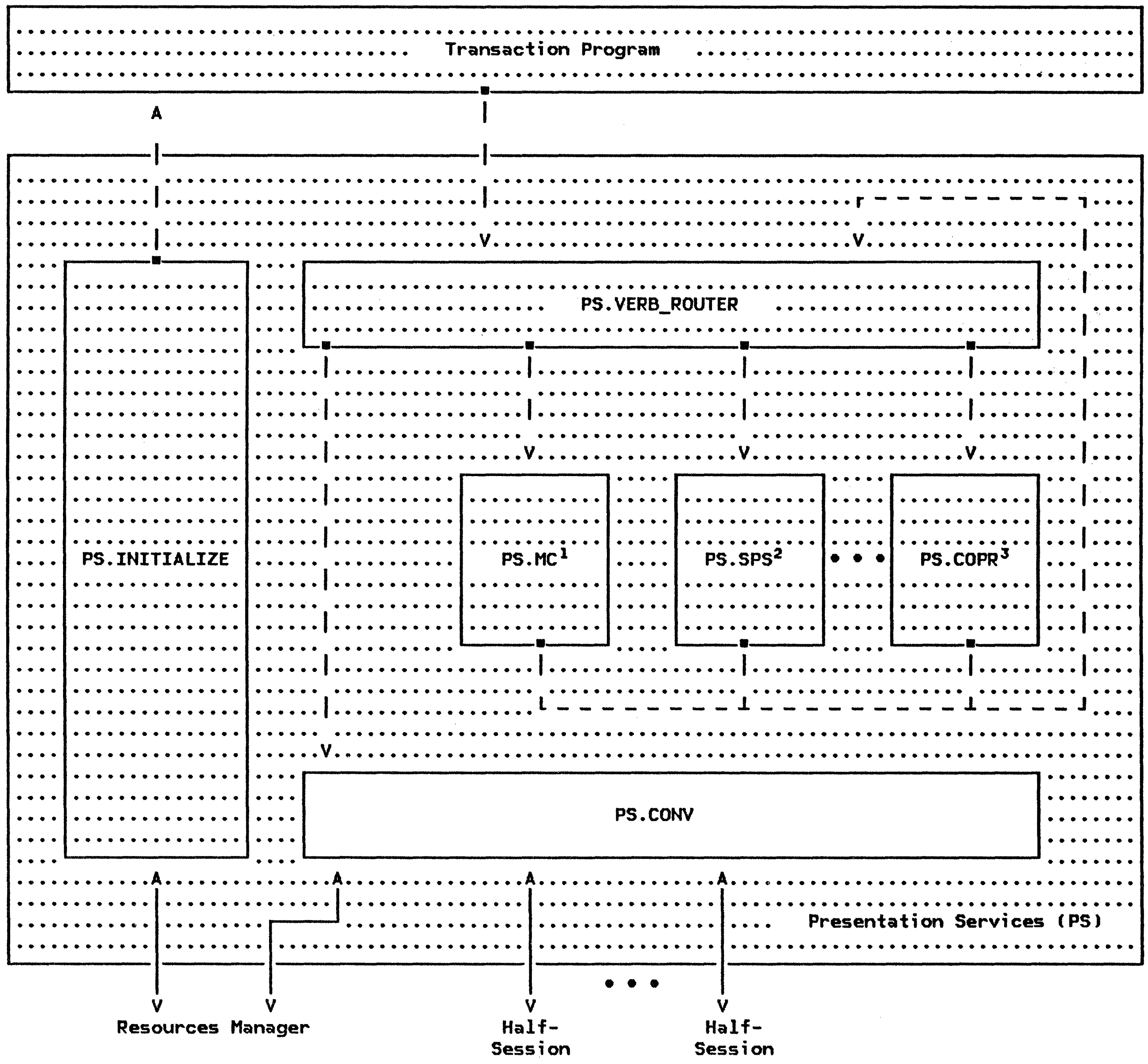
### PS.CONV DATA-BASE STRUCTURE

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.

#### LU Control Block (LUCB) and Associated Lists

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 fully 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 by the LU. The information in a TRANSACTION_PROGRAM_LIST entry includes the trans-



[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

action program name and whether it supports various optional features (e.g., sync point, mapped conversations).

Another list associated with the 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 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 fully qualified names of the partner LU.

Associated with each PARTNER_LU entry is a MODE_LIST (see Figure 5.1-2 on page 5.1-3), which has one entry for each mode name that is defined for the particular partner LU name. The MODE entry contains information that is fixed on a mode basis, such as the mode name.

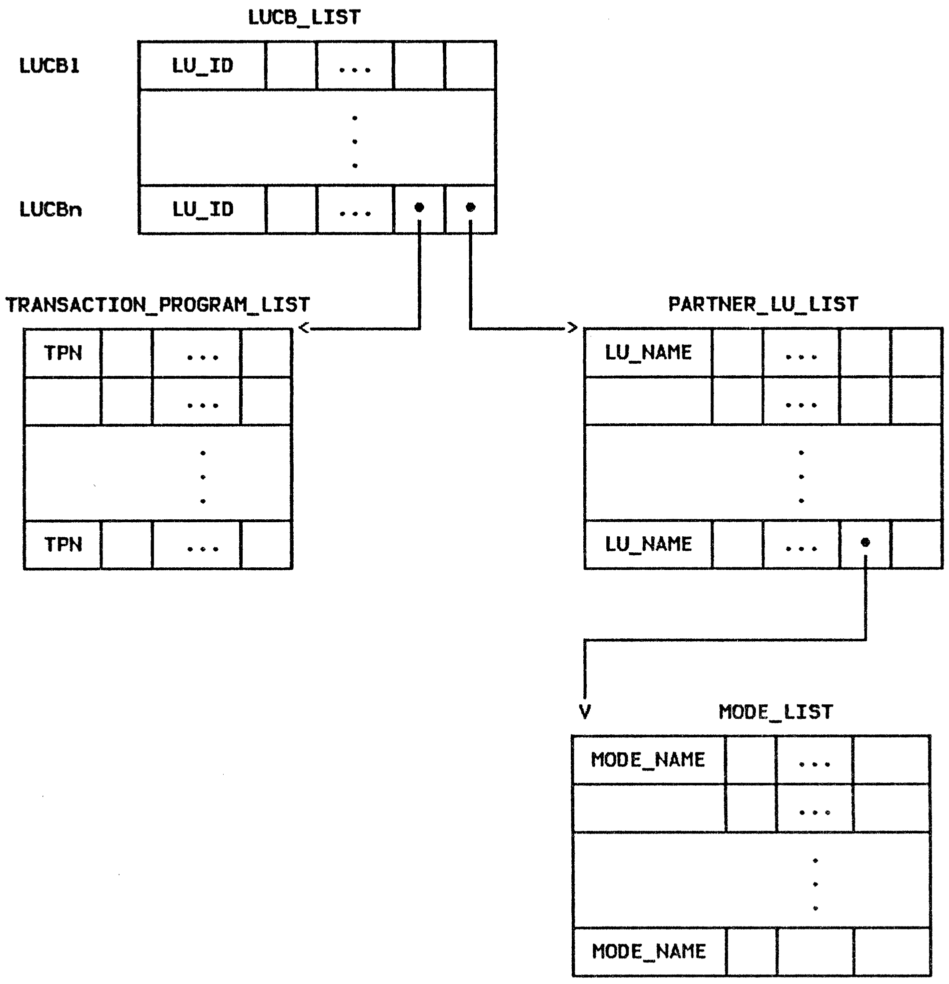


Figure 5.1-2. LU Control Block List and Associated Lists

## Transaction Control Block (TCB)

The transaction control block (TCB--see Figure 5.1-3 on page 5.1-4) contains information associated with the 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 servicing 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 transaction programs it manages, which PS process 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 associated with the transaction program.

## PS PROCESS DATA

PS_PROCESS_DATA (page 5.0-20) 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 and the pointer to the RCB_LIST. It is initialized by the root procedure of the PS process (page 5.0-5) from parameters received 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

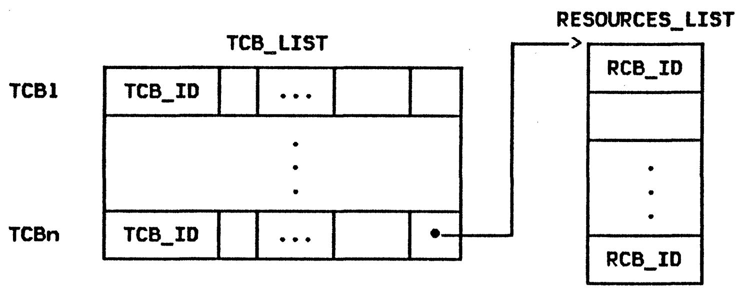


Figure 5.1-3. Transaction Control Block (TCB)

for the process contains, in its RESOURCES_LIST, the list of RCBs for resources allocated to the process.

An RCB 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 5.1-59) 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.

PS_TO_HS_RECORD (page A-24) is used as a buffer to contain data that has been generated by verb processing but that has not yet been sent to the half-session. The record is sent to the half-session when either a maximum size is reached or as the result of some transaction program verb (e.g., FLUSH, CONFIRM).

FSM_ERROR_OR_FAILURE (page 5.1-61) is a finite-state machine that stores error or failure records (which may arrive from RM or the half-session) until they can be returned to the TP in verb parameters.

HS_TO_PS_BUFFER_LIST contains a list of records that have been received from the half-session but not yet passed to the transaction program.

## 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 to the TP in the returned parameter fields. Each verb issuance has

1. an indicator of which verb is being issued (ALLOCATE, CONFIRM, etc.),
2. some supplied parameters, including (typically) an identifier of the conversation on which the verb is being issued, and
3. 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 whole 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 PS_TO_RM_RECORDs (page A-25) to RM and receives RM_TO_PS_RECORDs (page A-31) from RM. There are several types of PS_TO_RM records. Each contains a TCB_ID identifying the PS process that sent the record, and possibly additional fields. RM_TO_PS_RECORDs are usually sent in reply to a PS_TO_RM_RECORD request, as shown in Figure 5.1-5.

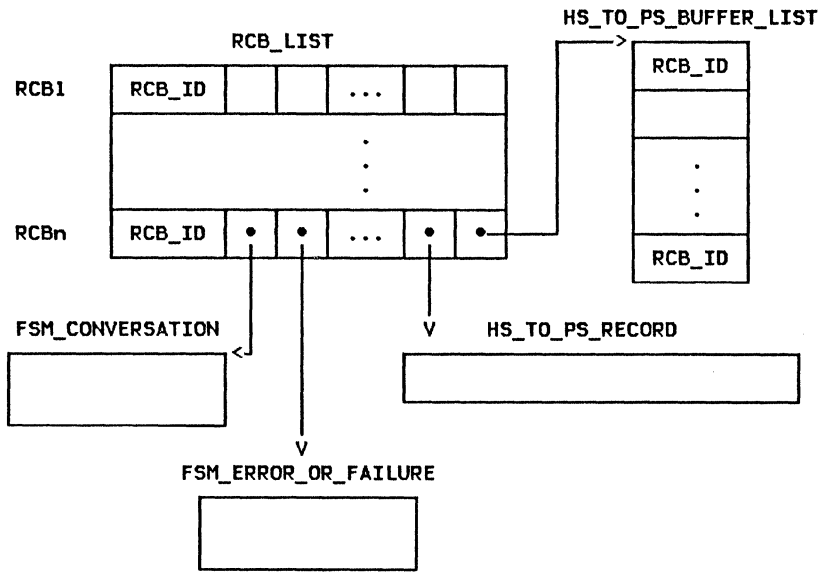


Figure 5.1-4. Resource Control Block (RCB)

| PS.CONV Request | RM Response |
|---|---|
| ALLOCATE_RCB | RCB_ALLOCATED |
| GET_SESSION | SESSION_ALLOCATED |
| DEALLOCATE_RCB | RCB_DEALLOCATED |

Figure 5.1-5. PS.CONV Requests and Associated RM Responses

The only exception is CONVERSATION_FAILURE, which is sent, unsolicited, to PS.CONV when a conversation being used by PS.CONV fails.

PS-HS RECORDS

PS.CONV sends PS_TO_HS_RECORDs (page A-24) to a half-session and receives HS_TO_PS_RECORDs (page A-12) from a half-session.

A PS_TO_HS_RECORD contains a VARIANT_NAME field, identifying the particular variant; 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 "Appendix D. RH 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 with a SEND_DATA_RECORD is buffered by PS.CONV until either a maximum data length (given by RCB.MAX_BUFFER_LENGTH) is reached, or the transaction program issues a verb that forces the data to be passed on for transmission (e.g., CONFIRM, RECEIVE_AND_WAIT, or DEALLOCATE).

The other PS_TO_HS_RECORD variants are sent 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.

The HS_TO_PS_RECORD contains a VARIANT_NAME field and an HS_ID field. The HS_ID is used to identify which half-session sent the record to PS.CONV. The HS_TO_PS_RECORD is symmetric to the PS_TO_HS_RECORD. That is, RECEIVE_DATA corresponds to a SEND_DATA_RECORD issued by the remote PS.CONV, CONFIRMED corresponds to CONFIRMED, RECEIVE_ERROR to SEND_ERROR, and REQUEST_TO_SEND to REQUEST_TO_SEND. RSP_TO_REQUEST_TO_SEND has no equivalent in the PS_TO_HS_RECORD, since RSP_TO_REQUEST_TO_SEND is generated internally to the remote half-session.

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 "Chapter 5.2. Presentation

Services--Mapped Conversation Verbs" in Chapter 5.2.) The value in the Length field includes the length of the field itself; thus the length value is normally in the range 2-32767. Length values of 0 and 1 are used to indicate a PS header (See "Chapter 5.3. Presentation Services--Sync Point Services Verbs" in Chapter 5.3 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.

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. PS.CONV enforces this rule by remembering how many bytes are remaining to be sent in the current logical record, and terminating the transaction program abnormally if this remainder is not 0 when the verb is issued. 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 on 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 with FILL(LL), PS uses the receive count remainder 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 5.1-59). 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 perform a number of checks on verb requests received from the transaction program. These include:

- Parameter checks, such as that:
  - The parameters specified on the ALLOCATE are supported by the LUs.
  - 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" on page 5.1-5).
- State checks, such as that:
  - The verb can be issued in the current conversation state (see "Maintaining and Checking the Basic Conversation State").
  - The transaction program has completed the current logical record, if necessary (see "Tracking Logical Record Length" on page 5.1-5).

### 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 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.

If DELAYED_ALLOCATION_PERMITTED is specified on the ALLOCATE, the session allocation request is delayed until either the PS.CONV send buffer is full or the transaction program issues a verb that causes the data to be passed on for transmission. Furthermore, PS.CONV instructs RM (via the GET_SESSION record) whether to bid for (request use of) the session with or without sending the buffered data. The bid is to be sent without data if the data buffered thus far would not require a definite response from the partner LU. Otherwise, the bid is to be sent with data.

PS.CONV creates an Attach FMH-5 based on the parameter settings on the ALLOCATE verb. The Attach is inserted in RCB.PS_TO_HS_RECORD.DATA, to be sent later.

## POST ON RECEIPT

POST_ON_RECEIPT 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, also in the RCB. Whenever PS.CONV receives information 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 now, 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, 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 not in send state:
  - Sending a SEND_ERROR record to the half-session. This causes a -RSP(0846) to be sent to the partner LU.
  - Waiting until EC is received from the partner LU. The half-session purges all data until EC is received.
- Creating an FMH-7 with the sense data based on the SEND_ERROR type and the current state of the conversation.
- Creating a log data variable, if log data is present.

-RSP(0846) is not sent and data is not purged if the conversation is in send state for the transaction program issuing the SEND_ERROR. In the case of both sides of a conversation issuing SEND_ERROR, the side that was in receive state always wins the SEND_ERROR race. 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 5.1-8 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 end-of-chain 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.

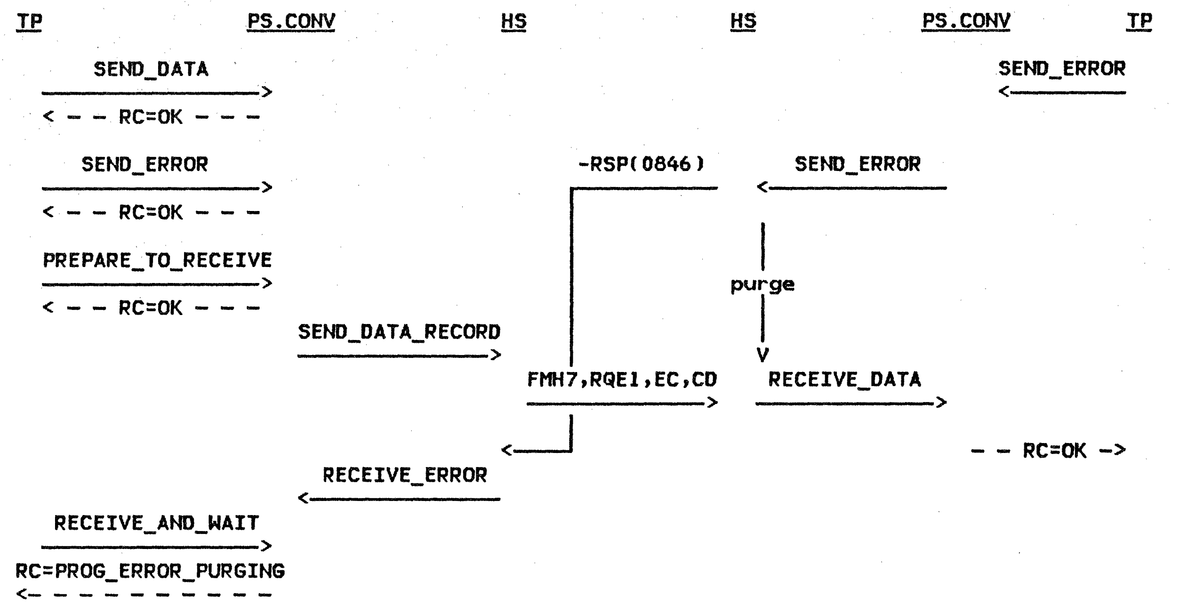


Figure 5.1-6. SEND_ERROR Race

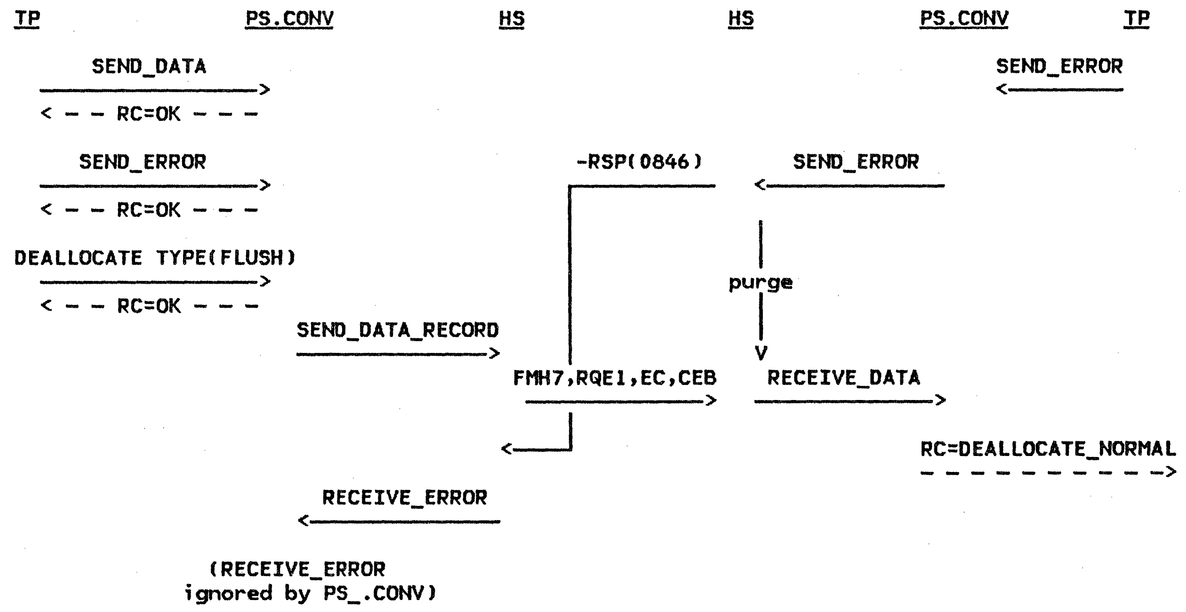


Figure 5.1-7. SEND_ERROR Race with Deallocation

PROTOCOL ERRORS

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 5.1-5 )
- 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 to indicate that a conversation failure (protocol error) occurred.

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 record, PS.CONV sets RCB.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 admits one.

## PS_CONV

| | |
|---|---|
| FUNCTION: | Receives conversation verbs issued by the TP or by other PS components, and calls appropriate procedures to process them. |
| INPUT: | Transaction program verb and parameters |
| OUTPUT: | Refer to the procedures that are called from this process for the specific outputs. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| ALLOCATE_PROC | page 5.1-11 |
| CONFIRM_PROC | page 5.1-12 |
| CONFIRMED_PROC | page 5.1-14 |
| DEALLOCATE_PROC | page 5.1-14 |
| FLUSH_PROC | page 5.1-16 |
| GET_ATTRIBUTES_PROC | page 5.1-17 |
| POST_ON_RECEIPT_PROC | page 5.1-17 |
| PREPARE_TO_RECEIVE_PROC | page 5.1-18 |
| RECEIVE_AND_WAIT_PROC | page 5.1-19 |
| REQUEST_TO_SEND_PROC | page 5.1-21 |
| SEND_DATA_PROC | page 5.1-22 |
| SEND_ERROR_PROC | page 5.1-24 |

```
Select based on the transaction program verb:
   When ALLOCATE
      Call ALLOCATE_PROC(verb parameters) (page 5.1-11).
   When CONFIRM
      Call CONFIRM_PROC(verb parameters) (page 5.1-12).
   When CONFIRMED
      Call CONFIRMED_PROC(verb parameters) (page 5.1-14).
   When DEALLOCATE
      Call DEALLOCATE_PROC(verb parameters) (page 5.1-14).
   When FLUSH
      Call FLUSH_PROC(verb parameters) (page 5.1-16).
   When GET_ATTRIBUTES
      Call GET_ATTRIBUTES_PROC(verb parameters) (page 5.1-17).
   When POST_ON_RECEIPT
      Call POST_ON_RECEIPT_PROC(verb parameters) (page 5.1-17).
   When PREPARE_TO_RECEIVE
      Call PREPARE_TO_RECEIVE_PROC(verb parameters) (page 5.1-18).
   When RECEIVE_AND_WAIT
      Call RECEIVE_AND_WAIT_PROC(verb parameters) (page 5.1-19).
   When REQUEST_TO_SEND
      Call REQUEST_TO_SEND_PROC(verb parameters) (page 5.1-21).
   When SEND_DATA
      Call SEND_DATA_PROC(verb parameters) (page 5.1-22).
   When SEND_ERROR
      Call SEND_ERROR_PROC(verb parameters) (page 5.1-24).
```

## ALLOCATE_PROC

| | |
|---|---|
| FUNCTION: | Handles allocation of new resources to the transaction program. <br><br> 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 DELAYED_ALLOCATION_PERMITTED or WHEN_SESSION_ALLOCATED, PS sends a separate session request to RM at a later time. |
| INPUT: | ALLOCATE verb with parameters; RCB_ALLOCATED record received from RM |
| OUTPUT: | ALLOCATE_RCB to RM |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| RM | page 3-17 |
| RCB_ALLOCATED_PROC | page 5.1-44 |
| WAIT_FOR_RM_REPLY | page 5.1-56 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| ALLOCATE_RCB | page A-25 |
| RCB_ALLOCATED | page A-32 |
| MODE | page A-3 |

```
Check ALLOCATE for ABEND conditions (see ALLOCATE verb in
 SNA Transaction Programmer's Reference Manual for LU Type 6.2).

If ABEND conditions found then
   Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).

Else
   If a MODE control block exists for the LU_NAME and MODE_NAME parameters specified in the
    ALLOCATE 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 5.1-56).
      Call RCB_ALLOCATED_PROC(RCB_ALLOCATED, ALLOCATE parameters),
       to build an FMH-5 Attach header, and to set the RETURN_CODE
       parameter to the appropriate value (page 5.1-44).

    Else
       Set RETURN_CODE of the ALLOCATE verb to PARAMETER_ERROR.
```

## CONFIRM_PROC

FUNCTION: 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 first 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: 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 CONFIRM to RESOURCE_FAILURE.

2. If the local LU has detected an error while attempting to allocate a session to this conversation, but PS has not yet had the opportunity to relay that information to the transaction program, it does so at this time by setting the RETURN_CODE parameter of the CONFIRM to reflect the type of allocation error.

3. If a RECEIVE_ERROR has been received from HS, PS sends a SEND_DATA record with the 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.

4. If there are no error or failure conditions, COMPLETE_CONFIRM_PROC (page 5.1-27) is called to complete the processing of the CONFIRM.

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| PROCESS_RM_OR_HS_TO_PS_RECORDS | page 5.1-43 |
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| POST_AND_WAIT_PROC | page 5.1-37 |
| DEQUEUE_FMH7_PROC | page 5.1-33 |
| COMPLETE_CONFIRM_PROC | page 5.1-27 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |

```
Find the RCB for the conversation identified in the RESOURCE parameter.

If RCB.SYNC_LEVEL = NONE and the send data is not at a logical record boundary then
   Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).
Else
   If executing FSM_CONVERSATION(S, CONFIRM, RCB)
    (page 5.1-59) would cause a state-check (>) condition then
      Execute the corresponding output code in the FSM.
   Else
      Call PROCESS_RM_OR_HS_TO_PS_RECORDS(RCB.RCB_ID, NO_SUSPEND) (page 5.1-43).

      Select based on the state of FSM_ERROR_OR_FAILURE:
         When CONV_FAILURE_PROTOCOL_ERROR
            Set RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
            Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
         When CONV_FAILURE_SON
            Set RETURN_CODE to RESOURCE_FAILURE_RETRY.
            Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
         When ALLOCATE_FAILURE_RETRY, ALLOCATE_FAILURE_NO_RETRY,
          or SYNCLEVEL_NOT_SUPPORTED
            Set RETURN_CODE to ALLOCATION_ERROR concatenated with
             ALLOCATION_FAILURE_RETRY, ALLOCATION_FAILURE_NO_RETRY, or
             SYNC_LEVEL_NOT_SUPPORTED_BY_LU, as appropriate.
            Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-59).
         When RCVD_ERROR
            Set RCB.PS_TO_HS_RECORD to PREPARE_TO_RCV_FLUSH.
            Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).
            Call POST_AND_WAIT_PROC(RCB, LL, X'7FFF') to post the
             resource when the whole FMH7 is received (page 5.1-37).
            Call DEQUEUE_FMH7_PROC(CONFIRM verb parameters, RCB) (page 5.1-33).
         When NO_RQS
            Call COMPLETE_CONFIRM_PROC(CONFIRM verb parameters, RCB) (page 5.1-27).

   If REQUEST_TO_SEND has been received but not reported to TP then
      Set returned REQUEST_TO_SEND_RECEIVED parameter to YES.
```

## CONFIRMED_PROC

| | | |
|---|---|---|
| FUNCTION: | | 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: | | See below. |
| 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 have a RETURN_CODE parameter, 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 with a RETURN_CODE parameter.) |
| | 2. | PS sends a CONFIRMED record (page A-24) to HS. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| HS | page 6.0-3 |
| PROCESS_RM_OR_HS_TO_PS_RECORDS | page 5.1-43 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |
| CONFIRMED | page A-24 |

```
Find the RCB for the conversation identified by the RESOURCE parameter.
If executing FSM_CONVERSATION(S, CONFIRMED, RCB) (page 5.1-59).
 would cause a state-check (>) condition then
   Execute the corresponding output code in the FSM.
Else
   Call PROCESS_RM_OR_HS_TO_PS_RECORD(RCB.RCB_ID, NO_SUSPEND)
    (page 5.1-43).
   If state of FSM_ERROR_OR_FAILURE is  NO_RQS then
      Call FSM_CONVERSATION(S, CONFIRMED, RCB) (page 5.1-59).
      Create a CONFIRMED record, initialize it, and send it to HS.
   Else (errors reported)
      Do nothing (see Note 1).
```

## DEALLOCATE_PROC

| | |
|---|---|
| FUNCTION: | 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 pertinent deallocation procedure is called. When appropriate, PS sends DEALLOCATE_RCB to RM. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| DEALLOCATE_FLUSH_PROC | page 5.1-32 |
| DEALLOCATE_CONFIRM_PROC | page 5.1-31 |
| DEALLOCATE_ABEND_PROC | page 5.1-30 |
| FSM_CONVERSATION | page 5.1-59 |
| DEALLOCATE_RCB | page A-26 |
| RCB | page A-7 |

```
Find the RCB for the conversation identified in the supplied RESOURCE
 parameter of the DEALLOCATE.
Select based on the following criteria:
   When TYPE parameter of DEALLOCATE is FLUSH, or TYPE parameter is SYNC_LEVEL
    and RCB.SYNC_LEVEL is NONE
      If executing FSM_CONVERSATION(S, DEALLOCATE_FLUSH, RCB) (page 5.1-59) would
       cause a state-check (>) condition then
         Execute the corresponding output code in the FSM.
      Else
         Call DEALLOCATE_FLUSH_PROC(DEALLOCATE verb parameters, RCB) (page 5.1-32).
         Purge all records from HS to PS process.
         Create DEALLOCATE_RCB, initialize it, and send it to RM.
   When TYPE parameter is  CONFIRM
      If executing FSM_CONVERSATION(S, DEALLOCATE_CONFIRM, RCB) (page 5.1-59) would
       cause a state-check (>) condition then
         Execute the corresponding output code in the FSM.
      Else
         If RCB.SYNC_LEVEL is CONFIRM or SYNCPT then
            Call DEALLOCATE_CONFIRM_PROC(DEALLOCATE verb parameters, RCB) (page 5.1-31).
         Else
            Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).
   When TYPE parameter is SYNC_LEVEL and RCB.SYNC_LEVEL = CONFIRM
      If executing FSM_CONVERSATION(S, DEALLOCATE_CONFIRM, RCB) (page 5.1-59) would
       cause a state-check (>) condition then
         Execute the corresponding output code in the FSM.
      Else
         Call DEALLOCATE_CONFIRM_PROC(DEALLOCATE, RCB) (page 5.1-31).
   When TYPE parameter is SYNC_LEVEL and RCB.SYNC_LEVEL = SYNCPT
      If executing FSM_CONVERSATION(S, DEALLOCATE_DEFER, RCB) (page 5.1-59) would
       cause a state-check (>) condition then
         Execute the corresponding output code in the FSM.
      Else
         If the data sent by TP is on a logical record boundary then
            Call FSM_CONVERSATION(S, DEALLOCATE_DEFER, RCB) (page 5.1-59).
            Set RETURN_CODE to OK.
         Else
            Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).
   When TYPE parameter is ABEND_PROG, ABEND_SVC, or ABEND_TIMER
      If executing FSM_CONVERSATION(S, DEALLOCATE_ABEND, RCB) (page 5.1-59) would
       cause a state-check (>) condition then
         Execute the corresponding output code in the FSM.
      Else
         Call DEALLOCATE_ABEND_PROC(DEALLOCATE verb parameters, RCB) (page 5.1-30).
         Purge all records from HS to PS process.
         Create DEALLOCATE_RCB, initialize it, and send it to RM.
   When TYPE parameter is LOCAL
      If executing FSM_CONVERSATION(S, DEALLOCATE_LOCAL, RCB) (page 5.1-59) would
       cause a state-check (>) condition then
         Execute the corresponding output code in the FSM.
      Else
         Call FSM_CONVERSATION(S, DEALLOCATE_LOCAL, RCB) (page 5.1-59).
         Set RETURN_CODE to OK.
         Purge all records from HS to PS process.
         Create DEALLOCATE_RCB record, initialize it, and send it to RM.
```

## FLUSH_PROC

| | | |
|---|---|---|
| FUNCTION: | | 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: | | See below. |
| 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 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 or allocation error has occurred, PS returns to the transaction program without sending any data to HS. Since FLUSH does not have a RETURN_CODE parameter, 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 has a RETURN_CODE field). |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| HS | page 6.0-3 |
| RM | page 3-17 |
| PROCESS_RM_OR_HS_TO_PS_RECORDS | page 5.1-43 |
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |
| DEALLOCATE_RCB | page A-26 |

```
Find RCB for the conversation identified by the RESOURCE parameter.
If executing FSM_CONVERSATION(S, FLUSH, RCB) (page 5.1-59) would
 cause a state-check (>) condition then
   Execute the corresponding output code in the FSM.

Else
   Call PROCESS_RM_OR_HS_TO_PS_RECORDS(RCB.RCB_ID, NO_SUSPEND) (page 5.1-43).

   If the state of FSM_ERROR_OR_FAILURE (page 5.1-61)
    is RCVD_ERROR or NO_RQS then

         Select based on state of FSM_CONVERSATION (page 5.1-59):
            When SEND
               Set RCB.PS_TO_HS_RECORD.TYPE to FLUSH.
            When PREP_TO_RCV_DEFER
               Set RCB.PS_TO_HS_RECORD.TYPE to PREPARE_TO_RCV_FLUSH.
            When DEALL_DEFER
               Set RCB.PS_TO_HS_RECORD.TYPE to DEALLOCATE_FLUSH.
         Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).
         If state of FSM_CONVERSATION is DEALL_DEFER then
            Send a DEALLOCATE_FLUSH record to HS.
            Purge all records from HS to this PS.
            Create DEALLOCATE_RCB, initialize it, and send it to RM.
         Call FSM_CONVERSATION(S, FLUSH, RCB) (page 5.1-59).
```

GET_ATTRIBUTES_PROC

| | |
|---|---|
| FUNCTION: | Handles requests for information about a conversation.<br><br>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 5.1-59 |
| LUCB | page A-1 |
| PARTNER_LU | page A-2 |
| RCB | page A-7 |

```
Find the RCB for the conversation identified in the supplied RESOURCE parameter.

Set the GET_ATTIBUTES returned parameters as follows:
   OWN_FULLY_QUALIFIED_LU_NAME to LUCB.FULLY_QUALIFIED_LU_NAME,
   PARTNER_LU_NAME to RCB.LU_NAME,
   PARTNER_FULLY_QUALIFIED_LU_NAME to PARTNER_LU.FULLY_QUALIFIED_LU_NAME,
   MODE_NAME to RCB.MODE_NAME,
   SYNC_LEVEL to RCB.SYNC_LEVEL.
Call FSM_CONVERSATION(S, GET_ATTRIBUTES, RCB) (page 5.1-59).
```

POST_ON_RECEIPT_PROC

| | |
|---|---|
| FUNCTION: | Performs the processing of the POST_ON_RECEIPT verb.<br><br>The 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: | Updated FSM_CONVERSATION, FSM_POST, and post conditions in the RCB |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PROCESS_RM_OR_HS_TO_PS_RECORDS | page 5.1-43 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_POST | page 5.1-62 |
| RCB | page A-7 |

```
Find the RCB for the conversation identified by the RESOURCE parameter.
If executing FSM_CONVERSATION(S, POST_ON_RECEIPT, RCB) (page 5.1-59).
 would cause a state-check (>) condition then
   Execute the corresponding output code in the FSM.

Else
   Call FSM_CONVERSATION(S, POST_ON_RECEIPT, RCB) (page 5.1-59).
   Call FSM_POST(POST_ON_RECEIPT) (page 5.1-62).
   Copy FILL and LENGTH parameters of the POST_ON_RECEIPT verb into RCB.POST_CONDITIONS.
   Call PROCESS_RM_OR_HS_TO_PS_RECORDS(RCB.RCB_ID, NO_SUSPEND) (page 5.1-43).
```

## PREPARE_TO_RECEIVE_PROC

| | |
|---|---|
| FUNCTION: | 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 (page 5.1-59) 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_FLUSH_PROC | page 5.1-39 |
| PREPARE_TO_RECEIVE_CONFIRM_PROC | page 5.1-38 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| FSM_CONVERSATION | page 5.1-59 |
| RCB | page A-7 |

```
Find the RCB for the conversation identified by the RESOURCE parameter.
If data sent by TP is on a logical record boundary then
   Select based on one of the following conditions:
      When TYPE parameter = FLUSH or (TYPE parameter = SYNC_LEVEL and the
       conversation sync level of the conversation is = NONE)
         If executing FSM_CONVERSATION(S, PREPARE_TO_RECEIVE_FLUSH, RCB) (page 5.1-59).
          would cause a state-check (>) condition then
            Execute the corresponding output code in the FSM.
         Else
            Call PREPARE_TO_RECEIVE_FLUSH_PROC(PREPARE_TO_RECEIVE parameters, RCB)
             (page 5.1-39).
      When TYPE parameter = CONFIRM
         If executing FSM_CONVERSATION(S, PREPARE_TO_RECEIVE_CONFIRM, RCB) (page 5.1-59).
          would cause a state-check (>) condition then
            Execute the corresponding output code in the FSM.
         Else
            If sync level of the conversation is CONFIRM or SYNCPT then
               Call PREPARE_TO_RECEIVE_CONFIRM_PROC(PREPARE_TO_RECEIVE parameters, RCB)
                (page 5.1-38).
            Else
               Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).
      When TYPE parameter = SYNC_LEVEL
         If executing FSM_CONVERSATION(S, PREPARE_TO_RECEIVE_DEFER, RCB) (page 5.1-59).
          would cause a state-check (>) condition then
            Execute the corresponding output code in the FSM.
         Else
            If sync level of the conversation is SYNCPT then
               Call FSM_CONVERSATION(S, PREPARE_TO_RECEIVE_DEFER, RCB) (page 5.1-59).
               Copy LOCKS parameter into RCB.
               Set RETURN_CODE parameter to OK.
            Else
               Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).
   Else
      Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).
```

## RECEIVE_AND_WAIT_PROC

FUNCTION: Handles the RECEIVE_AND_WAIT verb.

If the resource specified in the RECEIVE_AND_WAIT is valid and 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: 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 to RESOURCE_FAILURE.
2. If the local LU has detected an error while attempting to allocate a session to this conversation, but PS has not yet had the opportunity to relay that information to the transaction program, it does so at this time by setting the RETURN_CODE parameter to reflect the type of allocation error.
3. If a RECEIVE_ERROR record has been received from HS, PS sends a SEND_DATA_RECORD with the TYPE field set to PREPARE_TO_RCV_FLUSH to HS. (Any data in the RCB data 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 is set based on the sense data carried in the FMH-7.
4. If the conversation is in the SEND state, PS sends a SEND_DATA_RECORD with the TYPE field set to PREPARE_TO_RCV_FLUSH to HS. All data in the RCB send buffer is placed in the SEND_DATA_RECORD. 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:

| | |
|---|---|
| PROCESS_RM_OR_HS_TO_PS_RECORDS | page 5.1-43 |
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| POST_AND_WAIT_PROC | page 5.1-37 |
| DEQUEUE_FMH7_PROC | page 5.1-33 |
| PERFORM_RECEIVE_PROCESSING | page 5.1-36 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RECEIVE_ERROR | page A-12 |
| SEND_DATA_RECORD | page A-24 |
| RCB | page A-7 |

```
Find RCB for the resource identified in the RESOURCE parameter.
If executing FSM_CONVERSATION(S,RECEIVE_AND_WAIT, RCB) would
 cause a state-check (>) condition then
   Execute the corresponding output code in the FSM.

Else
   Call PROCESS_RM_OR_HS_TO_PS_RECORDS(RCB.RCB_ID, NO_SUSPEND) (page 5.1-43).

   Select based on the state of FSM_ERROR_OR_FAILURE (page 5.1-61):
      When CONV_FAILURE_PROTOCOL_ERROR
         Set RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
         Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
      When CONV_FAILURE_SON
         Set RETURN_CODE to RESOURCE_FAILURE_RETRY.
         Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
      When ALLOCATE_FAILURE_RETRY | ALLOCATE_FAILURE_NO_RETRY | SYNCLEVEL_NOT_SUPPORTED
         Set RETURN_CODE to ALLOCATION_ERROR concatenated with
          ALLOCATION_FAILURE_RETRY, ALLOCATION_FAILURE_NO_RETRY, or
          SYNC_LEVEL_NOT_SUPPORTED_BY_LU, as appropriate.
         Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-59).
```

```
            When RCVD_ERROR
               If state of FSM_CONVERSATION = SEND then
                  Set RCB.PS_TO_HS_RECORD type to PREPARE_TO_RCV_FLUSH.
                  Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).
                  Call POST_AND_WAIT_PROC(RCB, LL, X'7FFF') (page 5.1-37).
                  Call DEQUEUE_FMH7_PROC(RECEIVE_AND_WAIT, RCB) (page 5.1-33).
               Else
                  Call POST_AND_WAIT_PROC(RCB, LL, X'7FFF') (page 5.1-37).
                  Call DEQUEUE_FMH7_PROC(RECEIVE_AND_WAIT, RCB) (page 5.1-33).
            When NO_RQS
               Call FSM_CONVERSATION(S, RECEIVE_AND_WAIT, RCB) (page 5.1-59).
               If state of FSM_ERROR_OR_FAILURE is ALLOCATE_FAILURE_RETRY,
                ALLOCATE_FAILURE_NO_RETRY, OR SYNCLEVEL_NOT_SUPPORTED then
                  Set RETURN_CODE to ALLOCATION_ERROR concatenated with
                   ALLOCATION_FAILURE_RETRY, ALLOCATION_FAILURE_NO_RETRY, or
                   to SYNC_LEVEL_NOT_SUPPORTED_BY_LU, as appropriate.
                  Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-59).
               Else
                  Call POST_AND_WAIT_PROC with RCB, FILL and LENGTH parameters (page 5.1-37).
                  Call PERFORM_RECEIVE_PROCESSING(RCB, RECEIVE_AND_WAIT parameters) (page 5.1-36).
         If REQUEST_TO_SEND has been received but not reported to TP then
            Set REQUEST_TO_SEND_RECEIVED parameter to YES.
```

## REQUEST_TO_SEND_PROC

| | | |
|---|---|---|
| FUNCTION: | | Handles REQUEST_TO_SEND verb processing.<br><br>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: | | See below. |
| NOTES: | 1. | Since REQUEST_TO_SEND does not have a RETURN_CODE parameter, 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 has a RETURN_CODE parameter). |
| | 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:

| | |
|---|---|
| PS | page 5.0-5 |
| HS | page 6.0-3 |
| PROCESS_RM_OR_HS_TO_PS_RECORDS | page 5.1-43 |
| WAIT_FOR_RSP_TO_RQ_TO_SEND_PROC | page 5.1-57 |
| FSM_CONVERSATION | page 5.1-59 |
| RCB | page A-7 |
| REQUEST_TO_SEND | page A-24 |

```
Find RCB for the resource identifier in REQUEST_TO_SEND.
If executing FSM_CONVERSATION(S, REQUEST_TO_SEND, RCB) (page 5.1-59) would
 cause a state-check (>) condition then
   Execute the corresponding output code in the FSM.
Else
   Call PROCESS_RM_OR_HS_TO_PS_RECORDS(RCB.RCB_ID, NO_SUSPEND) (page 5.1-43).
   If Change Direction (CD) indication has not been received from HS then
      Send a REQUEST_TO_SEND record to the HS.
      Call WAIT_FOR_RSP_TO_RQ_TO_SEND_PROC(RCB) (page 5.1-57).
```

## SEND_DATA_PROC

FUNCTION: 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

OUTPUT: See below.

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 the local LU has detected an error while attempting to allocate a session to this conversation, but PS has not yet had the opportunity to relay that information to the transaction program, it does so at this time by setting the RETURN_CODE parameter of the SEND_DATA to reflect the type of allocation error.

3. If a RECEIVE_ERROR has been received from HS, PS sends a SEND_DATA with the TYPE field set to PREPARE_TO_RCV_FLUSH to HS. (Any data in the RCB data 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 of the SEND_DATA is set based on the sense data carried in the FMH-7.

4. 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.

5. If no session has been allocated to this conversation (i.e., the ALLOCATE that allocated the conversation specified RETURN_CONTROL = DELAYED_ALLOCATION_PERMITTED), PS now requests a session from the resources manager. If, while attempting to allocate a session, the local LU detects an error, PS sets the RETURN_CODE field in the SEND_DATA to reflect the type of allocation error and returns control to the transaction program.

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PROCESS_RM_OR_HS_TO_PS_RECORDS | page 5.1-43 |
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| POST_AND_WAIT_PROC | page 5.1-37 |
| DEQUEUE_FMH7_PROC | page 5.1-33 |
| SEND_DATA_BUFFER_MANAGEMENT | page 5.1-47 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |
| CONVERSATION_FAILURE | page A-32 |
| RECEIVE_ERROR | page A-12 |

```
Find the RCB for the resource identified in the RESOURCE parameter.
If executing FSM_CONVERSATION(S, SEND_DATA, RCB) (page 5.1-59)
 would cause a state-check (>) condition then
   Execute the corresponding output code in the FSM.

Else
   Call PROCESS_RM_OR_HS_TO_PS_RECORDS(RCB.RCB_ID, NO_SUSPEND) (page 5.1-43).

   Select based on state of FSM_ERROR_OR_FAILURE (page 5.1-61):
      When CONV_FAILURE_SON (see Note 1)
         Set RETURN_CODE to RESOURCE_FAILURE_RETRY.
         Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
      When CONV_FAILURE_PROTOCOL_ERROR (see Note 1)
         Set RETURN_CODE to RESOURCE_FAILURE_NO_RETRY.
         Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
   When ALLOCATE_FAILURE_RETRY, ALLOCATE_FAILURE_NO_RETRY, or
    SYNCLEVEL_NOT_SUPPORTED (see Note 2)
      Set RETURN_CODE to ALLOCATION_ERROR concatenated with
       ALLOCATION_FAILURE_RETRY, ALLOCATION_FAILURE_NO_RETRY, or
       SYNC_LEVEL_NOT_SUPPORTED_BY_LU, as appropriate.
      Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-59).
   When RCVD_ERROR (see Note 3)
      Set RCB.PS_TO_HS_RECORD type to PREPARE_TO_RCV_FLUSH.
      Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).
      Call POST_AND_WAIT_PROC(RCB, LL, X'7FFF') (page 5.1-37).
      Call DEQUEUE_FMH7_PROC(RECEIVE_AND_WAIT, RCB) (page 5.1-33).
   When NO_RQS
      Perform the LL processing (see Note 4).
       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 Appendix H) then
          Call DEALLOCATION_CLEANUP_ABEND (page 5.0-14).
      Call SEND_DATA_BUFFER_MANAGEMENT (page 5.1-47)
       with the first LENGTH bytes of the DATA (see SEND_DATA verb parameters) and RCB.

      If the state of FSM_ERROR_OR_FAILURE (page 5.1-61)
       is ALLOCATE_FAILURE_RETRY, ALLOCATE_FAILURE_NO_RETRY, or
       SYNCLEVEL_NOT_SUPPORTED (see Note 5) then
         Set RETURN_CODE to ALLOCATION_ERROR concatenated with
          ALLOCATION_FAILURE_RETRY, ALLOCATION_FAILURE_NO_RETRY, or
          SYNC_LEVEL_NOT_SUPPORTED_BY_LU, as appropriate.
         Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-59).
      Else
         Set RETURN_CODE to OK.

If REQUEST_TO_SEND has been received but not reported to TP then
   Set REQUEST_TO_SEND_RECEIVED return parameter to YES.
```

## SEND_ERROR_PROC

FUNCTION: 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: 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 SEND_ERROR to RESOURCE_FAILURE.
2. If the local LU has detected an error while attempting to allocate a session to this conversation, but PS has not yet had the opportunity to relay that information to the transaction program, it does so at this time by setting the RETURN_CODE parameter of the SEND_ERROR to reflect the type of allocation error.
3. 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:

| | |
|---|---|
| PS | page 5.0-5 |
| HS | page 6.0-3 |
| PROCESS_RM_OR_HS_TO_PS_RECORDS | page 5.1-43 |
| SEND_ERROR_IN_SEND_STATE | page 5.1-51 |
| SEND_ERROR_DONE_PROC | page 5.1-49 |
| SEND_ERROR_IN_RECEIVE_STATE | page 5.1-50 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |
| BUFFER_ELEMENT | page A-8 |
| SEND_ERROR | page A-24 |

```
Find the RCB for the resource identifier for SEND_ERROR.
If executing FSM_CONVERSATION(S, SEND_ERROR, RCB) (page 5.1-59)
 would cause a state-check (>) condition then
   Execute the corresponding output code in the FSM.

Else
   Call PROCESS_RM_OR_HS_TO_PS_RECORDS(RCB.RCB_ID, NO_SUSPEND) (page 5.1-43).

   Select based on state of FSM_ERROR_OR_FAILURE:
      When CONV_FAILURE_SON (see Note 1)
         Set RETURN_CODE of SEND_ERROR verb to RESOURCE_FAILURE_RETRY.
         Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
      When CONV_FAILURE_PROTOCOL_ERROR (see Note 1)
         Set RETURN_CODE of SEND_ERROR verb to RESOURCE_FAILURE_NO_RETRY.
         Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
      When ALLOCATE_FAILURE_RETRY, ALLOCATE_FAILURE_NO_RETRY, or
       SYNCLEVEL_NOT_SUPPORTED (see Note 2)
         Set RETURN_CODE of SEND_ERROR verb to ALLOCATION_ERROR concatenated with
          ALLOCATION_FAILURE_RETRY, ALLOCATION_FAILURE_NO_RETRY, or
          SYNC_LEVEL_NOT_SUPPORTED_BY_LU, as appropriate.
         Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-59).
   When NO_RQS or RCVD_ERROR (see Note 3)

      Select based on the state of FSM_CONVERSATION (page 5.1-59):
         When SEND
            Call SEND_ERROR_IN_SEND_STATE(SEND_ERROR parameters, RCB) (page 5.1-51).
         When RCVD_CONFIRM, RCVD_CONFIRM_SEND, or RCVD_CONFIRM_DEALL
            Send SEND_ERROR record to HS.
            Call FSM_CONVERSATION(S, SEND_ERROR, RCB) (page 5.1-59).
            Call SEND_ERROR_DONE_PROC(SEND_ERROR, RCB) (page 5.1-49).
         When RCV
            Call SEND_ERROR_IN_RECEIVE_STATE(SEND_ERROR parameters, RCB) (page 5.1-50).

Remove all entries in the RCB.HS_TO_PS_BUFFER_LIST.
If REQUEST_TO_SEND has been received but not reported to TP then
   Set REQUEST_TO_SEND_RECEIVED parameter of SEND_ERROR verb to YES.
```

## TEST_PROC

| | |
|---|---|
| FUNCTION: | 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. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PROCESS_RM_OR_HS_TO_PS_RECORDS | page 5.1-43 |
| POST_AND_WAIT_PROC | page 5.1-37 |
| DEQUEUE_FMH7_PROC | page 5.1-33 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| FSM_POST | page 5.1-62 |
| TEST | page 5.1-63 |
| RCB | page A-7 |
| BUFFER_ELEMENT | page A-8 |

```
Find the RCB for the resource identified in the RESOURCE field of the TEST record.
If executing FSM_CONVERSATION(S, TEST, RCB) (page 5.1-59)
 would cause a state-check (>) condition then
   Execute the corresponding output code in the FSM.
Else
   Call PROCESS_RM_OR_HS_TO_PS_RECORDS(RCB.RCB_ID, NO_SUSPEND) (page 5.1-43).
   Select based on test type (recorded in TEST.TEST):
      When POSTED
         If state of FSM_POST = RESET then
            Set RETURN_CODE of TEST to POSTING_NOT_ACTIVE.
         Else
            Select based on the state of FSM_ERROR_OR_FAILURE:
               When CONV_FAILURE_SON
                  Set RETURN_CODE of TEST to RESOURCE_FAILURE_RETRY.
                  Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
               When CONV_FAILURE_PROTOCOL_ERROR
                  Set RETURN_CODE of TEST to RESOURCE_FAILURE_NO_RETRY.
                  Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
               When ALLOCATE_FAILURE_RETRY, ALLOCATE_FAILURE_NO_RETRY, or
                SYNCLEVEL_NOT_SUPPORTED
                  Set RETURN_CODE of TEST to ALLOCATION_ERROR concatenated with
                  ALLOCATION_FAILURE_RETRY, ALLOCATION_FAILURE_NO_RETRY, or
                   SYNC_LEVEL_NOT_SUPPORTED_BY_LU respectively.
                  Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-59).
               When RCVD_ERROR
                  Call POST_AND_WAIT_PROC(RCB, LL, X'77FF') (page 5.1-37).
                  Call DEQUEUE_FMH7_PROC(TEST, RCB) (page 5.1-33).
              When NO_RQS
                  Select on state of FSM_POST:
                     When PEND_POSTED
                        Set RETURN_CODE of TEST to UNSUCCESSFUL.
                     When POSTED
                        Set RETURN_CODE of TEST to OK concatenated to NOT_DATA
                         or DATA as the RCB.HS_TO_PS_BUFFER_LIST is or is not empty.

         Call FSM_CONVERSATION(S, TEST, RCB) (page 5.1-59).
         Call FSM_POST(TEST) (page 5.1-62).
      When REQUEST_TO_SEND_RECEIVED
         If REQUEST_TO_SEND has been received but not reported to TP then
            Set RETURN_CODE of TEST to OK.
            Record as reported to TP the REQUEST_TO_SEND.
         Else
            Set RETURN_CODE to UNSUCCESSFUL.
```

COMPLETE_CONFIRM_PROC

| | | |
|---|---|---|
| FUNCTION: | | Completes the processing of a CONFIRM verb.<br><br>It is called by CONFIRM_PROC (page 5.1-12) when there are no error or failure conditions indicated by FSM_ERROR_OR_FAILURE (page 5.1-61). 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: | | See below. |
| NOTES: | 1. | If FSM_CONVERSATION is in the SEND state, a SEND_DATA_RECORD with TYPE field set to CONFIRM is sent to HS. |
| | 2. | If FSM_CONVERSATION is in the PREPARE_TO_RECEIVE_DEFER state, a SEND_DATA_RECORD with TYPE field set to PREPARE_TO_RCV_CONFIRM is sent to HS. |
| | 3. | If FSM_CONVERSATION is in the DEALLOCATE_DEFER state, a SEND_DATA_RECORD with TYPE field set to DEALLOCATE_CONFIRM is sent to HS. |
| | 4. | If no session has been allocated to this conversation (i.e., the ALLOCATE verb issued to allocate the conversation specified RETURN_CONTROL = DELAYED_ALLOCATION_PERMITTED), PS now requests a session from the resources manager. If, while attempting to allocate a session, the local LU detects an error, PS sets the RETURN_CODE parameter of the CONFIRM to reflect the type of allocation error and returns control to the transaction program. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| WAIT_FOR_CONFIRMED_PROC | page 5.1-55 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |
| SEND_DATA_RECORD | page A-24 |

```
Select based on the state of FSM_CONVERSATION (page 5.1-59):
   When SEND (see Note 1.)
      Set RCB.PS_TO_HS_RECORD.TYPE to CONFIRM.
   When PREP_TO_RCV_DEFER (see Note 2)
      Set RCB.PS_TO_HS_RECORD.TYPE to PREPARE_TO_RCV_CONFIRM_SHORT or
       PREPARE_TO_RCV_CONFIRM_LONG as indicated by RCB.LOCKS.
   When DEALL_DEFER (see Note 3)
      Set RCB.PS_TO_HS_RECORD.TYPE to DEALLOCATE_CONFIRM.
Call FSM_CONVERSATION(S, CONFIRM, RCB) (page 5.1-59).
Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).

If state of FSM_ERROR_OR_FAILURE is ALLOCATE_FAILURE_RETRY,
 ALLOCATE_FAILURE_NO_RETRY, OR SYNCLEVEL_NOT_SUPPORTED (see Note 4) then
   Set RETURN_CODE to ALLOCATION_ERROR concatenated with
    ALLOCATION_FAILURE_RETRY, ALLOCATION_FAILURE_NO_RETRY, or
    to SYNC_LEVEL_NOT_SUPPORTED_BY_LU, as appropriate.
   Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-59).

Else
   Call WAIT_FOR_CONFIRMED_PROC(CONFIRM parameters, RCB) (page 5.1-55).
```

## COMPLETE_DEALLOCATE_ABEND_PROC

| | |
|---|---|
| FUNCTION: | Completes the processing of a DEALLOCATE verb that specifies TYPE = ABEND.<br><br>PS creates an FMH-7 and places it in the RCB send buffer. The FMH-7 carries sense data indicating DEALLOCATE_ABEND. If there is any log data associated with the DEALLOCATE, PS creates an Error Log GDS variable (see "Appendix H. FM Header and LU Services Commands") and places it in the RCB buffer 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 a SEND_DATA_RECORD, 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: | SEND_DATA to HS. Any log data supplied with the DEALLOCATE is logged. |
| NOTE: | If no session has been allocated to this conversation (i.e., the ALLOCATE that allocated the conversation specified RETURN_CONTROL = DELAYED_ALLOCATION_PERMITTED), PS now requests a session from the resources manager. If, while attempting to allocate a session, the local LU detects an error, PS sets the RETURN_CODE parameter in the DEALLOCATE to reflect the type of allocation error and returns control to the transaction program. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| SEND_DATA_BUFFER_MANAGEMENT | page 5.1-47 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |
| SEND_DATA_RECORD | page A-24 |

```
Set SENSE_DATA based on the DEALLOCATE type as follows:
 set to X'08640000' if ABEND_PROG, to X'08640001' if ABEND_SVC, or
 to X'8640002'if ABEND_TIMER.
Set CONTINUE to true.

If state of FSM_CONVERSATION (page 5.1-59)
 is SEND, PREP_TO_RCV_DEFER, or DEALL_DEFER
 and RCB.PS_TO_HS_RECORD.SEND_PARM.DATA is not null then
   Set RCB.PS_TO_HS_RECORD.TYPE to FLUSH._
   Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).
   If state of FSM_ERROR_OR_FAILURE (page 5.1-61)
    is ALLOCATE_FAILURE_RETRY, ALLOCATE_FAILURE_NO_RETRY, or SYNCLEVEL_NOT_SUPPORTED then
      Set CONTINUE to false.

If CONTINUE then
   If LOG_DATA parameter has been supplied then
      Create FMH-7 with log data indicator and SENSE_DATA.
      Set RCB.HS_TO_PS_RECORD.DATA to this FMH-7.
      Create Error Log GDS variable and concatenate it to
       RCB.PS_TO_HS_RECORD.DATA.
      Log it in the system error log.

    Else
      Create FMH-7 with SENSE_DATA  but no log data.
      Set RCB.HS_TO_PS_RECORD.DATA to this FMH-7.
    Call SEND_DATA_BUFFER_MANAGEMENT(null data, RCB) (page 5.1-47).
    Set RCB_PS_TO_HS record type to DEALLOCATE_FLUSH.
    Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).
```

## CONVERSATION_FAILURE_PROC

| | |
|---|---|
| FUNCTION: | 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. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| FSM_POST | page 5.1-62 |
| CONVERSATION_FAILURE | page A-32 |
| RCB | page A-7 |

```
Find the RCB for the CONVERSATION_FAILURE record.
If CONVERSATION_FAILURE.REASON = PROTOCOL_VIOLATION then
   Call FSM_ERROR_OR_FAILURE (page 5.1-61) and
    pass it a CONV_FAIL_PROTOCOL signal.

Else
   Call FSM_ERROR_OR_FAILURE (page 5.1-61)
    and pass it a CONV_FAIL_SON signal.

If state of FSM_POST is PEND_POSTED then
   Call FSM_POST(POST) (page 5.1-62).
```

## DEALLOCATE_ABEND_PROC

| | |
|---|---|
| FUNCTION: | Invoked when the TYPE parameter of DEALLOCATE verb is ABEND_PROG, ABEND_SVC, or ABEND_TIMER.<br><br>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 record 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 record) is created and sent to HS, or no output is created. All elements are purged from the HS_TO_PS_BUFFER_LIST before returning to the transaction program. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| HS | page 6.0-3 |
| PROCESS_RM_OR_HS_TO_PS_RECORDS | page 5.1-43 |
| WAIT_FOR_SEND_ERROR_DONE_PROC | page 5.1-58 |
| COMPLETE_DEALLOCATE_ABEND_PROC | page 5.1-28 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |
| SEND_ERROR | page A-24 |

```
Call PROCESS_RM_OR_HS_TO_PS_RECORDS(RCB.RCB_ID, NO_SUSPEND) (page 5.1-43).

If the state of FSM_ERROR_OR_FAILURE (page 5.1-61) is
 NO_RQS or RCVD_ERROR then

   Select based on the state of FSM_CONVERSATION (page 5.1-59):
      When RCV
         If the first entry of RCB.HS_TO_PS_BUFFER_LIST is not DEALLOCATE_FLUSH then
            Send SEND_ERROR record to HS.
            Call WAIT_FOR_SEND_ERROR_DONE_PROC(DEALLOCATE parameters, RCB)
             (page 5.1-58).
      When RCVD_CONFIRM | RCVD_CONFIRM_SEND | RCVD_CONFIRM_DEALL
         Send SEND_ERROR record to HS.
         Call COMPLETE_DEALLOCATE_ABEND_PROC(DEALLOCATE parameters, RCB)
          (page 5.1-28).
      When SEND | PREP_TO_RCV_DEFER | DEALL_DEFER
         Call COMPLETE_DEALLOCATE_ABEND_PROC(DEALLOCATE parameters, RCB)
         (page 5.1-28).

Purge all buffers in HS_TO_PS_BUFFER_LIST.
Set RETURN_CODE to OK.
Call FSM_CONVERSATION(S, DEALLOCATE_ABEND) (page 5.1-59).
```

DEALLOCATE_CONFIRM_PROC

| | | |
|---|---|---|
| FUNCTION: | | Invoked when DEALLOCATE TYPE(SYNC_LEVEL) is issued for a conversation whose SYNC_LEVEL is CONFIRM.<br><br>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 the local LU has detected an error while attempting to allocate a session to this conversation, but PS has not yet had the opportunity to relay that information to the transaction program, it does so at this time by setting the RETURN_CODE parameter of the DEALLOCATE to reflect the type of allocation error. |
| | 3. | If a RECEIVE_ERROR has been received from HS, PS sends a SEND_DATA_RECORD with the 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 DEALLOCATE is set based on the sense data carried in the FMH-7. |
| | 4. | If no error or failure condition has occurred, PS sends a SEND_DATA_RECORD with the TYPE field set to DEALLOCATE_CONFIRM to HS. |
| | 5. | If no session has been allocated to this conversation (i.e., the ALLOCATE that allocated the conversation specified RETURN_CONTROL = DELAYED_ALLOCATION_PERMITTED), PS now requests a session from the resources manager. If, while attempting to allocate a session, the local LU detects an error, PS sets the RETURN_CODE parameter of the DEALLOCATE to reflect the type of allocation error and returns control to the transaction program. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PROCESS_RM_OR_HS_TO_PS_RECORDS | page 5.1-43 |
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| POST_AND_WAIT_PROC | page 5.1-37 |
| DEQUEUE_FMH7_PROC | page 5.1-33 |
| WAIT_FOR_CONFIRMED_PROC | page 5.1-55 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |
| SEND_DATA_RECORD | page A-24 |

```
If data sent by TP is not at a logical record boundary then
  Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).

Else
  Call FSM_CONVERSATION(S, DEALLOCATE_CONFIRM, RCB) (page 5.1-59).
  Call PROCESS_RM_OR_HS_TO_PS_RECORDS(RCB.RCB_ID, NO_SUSPEND) (page 5.1-43).

  Select based on the state of FSM_ERROR_OR_FAILURE (see Note 1):
     When CONV_FAILURE_PROTOCOL_ERROR
        Set RETURN_CODE of DEALLOCATE to RESOURCE_FAILURE_NO_RETRY.
        Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
     When CONV_FAILURE_SON
        Set RETURN_CODE of DEALLOCATE to RESOURCE_FAILURE_RETRY.
        Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
     When ALLOCATE_FAILURE_RETRY, ALLOCATE_FAILURE_NO_RETRY, or
      SYNCLEVEL_NOT_SUPPORTED (see Note 2)
        Set RETURN_CODE of DEALLOCATE to ALLOCATION_ERROR concatenated with
         ALLOCATION_FAILURE_RETRY, ALLOCATION_FAILURE_NO_RETRY, or
         SYNC_LEVEL_NOT_SUPPORTED_BY_LU, as appropriate.
        Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-59).
```

DEALLOCATE_CONFIRM_PROC

```
    When RCVD_ERROR (see Note 3)
       Set RCB.PS_TO_HS_RECORD type to PREPARE_TO_RCV_FLUSH.
       Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).
       Call POST_AND_WAIT_PROC(RCB, LL, X'7FFF') to post the resource
        when the whole FMH7 is received (page 5.1-37).
       Call DEQUEUE_FMH7_PROC(RECEIVE_AND_WAIT, RCB) (page 5.1-33).
    When NO_RQS (see Note 4)
       Set RCB.PS_TO_HS_RECORD type to PREPARE_TO_RCV_FLUSH.
       Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).

       If state of FSM_ERROR_OR_FAILURE is ALLOCATE_FAILURE_RETRY,
        ALLOCATE_FAILURE_NO_RETRY, or SYNCLEVEL_NOT_SUPPORTED then
          Set RETURN_CODE of DEALLOCATE to ALLOCATION_ERROR concatenated with
           ALLOCATION_FAILURE_RETRY, ALLOCATION_FAILURE_NO_RETRY, or
           SYNC_LEVEL_NOT_SUPPORTED_BY_LU, as appropriate.
          Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-59).
          Call WAIT_FOR_CONFIRMED_PROC(DEALLOCATE parameters, RCB) (page 5.1-55).
```

DEALLOCATE_FLUSH_PROC

| | |
|---|---|
| FUNCTION: | Invoked when a DEALLOCATE is received that specifies TYPE = FLUSH, or TYPE = SYNC_LEVEL and the SYNC_LEVEL of the conversation is NONE.<br><br>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: | See below. |
| NOTES: | 1. If a RECEIVE_ERROR was received from HS, or no error records have been received, PS sends any data remaining in the RCB send buffer to HS with the TYPE field of the SEND_DATA_RECORD set to DEALLOCATE_FLUSH. (If a RECEIVE_ERROR was received, any data in PS's buffer has already been purged.)<br><br>2. If CONVERSATION_FAILURE record has been received from RM, or if an allocation error has been detected by the local LU, no further records are sent to HS. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PROCESS_RM_OR_HS_TO_PS_RECORDS | page 5.1-43 |
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |
| SEND_DATA_RECORD | page A-24 |
| RECEIVE_ERROR | page A-12 |

```
If the data sent by TP is not at a logical record boundary then
   Call DEALLOCATION_CLEANUP_PROC (page 5.0-14).

Else
   Call PROCESS_RM_OR_HS_TO_PS_RECORDS(RCB.RCB_ID, NO_SUSPEND) (page 5.1-43).
   If state of FSM_ERROR_OR_FAILURE is RCVD_ERROR or NO_RQS (see Note 1) then
      Set RCB.PS_TO_HS record type to DEALLOCATE_FLUSH.
      Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).
   Else (see Note 2)
      Do nothing.
   Set RETURN_CODE of DEALLOCATE to OK.
   Call FSM_CONVERSATION(S, DEALLOCATE_FLUSH, RCB) (page 5.1-59).
```

## DEQUEUE_FMH7_PROC

| | |
|---|---|
| FUNCTION: | Invoked upon receipt of a RECEIVE_ERROR from HS. The next element expected in the HS_TO_PS_BUFFER_LIST is an FMH-7 buffer element. 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 buffer element). If the next element is not an FMH-7 buffer element, the partner LU has violated the protocol and, as an implementation-dependent option, the session over which the protocol violation occurred is deactivated. |
| INPUT: | The transaction program verb parameters currently being processed and the RCB corresponding to the resource specified in parameters of the verb |
| OUTPUT: | The RETURN_CODE parameter is set to reflect the sense data carried in the FMH-7 buffer element. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PROCESS_FMH7_PROC | page 5.1-42 |
| PS_PROTOCOL_ERROR | page 5.0-16 |
| FSM_POST | page 5.1-62 |
| RCB | page A-7 |
| BUFFER_ELEMENT | page A-8 |

```
Call FSM_POST(RECEIVE_IMMEDIATE) (page 5.1-62).
If first entry in RCB.HS_TO_PS_BUFFER_LIST is FMH-7 then
   Remove the first entry of RCB.HS_TO_PS_BUFFER_LIST.
   Call PROCESS_FMH7_PROC(RCB, BUFFER_ELEMENT.DATA, TP verb parameters)
    (page 5.1-42).
   Set RCB.RECEIVE_LL_REMAINDER to 0.
Else (as an implementation-dependent option)
   Call PS_PROTOCOL_ERROR with X'1008201D' for Request Error,
    FMH-7, and Associated Data Mismatch (page 5.0-16).
```

## GET_END_CHAIN_FROM_HS

FUNCTION: Invoked after PS sends a SEND_ERROR record to HS (as a result of a SEND_ERROR or DEALLOCATE (TYPE = ABEND_PROG, ABEND_SVC, ABEND_TIMER) issued for the conversation while it is in the receive state). This procedure waits for a RECEIVE_DATA whose TYPE field indicates EC to arrive from HS. TYPE values that indicate EC are CONFIRM, PREPARE_TO_RCV_CONFIRM, PREPARE_TO_RCV_FLUSH, DEALLOCATE_CONFIRM, and DEALLOCATE_FLUSH.

INPUT: The RCB corresponding to the conversation for which the EC is desired

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 EC.
2. If a RECEIVE_ERROR record is received, no action is taken. PS continues to wait for the EC 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_*).
3. When PS sends SEND_ERROR to HS, it begins to purge any data it receives from HS until a record indicating EC is received.

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| RECEIVE_RM_OR_HS_TO_PS_RECORD | page 5.1-47 |
| CONVERSATION_FAILURE_PROC | page 5.1-29 |
| RCB | page A-7 |
| BUFFER_ELEMENT | page A-8 |
| RECEIVE_DATA | page A-12 |

```
If the type of BUFFER_ELEMENT in the RCB.HS_TO_PS_BUFFER_LIST is
 CONFIRM, PREPARE_TO_RCV_CONFIRM, PREPARE_TO_RCV_FLUSH,
 DEALLOCATE_CONFIRM, or DEALLOCATE_FLUSH then
   Set EC_HAS_ARRIVED to true.
Else
   Set EC_HAS_ARRIVED to false.

Set NOT_CONVERSATION_FAILURE to true.
Do while EC_HAS_ARRIVED not true and NOT_CONVERSATION_FAILURE is true:
   Call RECEIVE_RM_OR_HS_TO_PS_RECORD(RCB.RCB_IS, SUSPEND) to receive
    record (page 5.1-47) to receive RECORD.
  If RECORD arrived from RM then
     Call CONVERSATION_FAILURE_PROC with RECORD (page 5.1-29).
     Set NOT_CONVERSATION_FAILURE to false.
  If RECORD arrived from HS then
     Select based on the RECORD type received:
        When REQUEST_TO_SEND (see Note 1)
           Set RCB.RQ_TO_SEND_RCVD to YES.
        When RECEIVE_ERROR (see Note 2)
           Do nothing.
        When RECEIVE_DATA (see Note 3)
           If RECEIVE_DATA type is CONFIRM, PREPARE_TO_RCV_CONFIRM,
            PREPARE_TO_RCV_FLUSH, DEALLOCATE_CONFIRM, or DEALLOCATE_FLUSH then
              Enqueue RECORD to RCB.HS_TO_PS_BUFFER_LIST.
              Set EC_HAS_ARRIVED to false.
```

## OBTAIN_SESSION_PROC

| | |
|---|---|
| FUNCTION: | Handles the acquisition of a session for use by a conversation resource. <br><br> This procedure sends a GET_SESSION record to the resources manager and waits for a SESSION_ALLOCATED reply. <br><br> PS can instruct RM to send the RCB send buffer containing the FMH-5 (Attach) for this conversation to HS when both of the following conditions hold: <br><br> • The transaction program has issued DEALLOCATE, PREPARE_TO_RECEIVE, and/or CONFIRM. <br> • No data has as yet been sent to HS (i.e., the data sent by the transaction program to PS has not been of sufficient quantity to cause PS's send buffer to overflow). <br><br> This situation can occur only if the ALLOCATE that caused this conversation to be initiated specified RETURN_CONTROL = DELAYED_ALLOCATION_PERMITTED. <br><br> If the allocation of a session fails, the reason is saved in FSM_ERROR_OR_FAILURE. PS informs the transaction program at the earliest opportunity of the failure with an allocation error return code of the appropriate kind. |
| INPUT: | The RCB corresponding to the conversation that is to use the obtained session, and an ATTACH/NO_ATTACH indicator (specifying whether RM is to send the send buffer containing the Attach to HS as it acquires the session) are passed as parameters to this procedure. SESSION_ALLOCATED is received from RM. |
| OUTPUT: | GET_SESSION is sent to RM |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| WAIT_FOR_RM_REPLY | page 5.1-56 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |
| GET_SESSION | page A-26 |
| SESSION_ALLOCATED | page A-33 |

```
Copy TCB_ID and RCB_ID from RCB into GET_SESSION record.
Set GET_SESSION.BID_INDICATOR to ATTACH or NO_ATTACH to agree with the input indicator.
Send GET_SESSION request to RM.
Call WAIT_FOR_RM_REPLY (page 5.1-56) to receive SESSION_ALLOCATED.
Select based on SESSION_ALLOCATED.RETURN_CODE:
   When OK
      Do nothing.
   When UNSUCCESSFUL_NO_RETRY
      Call FSM_ERROR_OR_FAILURE (page 5.1-61)
       and pass it an ALLOCATE_FAIL_NO_RETRY signal.
   When SYNC_LEVEL_NOT_SUPPORTED
      Call FSM_ERROR_OR_FAILURE (page 5.1-61)
       and pass it a SYNCLEVEL_NOT_SUPPTD signal.
```

## PERFORM_RECEIVE_PROCESSING

| | | |
|---|---|---|
| FUNCTION: | | Checks the appropriate 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.<br><br>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 entry in the RCB_LIST corresponding to the resource specified in the verb parameters, and RECEIVE verb parameters |
| OUTPUT: | | Various parameters are updated, depending on the type of information contained in the receive buffer. The information is removed from the HS_TO_PS_BUFFER_LIST after being placed in RECEIVE_VERB. |
| NOTES: | 1. | PS 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 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. |
| | 3. | Logical record processing begins anew following receipt of an FMH-7. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS_PROTOCOL_ERROR | page 5.0-16 |
| PROCESS_FMH7_PROC | page 5.1-42 |
| PROCESS_DATA_PROC | page 5.1-40 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| FSM_POST | page 5.1-62 |
| RCB | page A-7 |
| BUFFER_ELEMENT | page A-8 |

```
Call FSM_POST(RECEIVE_IMMEDIATE) (page 5.1-61),
 to reset posting, if activated.
If RCB.HS_TO_PS_BUFFER_LIST is not empty then
   Set BUFFER_ELEMENT to first entry of the list.

   If partner sent CONFIRM, DEALLOCATE_CONFIRM, DEALLOCATE_FLUSH,
    PREPARE_TO_RCV_CONFIRM, or PREPARE_TO_RCV_FLUSH
    before completing sending of the logical record, or
    partner sent CONFIRM, PREPARE_TO_RCV_CONFIRM, or DEALLOCATE on a
    conversation with SYNC_LEVEL = NONE then (as an implementation-dependent option)
      Call PS_PROTOCOL_ERROR (page 5.0-16)
       with X'10010000' for RU Data Error.

   Else
      Remove BUFFER_ELEMENT from the list.

      Select based on BUFFER_ELEMENT type:
         When CONFIRM
            Set RETURN_CODE parameter to OK.
            Set WHAT_RECEIVED parameter CONFIRM.
            Call FSM_CONVERSATION(R, CONFIRM_INDICATOR, RCB) (page 5.1-59).
         When PREPARE_TO_RCV_CONFIRM
            Set RETURN_CODE parameter to OK.
            Set WHAT_RECEIVED parameter to CONFIRM_SEND.
            Call FSM_CONVERSATION(R, CONFIRM_SEND_INDICATOR, RCB) (page 5.1-59).
```

```
When PREPARE_TO_RCV_FLUSH
   Set RETURN_CODE parameter to OK.
   Set WHAT_RECEIVED parameter to SEND.
   Call FSM_CONVERSATION(R, SEND_INDICATOR, RCB) (page 5.1-59).
When DEALLOCATE_CONFIRM
   Set RETURN_CODE parameter to OK.
   Set WHAT_RECEIVED parameter to CONFIRM_DEALLOCATE.
   Call FSM_CONVERSATION(R, CONFIRM_DEALLOCATE_INDICATOR, RCB) (page 5.1-59).
When  DEALLOCATE_FLUSH
   Set RETURN_CODE parameter to DEALLOCATE_NORMAL.
   Call FSM_CONVERSATION(R, CONFIRM_DEALLOCATE_NORMAL_RC, RCB) (page 5.1-59).
When FMH7
   Call PROCESS_FMH7_PROC(RCB, BUFFER_ELEMENT.DATA, RECEIVE verb parameters)
    (page 5.1-42).
When DATA
   Call PROCESS_DATA_PROC(RCB, BUFFER_ELEMENT.DATA, RECEIVE verb parameters)
    (page 5.1-40).
   If length of BUFFER_ELEMENT.DATA > 0 then
      Insert BUFFER_ELEMENT at the beginning of the RCB.HS_TO_PS_BUFFER_LIST.
```

## POST_AND_WAIT_PROC

| | |
|---|---|
| FUNCTION: | Establishes post conditions for a resource and waits for information to arrive from HS to cause those post conditions to be satisfied. |
| INPUT: | The RCB corresponding to the resource to be posted, a FILL indicator specifying whether data is to be received independent of its logical record format (FILL = BUFFER versus LL), and the length of the maximum amount of data that is to be received |
| OUTPUT: | The post conditions are satisfied on return to the calling procedure. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| TEST_FOR_POST_SATISFIED | page 5.1-54 |
| PROCESS_RM_OR_HS_TO_PS_RECORDS | page 5.1-43 |
| FSM_POST | page 5.1-62 |
| RCB | page A-7 |

```
Call FSM_POST(POST_ON_RECEIPT) (page 5.1-62).
Set RCB.POST_CONDITIONS.FILL to supplied FILL parameter.
Set RCB.POST_CONDITIONS.MAX_LENGTH to the supplied LENGTH parameter.
Call TEST_FOR_POST_SATISFIED(RCB) (page 5.1-54).

Do while state of FSM_POST ≠ POSTED:
   Call PROCESS_RM_OR_HS_TO_PS_RECORDS(RCB.RCB_ID, SUSPEND) (page 5.1-43).
```

## PREPARE_TO_RECEIVE_CONFIRM_PROC

| | | |
|---|---|---|
| FUNCTION: | | 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: | | 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 PREPARE_TO_RECEIVE verb to RESOURCE_FAILURE. |
| | 2. | If a RECEIVE_ERROR has been received from HS, PS sends a SEND_DATA_RECORD with the 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 PREPARE_TO_RECEIVE verb is set based on the sense data carried in the FMH-7. |
| | 3. | If no error or failure condition has occurred, PS sends a SEND_DATA record with the TYPE field set to PREPARE_TO_RCV_CONFIRM to HS and waits for a CONFIRMED reply. |
| | 4. | If no session has been allocated to this conversation (i.e., the ALLOCATE that allocated the conversation specified RETURN_CONTROL = DELAYED_ALLOCATION_PERMITTED), PS now requests a session from the resources manager. If, while attempting to allocate a session, the local LU detects an error, PS sets the RETURN_CODE field in the PREPARE_TO_RECEIVE to reflect the type of allocation error and returns control to the transaction program. |
| | 5. | If the local LU has detected an error while attempting to allocate a session to this conversation, but PS has not yet had the opportunity to relay that information to the transaction program, it does so at this time by setting the RETURN_CODE parameter of the PREPARE_TO_RECEIVE to reflect the type of allocation error. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PROCESS_RM_OR_HS_TO_PS_RECORDS | page 5.1-43 |
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| POST_AND_WAIT_PROC | page 5.1-37 |
| DEQUEUE_FMH7_PROC | page 5.1-33 |
| WAIT_FOR_CONFIRMED_PROC | page 5.1-55 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |
| SEND_DATA_RECORD | page A-24 |
| RECEIVE_ERROR | page A-12 |

```
Call FSM_CONVERSATION(S, PREPARE_TO_RECEIVE_CONFIRM, RCB) (page 5.1-59).
Call PROCESS_RM_OR_HS_TO_PS_RECORDS(RCB.RCB_ID, NO_SUSPEND) (page 5.1-43).

Select based on state of FSM_ERROR_OR_FAILURE (page 5.1-61):
   When CONV_FAILURE_PROTOCOL_ERROR (see Note 1)
      Set RETURN_CODE of PREPARE_TO_RECEIVE to RESOURCE_FAILURE_NO_RETRY.
      Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
   When CONV_FAILURE_SON (see Note 1)
      Set RETURN_CODE of PREPARE_TO_RECEIVE to RESOURCE_FAILURE_RETRY.
      Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
   When RCVD_ERROR (see Note 2)
      Set PS_TO_HS_RECORD.TYPE to PREPARE_TO_RCV_FLUSH.
      Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).
      Call POST_AND_WAIT_PROC(RCB, LL, X'7FFF') to receive the whole
       FMH7 (page 5.1-37).
      Call DEQUEUE_FMH7_PROC(PREPARE_TO_RECEIVE parameters, RCB) (page 5.1-33).
```

```
When NO_RQS (see Note 3):
   If LOCKS supplied parameter is SHORT then
      Set RCB.PS_TO_HS.TYPE to PREPARE_TO_RCV_CONFIRM_SHORT.

   Else
      Set RCB.PS_TO_HS.TYPE to PREPARE_TO_RCV_CONFIRM_LONG.
   Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).
   If state of FSM_ERROR_OR_FAILURE (page 5.1-61) is
    ALLOCATE_FAILURE_RETRY | ALLOCATE_FAILURE_NO_RETRY | SYNCLEVEL_NOT_SUPPORTED
     (see Note 4) then
      Set RETURN_CODE of PREPARE_TO_RECEIVE to ALLOCATION_ERROR concatenated
       with ALLOCATION_FAILURE_RETRY, ALLOCATION_FAILURE_NO_RETRY,
       or with SYNC_LEVEL_NOT_SUPPORTED_BY_LU, as appropriate.
      Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-59).

   Else
      Call WAIT_FOR_CONFIRMED_PROC(PREPARE_TO_RECEIVE parameters, RCB) (page 5.1-55).
When ALLOCATE_FAILURE_RETRY | ALLOCATE_FAILURE_NO_RETRY | SYNCLEVEL_NOT_SUPPORTED
   Set RETURN_CODE of PREPARE_TO_RECEIVE to ALLOCATION_ERROR concatenated
    with ALLOCATION_FAILURE_RETRY, ALLOCATION_FAILURE_NO_RETRY,
    or with SYNC_LEVEL_NOT_SUPPORTED_BY_LU, as appropriate.
   Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-59).
```

## PREPARE_TO_RECEIVE_FLUSH_PROC

FUNCTION: 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. See below for additional output.

NOTES:
1. If a RECEIVE_ERROR record has been received from HS, or no error records have been received, PS sends any data remaining in the RCB send buffer to HS with the TYPE indicator set to PREPARE_TO_RCV_FLUSH. (If a RECEIVE_ERROR was received, any data in PS's send buffer has already been purged.)
2. If a locally detected allocation error (i.e., an allocation error detected by the local LU) or 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:

| | |
|---|---|
| PROCESS_RM_OR_HS_TO_PS_RECORDS | page 5.1-43 |
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |
| RECEIVE_ERROR | page A-12 |
| PS_TO_HS_RECORD | page A-24 |

```
Call PROCESS_RM_OR_HS_TO_PS_RECORDS(RCB.RCB_ID, NO_SUSPEND) (page 5.1-43).

If the state of FSM_ERROR_OR_FAILURE (page 5.1-61) is RCVD_ERROR or NO_RQS then
   Set RCB.PS_TO_HS_RECORD.TYPE to PREPARE_TO_RECEIVE_FLUSH.
   Call SEND_DATA_TO_HS_PROC with RCB (page 5.1-48)
Set RETURN_CODE of PREPARE_TO_RECEIVE to OK.
Call FSM_CONVERSATION(S, PREPARE_TO_RECEIVE_FLUSH, RCB) (page 5.1-59).
```

## PROCESS_DATA_PROC

FUNCTION: Handles the processing of a DATA buffer element from the HS_TO_PS_BUFFER_LIST.

The procedure first checks to see if the data at the beginning of the buffer is a PS header or a logical record having an invalid LL value, in order to take appropriate action.

If the data at the beginning of the buffer is not a PS header or an invalid LL, further processing of the DATA buffer element occurs, as described below.

INPUT: The RCB corresponding to the resource specified in the passed RECEIVE verb parameters, the DATA buffer element from the HS_TO_PS_BUFFER_LIST, and RECEIVE verb parameters.

OUTPUT: The RETURN_CODE and WHAT_RECEIVED parameters of the RECEIVE verb are updated.

NOTES:

1. If the data in the passed BUFFER_DATA 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 BUFFER_DATA, data is moved from the BUFFER_DATA parameter to the DATA parameter of the passed RECEIVE verb.

2. If the data in the passed BUFFER_DATA begins on a logical record boundary, but only the first byte of the next 2-byte LL field is present in BUFFER_DATA, 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 record.) 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 BUFFER_DATA does not begin on a conversation 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 BUFFER_DATA to the DATA parameter of the passed RECEIVE verb.

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS_SPS | page 5.3-20 |
| PS_PROTOCOL_ERROR | page 5.0-16 |
| RECEIVE_DATA_PROCESSING | page 5.1-46 |
| RCB | page A-7 |

```
Select based on the following conditions:
   When BUFFER_DATA is the beginning of a logical
    record and is a PS header
      If RCB.SYNC_LEVEL = SYNCPT then
         Call PS_SPS (page 5.3-20).
      Else (as an implementation-dependent option)
         Call PS_PROTOCOL_ERROR (page 5.0-16)
          with X'10010000' for RU Data Error
   When BUFFER_DATA is the beginning of a logical record and
    has an invalid LL (as an implementation-dependent option)
      Call PS_PROTOCOL_ERROR (page 5.0-16)
       with X'10010000' for RU Data Error.
   Otherwise
      Select based on the following conditions:
         When BUFFER_DATA is the beginning of a logical
          record and its length is greater than 1
            Call RECEIVE_DATA_PROCESSING (RCB, BUFFER_DATA, RECEIVE verb parameters)
             (page 5.1-46).
```

```
When BUFFER_DATA is the beginning of a logical record and
 its length is 1 (i.e., LL field possibly split at buffer boundaries):
   If HS_TO_PS_BUFFER_LIST is not empty then
      If buffer in the HS_TO_PS_BUFFER_LIST is of FMH-7 type then
         Set RETURN_CODE of the RECEIVE verb to OK.
         If RCB.POST_CONDITIONS.MAX_LENGTH > 0 then
            Set DATA of RECEIVE verb to BUFFER_DATA.
            Set BUFFER_DATA to null value.
            Set LENGTH of RECEIVE verb to 1.
         If RCB.POST_CONDITIONS.FILL = BUFFER then
            Set WHAT_RECEIVED of receive verb to DATA.

         Else
            Set WHAT_RECEIVED of RECEIVE verb to DATA_INCOMPLETE.

      Else (optional installation check)
         Call PS_PROTOCOL_ERROR (page 5.0-16)
          with X'10010000' for RU Data Error.
   Else (i.e., buffer list is empty)
      Set RETURN_CODE of RECEIVE verb to UNSUCCESSFUL.
When BUFFER_DATA is the continuation of a logical record partially
 already received:
   Call RECEIVE_DATA_PROCESSING(RCB.BUFFER_DATA, RECEIVE verb parameters)
    (page 5.1-46).
```

## PROCESS_FMH7_PROC

| | |
|---|---|
| FUNCTION: | Invoked upon encountering an FMH-7 buffer element in the HS_TO_PS_BUFFER_LIST. <br><br> 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 received FMH-7, 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. |
| NOTE: | This error occurs when the FMH-7 specifies that log data follows, but no log data is present. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS_PROTOCOL_ERROR | page 5.0-16 |
| POST_AND_WAIT_PROC | page 5.1-37 |
| PERFORM_RECEIVE_PROCESSING | page 5.1-36 |
| SET_FMH7_RC | page 5.1-53 |
| FSM_CONVERSATION | page 5.1-59 |
| RCB | page A-7 |

```
As an implementation-dependent option perform receive check of the FMH-7.
If error found then
   Call PS_PROTOCOL_ERROR (page 5.0-16)
    with X'10086000' (Request Error--FMH Length Incorrect) or with
    X'1008200E' (Request Error--Invalid Concatenation Bit).
Set RCB.RECEIVE_LL_REMAINDER to 0.
If Error Log GDS variable follows then
   Call POST_AND_WAIT_PROC(RCB, LL, X'7FFF') (page 5.1-37)
    to get the whole GDS variable.
   Create and initialize RECEIVE_AND_WAIT verb parameters.  Set the
    RESOURCE parameter to RCB.RCB_ID, FILL to LL, and LENGTH to X'7FFF'.
   Call PERFORM_RECEIVE_PROCESSING(RCB, RECEIVE_AND_WAIT parameters) (page 5.1-36).
   If RETURN_CODE of RECEIVE_AND_WAIT is OK and WHAT_RECEIVED
    is DATA_COMPLETE then
      Insert error data into system error log.
   Else (as an implementation-dependent option)
      Call PS_PROTOCOL_ERROR (page 5.0-16)
       with X'1008201D' (log data is expected but absent).
If sense data is DEALLOCATE_ABEND then
   Call PROCESS_RM_OR_HS_TO_PS_RECORDS (page 5.1-43)
    with RCB_ID and SUSPEND, and remove DEALLOCATE buffer from RCB.HS_TO_PS_BUFFER_LIST.
    If no DEALLOCATE_FLUSH or DEALLOCATE_CONFIRM is found then
       Call PS_PROTOCOL_ERROR (page 5.0-16) with X'1008201D'.
If the state of FSM_CONVERSATION (page 5.1-59) = SEND then
   Set RCB.SEND_LL_REMAINDER to 0.
   Set RCB.SEND_LL_BYTE of RCB to NOT_PRESENT.
Call SET_FMH7_RC(RCB, FMH-7, transaction program verb parameters) (page 5.1-53).
DECLARE TEMP_DATA CHARACTER(MAX_INTEGER) VARYING;
```

## PROCESS_RM_OR_HS_TO_PS_RECORDS

| | |
|---|---|
| FUNCTION: | Processes records received from RM and HS for the conversation identified by RCB_ID. If posting has been activated for the conversation, the TEST_FOR_POST_SATISFIED procedure is called to determine whether the post conditions have been satisfied by the newly received information. |
| INPUT: | RCB_ID, the ID of the conversation and SUSPEND_FLAG. If SUSPEND_FLAG = SUSPEND, this procedure waits for at least one record to be received from RM or HS. |
| OUTPUT: | The RCB associated with each received record is updated to record the receipt of the record. |
| NOTES: | 1. The only records that PS can receive from RM here are CONVERSATION_FAILURE records. |
| | 2. RECEIVE_DATA is enqueued in the appropriate RCB.HS_TO_PS_BUFFER_LIST. For other HS_TO_PS_RECORDs, an indication that the record was received is stored in the appropriate RCB. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| HS | page 6.0-3 |
| RM | page 3-17 |
| RECEIVE_RM_OR_HS_TO_PS_RECORD | page 5.1-47 |
| CONVERSATION_FAILURE_PROC | page 5.1-29 |
| PS_PROTOCOL_ERROR | page 5.0-16 |
| TEST_FOR_POST_SATISFIED | page 5.1-54 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| FSM_POST | page 5.1-62 |
| RCB | page A-7 |
| RM_TO_PS_RECORD | page A-31 |
| HS_TO_PS_RECORD | page A-12 |

```
Call RECEIVE_RM_OR_HS_TO_PS_RECORD(RCB_ID, SUSPEND_FLAG) (page 5.1-47)
 to receive RECORD.
Do while RECORD is not null:

   Select based on the origin of the record:
       When origin is RM
          Call CONVERSATION_FAILURE_PROC (page 5.1-29)
           with RECORD.
       When origin is HS

          Select based on RECORD type:
             When REQUEST_TO_SEND
                Record that a request to send was received
                 on this conversation.
             When RECEIVED_ERROR
                Call FSM_ERROR_OR_FAILURE(RECEIVE_ERROR, RCB) (page 5.1-61).
             When RECEIVE_DATA
                If state of FSM_CONVERSATION is RCV or
                 state of FSM_ERROR_OR_FAILURE (page 5.1-61) is RCVD_ERROR then
                   Insert the record into RCB.HS_TO_PS_LIST.

                Else (as an implementation-dependent option)
                   Call PS_PROTOCOL_ERROR (page 5.0-16)
                    with X'20040000' for State Error--Direction.

   Call RECEIVE_RM_OR_HS_TO_PS_RECORD(RCB_ID, SUSPEND_FLAG) (page 5.1-47)
    and receive RECORD.

If state of FSM_POST (page 5.1-62) is PEND_POSTED then
   Call TEST_FOR_POST_SATISFIED(RCB) (page 5.1-54).
```

## RCB_ALLOCATED_PROC

| | |
|---|---|
| FUNCTION: | Performs further processing of an ALLOCATE request. It is invoked when PS receives an RCB_ALLOCATED record from the resources manager. |
| | The RETURN_CODE parameter of the ALLOCATE verb is set based upon the return code field of the RCB_ALLOCATED record. If the return code is OK, PS 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 not OK, PS sets the RETURN_CODE parameter of the ALLOCATE appropriately. |
| INPUT: | RCB_ALLOCATED record and ALLOCATE verb parameters |
| OUTPUT: | PS creates an FMH-5 Attach header and stores it in the send buffer in the RCB. |
| NOTES: | 1. If RETURN_CONTROL = IMMEDIATE, RM has allocated both an RCB and a session as a result of receiving ALLOCATE_RCB from PS. If RETURN_CONTROL = DELAYED_ALLOCATION_PERMITTED, PS defers sending a session request to RM until it has accumulated enough data (via SEND_DATAs from the transaction program) to fill its send buffer. |
| | 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 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. |
| | 4. 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:

| | |
|---|---|
| OBTAIN_SESSION_PROC | page 5.1-35 |
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB_ALLOCATED | page A-32 |
| ATTACH_RECEIVED | page A-32 |
| RCB | page A-7 |

```
Select based on RCB_ALLOCATED.RETURN_CODE (see Notes 1, 2, and 3):
   When OK
      Set RETURN_CODE of ALLOCATE verb to OK.
      Call FSM_CONVERSATION(S, ALLOCATE, RCB) (page 5.1-59).
      Set RESOURCE parameter of ALLOCATE to RCB identifier.

      Initialize allocated RCB:  set RCB.PS_TO_HS_RECORD fields to
       ALLOCATE, FMH, NOT_END_OF_DATA, and null data.
       Set SEND_LL_REMAINDER to 0, RECEIVE_LL_REMAINDER to 0,
       MAX_BUFFER_LENGTH to maximum buffer length allowed
       (implementation dependent), RQ_TO_SEND_RCVD to NO,
       LOCKS to SHORT, POST_CONDITIONS.FILL to LL,
       POST_CONDITIONS.MAX_LENGTH to 0, SEND_LL_BYTE to
       NOT_PRESENT, CONVERSATION_TYPE to TYPE parameter value of ALLOCATE verb,
       and SYNC_LEVEL to ALLOCATE.SYNC_LEVEL.

      Build FMH-5 Attach header with data in ALLOCATE (see Appendix H)
       and place it in the RCB.PS_TO_HS_RECORD.DATA.

      If RETURN_CONTROL parameter is WHEN_SESSION_ALLOCATED
       (see Note 1 for the other cases) then
         Call OBTAIN_SESSION_PROC(RCB, NO_ATTACH) (page 5.1-35).

         If state of FSM_ERROR_OR_FAILURE (page 5.1-61) is
          (ALLOCATE_FAILURE_RETRY | SYNCLEVEL_NOT_SUPPORTED | ALLOCATE_FAILURE_NO_RETRY) then
            Set RETURN_CODE of ALLOCATE to ALLOCATION_ERROR concatenated with
             ALLOCATION_FAILURE_RETRY, ALLOCATION_FAILURE_NO_RETRY, or
             SYNC_LEVEL_NOT_SUPPORTED_BY_LU, as appropriate.
            Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-59).

         Else
            If the FMH-5 is to be flushed (as an implementation-dependent option) then
               Set the RCB.PS_TO_HS_RECORD.TYPE to FLUSH.
               Call SEND_DATA_TO_HS_PROC (RCB) (page 5.1-48).
   When UNSUCCESSFUL
      Set RETURN_CODE of ALLOCATE to UNSUCCESSFUL.
   When SYNC_LEVEL_NOT_SUPPORTED
      Call FSM_CONVERSATION(S, ALLOCATE, RCB) (page 5.1-59).
      Initialize allocated RCB (for details see above).
      Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-59).
      Set RETURN_CODE to ALLOCATION_ERROR_SYNC_LEVEL_NOT_SUPPORTED_BY_LU (see Note 4).
```

## RECEIVE_DATA_PROCESSING

| | |
|---|---|
| FUNCTION: | Moves data from the passed BUFFER_DATA parameter to the returned DATA parameter of the passed RECEIVE verb.<br><br>If the transaction program has specified that it is to receive data in terms of the logical record format of the data (i.e., RCB.POST_CONDITIONS.FILL = LL), data is moved from BUFFER_DATA to the DATA parameter of the passed RECEIVE verb.<br><br>If the transaction program has specified that it is to receive data independent of the logical record format of the data (i.e., RCB.POST_CONDITIONS.FILL = BUFFER), data is moved from BUFFER_DATA to the DATA parameter of the passed receive verb one or more times, depending upon the amount of data requested by the transaction program and the number of logical records in the buffer. For example, if the transaction program has requested 20 bytes of data and 25 bytes of data (comprising 3 logical records of lengths 7 bytes, 9 bytes, and 9 bytes, respectively) are in BUFFER_DATA, RECEIVE_DATA_BUFFER_MANAGEMENT is invoked once to receive the first logical record (yielding 7 bytes of data in the DATA field). It is invoked a second time to receive the second logical record (yielding 16 bytes of data in the DATA field). And finally it is invoked again to receive the first four bytes of the third logical record (yielding 20 bytes of data in the DATA field, which is the amount the transaction program requested). |
| INPUT: | The entry in the RCB_LIST corresponding to the resource specified in RECEIVE_VERB, a data buffer element (DATA_BUFFER) from the RCB.HS_TO_PS_BUFFER_LIST, and RECEIVE verb parameters |
| OUTPUT: | The DATA_BUFFER, and WHAT_RECEIVED and DATA parameters of the passed RECEIVE verb are updated. |
| NOTE: | When FILL = LL is specified, the WHAT_RECEIVED parameter of the RECEIVE verb is set to DATA_COMPLETE when a complete logical record or the last remaining portion of a logical record, is passed to the transaction program. Otherwise, the WHAT_RECEIVED is set to DATA_INCOMPLETE when FILL = LL is specified. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| RCB | page A-7 |

```
Select based on RCB.POST_CONDITIONS.FILL:
   When LL
      If already received a complete logical record
       (i.e., RCB.POST_CONDITIONS.MAX_LENGTH = 0) then
         Set WHAT_RECEIVED of receive verb to DATA_INCOMPLETE.
      Else
         Set DATA of receive verb to the first LEN bytes
          (LEN is the smaller of RECEIVE_LL_REMAINDER and
          RCB.POST_CONDITIONS.MAX_LENGTH) of the DATA_BUFFER and
          subtract LEN from RECEIVE_LL_REMAINDER and
          RCB.POST_CONDITIONS.MAX_LENGTH.  Remove the first LEN bytes
          from the DATA_BUFFER.
         If RECEIVE_LL_REMAINDER = 0 then
            Set WHAT_RECEIVED of RECEIVE verb to DATA_COMPLETE.
         Else
            Set WHAT_RECEIVED of RECEIVE verb to DATA_INCOMPLETE.
   When BUFFER
      Set WHAT_RECEIVED of RECEIVE verb to DATA.
      Do while RCB.POST_CONDITIONS.MAX_LENGTH > 0 and
       while the of length BUFFER_DATA > 0
         If RECEIVE_LL_REMAINDER = 0 and length of BUFFER_DATA = 1 then
            Set RCB.POST_CONDITIONS.MAX_LENGTH to 0.
         Else
            Set DATA of RECEIVE verb to DATA_BUFFER as described above.
Set RETURN_CODE of RECEIVE verb to OK.
Set LENGTH parameter of RECEIVE verb to length of DATA of RECEIVE verb.
```

## RECEIVE_RM_OR_HS_TO_PS_RECORD

| | |
|---|---|
| FUNCTION: | Returns a record sent by RM (or HS) to the conversation identified by RCB_ID. |
| INPUT: | RCB_ID (the ID of the conversation), and SUSPEND_FLAG |
| OUTPUT: | A record received from RM or HS. This record may be null if no record is available and SUSPEND_FLAG = NO_SUSPEND. |
| NOTE: | CONVERSATION_FAILURE is the only possible record that can arrive from RM. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| HS | page 6.0-3 |
| RM | page 3-17 |
| CONVERSATION_FAILURE | page A-32 |
| HS_TO_PS_RECORD | page A-12 |
| RCB | page A-7 |

```
If SUSPEND_FLAG = SUSPEND then
   Wait until a record has arrived from RM or HS for
    conversation RCB_ID.
Else (i.e., when SUSPEND_FLAG=NO_SUSPEND)
   Get the record arrived from RM or HS
    (Record may be null if no record has arrived yet.)
Return record.
```

## SEND_DATA_BUFFER_MANAGEMENT

| | |
|---|---|
| FUNCTION: | Determines if there is enough data to be sent to HS.<br><br>PS continues to send data to HS until the amount of data remaining to be sent is less than or equal to the maximum buffer size, in which case PS stores the data in the RCB until more data is issued by the transaction program or the buffer is flushed. If the data in the buffer is exactly equal to the maximum buffer 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 accumulated in the RCB buffer, one or more SEND_DATARECORDs are sent to HS. Otherwise, the data is stored in the RCB to be sent at a later time. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |

```
Set TEMP_BUFFER to RCB.PS_TO_HS_RECORD_DATA concatenated to DATA.
Set NO_ERROR_OR_FAILURE to true.
Do while length of TEMP_BUFFER > RCB.MAX_BUFFER_LENGTH
 and NO_ERROR_OR_FAILURE is true:
   Set RCB.PS_TO_HS_RECORD.DATA to first RCB.MAX_BUFFER_LENGTH bytes
    of the TEMP_BUFFER, remove these bytes from TEMP_BUFFER.
   Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).
   If state of FSM_ERROR_OR_FAILURE (page 5.1-61)
    is ALLOCATE_FAILURE_RETRY | ALLOCATE_FAILURE_NO_RETRY | SYNCLEVEL_NOT_SUPPORTED then
   Set NO_ERROR_OR_FAILURE to false.
Move TEMP_BUFFER into RCB.PS_TO_HS_RECORD.DATA.
```

## SEND_DATA_TO_HS_PROC

| | |
|---|---|
| FUNCTION: | Handles the sending of data to HS to be sent to the partner transaction program.<br><br>If no session has as yet been allocated to the conversation associated with the passed RCB, PS now requests a session from the resources manager. If the transaction program has stopped sending data and has issued DEALLOCATE, PREPARE_TO_RECEIVE, and/or CONFIRM, PS requests RM to send the data, which contains an Attach header, when RM allocates the session. Otherwise, PS sends the data itself when RM has allocated a session. |
| INPUT: | The RCB associated with the conversation |
| OUTPUT: | SEND_DATA_RECORD to HS |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| HS | page 6.0-3 |
| OBTAIN_SESSION_PROC | page 5.1-35 |
| RCB | page A-7 |
| SEND_DATA_RECORD | page A-24 |

```
If no session has been allocated to this conversation then
   If RCB.PS_TO_HS_RECORD.TYPE = FLUSH | NOT_END_OF_DATA then
      Call OBTAIN_SESSION_PROC(RCB, NO_ATTACH) (page 5.1-35).
      Create a SEND_DATA_RECORD, copy RCB.PS_TO_HS_RECORD into it, and
       send it to HS.
   Else
      Call OBTAIN_SESSION_PROC(RCB, ATTACH) (page 5.1-35).
Else (session previously allocated)
   Create a SEND_DATA_RECORD, copy RCB.PS_TO_HS_RECORD into it, and
    send it to HS.
Set RCB.PS_TO_HS_RECORD fields as follows:
 ALLOCATE to NO, FMH to NO, TYPE to NOT_END_OF_DATA, and DATA to null.
```

## SEND_ERROR_DONE_PROC

| | |
|---|---|
| FUNCTION: | This procedure performs further processing of the SEND_ERROR verb.<br><br>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 in 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.<br><br>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 "Appendix H. FM Header and LU Services Commands") 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:

| | |
|---|---|
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| SEND_DATA_BUFFER_MANAGEMENT | page 5.1-47 |
| FSM_CONVERSATION | page 5.1-59 |
| RCB | page A-7 |

```
Select based on the following conditions:
   When TYPE parameter of SEND_ERROR verb is PROG and state
    of FSM_CONVERSATION (page 5.1-59) is SEND
      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 5.1-59) is RCV,
    RCVD_CONFIRM, RCVD_CONFIRM_SEND, RCVD_CONFIRM_DEALL
      Set SENSE_DATA to X'08890000'.
   When  type of SEND_ERROR is SVC
    and state of FSM_CONVERSATION (page 5.1-59) is SEND
      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 5.1-59) is
    RCV | RCVD_CONFIRM | RCVD_CONFIRM_SEND | RCVD_CONFIRM_DEALL
      Set SENSE_DATA to X'08890100'.
If LOG_DATA parameter of SEND_ERROR is not null then
   Move SENSE_DATA into RCB.PS_TO_HS_RECORD.DATA as an FMH-7 record.
   Create Error log GDS variable with the LOG_DATA and concatenate
    it to RCB.PS_TO_HS_RECORD.DATA.
   Insert Error Log GDS variable into a system error log.
Else
   Move SENSE_DATA into RCB.PS_TO_HS_RECORD.DATA as an FMH-7 record.
If FLUSH verb is not implemented or the FMH-7 is
 to be flushed immediately then (as an implementation-dependent option)
   Set type of RCB.PS_TO_HS_RECORD to FLUSH.
   Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).
Else
   Call SEND_DATA_BUFFER_MANAGEMENT (null data, RCB) (page 5.1-47).
Set RETURN_CODE of SEND_ERROR to OK.
```

## SEND_ERROR_IN_RECEIVE_STATE

| | |
|---|---|
| FUNCTION: | 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 a RECEIVE_DATA record with 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 element in the RCB.HS_TO_PS_BUFFER_LIST is not a DEALLOCATE buffer element, 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:

| | |
|---|---|
| PS | page 5.0-5 |
| HS | page 6.0-3 |
| WAIT_FOR_SEND_ERROR_DONE_PROC | page 5.1-58 |
| FSM_CONVERSATION | page 5.1-59 |
| RCB | page A-7 |
| SEND_ERROR | page A-24 |

```
If first entry on RCB.HS_TO_PS_BUFFER_LIST is DEALLOCATE_FLUSH (see Note 1) then
   Set RETURN_CODE parameter of the SEND_ERROR verb to DEALLOCATE_NORMAL.
   Call FSM_CONVERSATION(R, DEALLOCATE_NORMAL_RC, RCB) (page 5.1-59).
Else (see Note 2)
   Send SEND_ERROR record to HS.
   Call WAIT_FOR_SEND_ERROR_DONE_PROC(SEND_ERROR verb parameters, RCB) (page 5.1-58).
```

## SEND_ERROR_IN_SEND_STATE

| | |
|---|---|
| FUNCTION: | Invoked when the transaction program issues a SEND_ERROR verb for a conversation that is in the SEND state.<br><br>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 the buffer.<br><br>If the state of FSM_ERROR_OR_FAILURE indicates that a RECEIVE_ERROR record has been received from HS, PS sends a SEND_DATA record with the 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. |
| NOTE: | If no session has been allocated to this conversation (i.e., the ALLOCATE verb issued to allocate the conversation specified RETURN_CONTROL = DELAYED_ALLOCATION_PERMITTED), PS now requests a session from the resources manager. If, while attempting to allocate a session, the local LU detects an error, PS sets the RETURN_CODE parameter in the SEND_ERROR to reflect the type of allocation error and returns control to the transaction program. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| SEND_ERROR_DONE_PROC | page 5.1-49 |
| POST_AND_WAIT_PROC | page 5.1-37 |
| DEQUEUE_FMH7_PROC | page 5.1-33 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |

```
If state of FSM_ERROR_OR_FAILURE = NO_RQS then
   Set CONTINUE to true.

   If RCB.PS_TO_HS_RECORD.TYPE is not null then
      Set RCB.PS_TO_HS_RECORD.TYPE to FLUSH.
      Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).

      If state of FSM_ERROR_OR_FAILURE (page 5.1-61) is
       ALLOCATE_FAILURE_RETRY | ALLOCATE_FAILURE_NO_RETRY
       | SYNCLEVEL_NOT_SUPPORTED then (see Note)
         Set RETURN_CODE parameter of SEND_ERROR verb to ALLOCATION_ERROR concatenated with
          ALLOCATION_FAILURE_RETRY, ALLOCATION_FAILURE_NO_RETRY, or
          SYNC_LEVEL_NOT_SUPPORTED_BY_LU, as appropriate.
         Call FSM_CONVERSATION(R, ALLOCATION_ERROR_RC, RCB) (page 5.1-59).
         Set CONTINUE to false.

      If CONTINUE then
         Call FSM_CONVERSATION(S, SEND_ERROR, RCB) (page 5.1-48).
         Call SEND_ERROR_DONE_PROC(SEND_ERROR verb parameters, RCB) (page 5.1-49).
         Set RCB.SEND_LL_REMAINDER to 0 and set RCB.SEND_LL_BYTE to NOT_PRESENT to
          indicate that the data sent by TP is at a logical boundary.

Else (i.e., RCVD_ERROR)
   Set RCB.PS_TO_HS_RECORD type to PREPARE_TO_RCV_FLUSH.
   Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).
   Call POST_AND_WAIT_PROC(RCB, LL, X'7FFF') (page 5.1-37) to post
    when the whole FMH7 is received.
   Call DEQUEUE_FMH7_PROC(SEND_ERROR verb parameters, RCB) (page 5.1-33).
```

## SEND_ERROR_TO_HS_PROC

| | |
|---|---|
| FUNCTION: | This procedure creates a SEND_ERROR and sends it to HS. |
| INPUT: | The RCB associated with the HS to which the SEND_ERROR is to be sent |
| OUTPUT: | SEND_ERROR (a variant of PS_TO_HS_RECORD) to PS |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS | page 5.0-5 |
| PS_PROCESS_DATA | page 5.0-20 |
| RCB | page A-7 |
| SEND_ERROR | page A-24 |

Create a SEND_ERROR record (page A-24) with RCB.RCB_ID.

Send this SEND_ERROR record to HS.

## SET_FMH7_RC

| | |
|---|---|
| FUNCTION: | Sets the RETURN_CODE parameter of the passed transaction program verb based upon the sense data carried in the passed FMH-7. |
| INPUT: | The RCB corresponding to the resource to which the FMH-7 applies, the received FMH-7, and the transaction program verb parameters currently being processed |
| OUTPUT: | The RETURN_CODE parameter of the verb is set, based upon the sense data carried in the FMH-7. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PS_PROTOCOL_ERROR | page 5.0-16 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |

```
Select based on the Sense Data in FMH-7:
   When ALLOCATION_ERROR code
      Call PROCESS_RM_OR_HS_TO_PS_RECORDS with RCB_ID and SUSPEND (page 5.1-43),
       and remove DEALLOCATE buffer from RCB.HS_TO_PS_BUFFER_LIST.
       If neither DEALLOCATE_FLUSH nor DEALLOCATE_CONFIRM are found then
          Call PS_PROTOCOL_ERROR (page 5.0-16) with X'1008201D'.
      Set RETURN_CODE parameter of the verb to the corresponding value (see Appendix H to
       find the value corresponding to a given Sense Data).
      Call FSM_CONVERSATION(R, ALLOCATION_ERROR, RCB) (page 5.1-59).
   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 5.1-59).
   When PROG_ERROR_NO_TRUNC or PROG_ERROR_PURGING
      If state of FSM_ERROR_OR_FAILURE (page 5.1-61) 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 5.1-59).
   When PROG_ERROR_TRUNC
      Set RETURN_CODE parameter of the verb to PROG_ERROR_TRUNC.
      Call FSM_CONVERSATION(R, PROGRAM_ERROR_RC, RCB) (page 5.1-59).
   When SVC_ERROR_NO_TRUNC or SVC_ERROR_PURGING
      If state of FSM_ERROR_OR_FAILURE (page 5.1-61) 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) (page 5.1-59).
   When SVC_ERROR_TRUNC
      Set RETURN_CODE parameter of the verb to SVC_ERROR_TRUNC.
      Call FSM_CONVERSATION(R, SERVICE_ERROR_RC, RCB) (page 5.1-59).
   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 Appendix G under X'0864' Sense Code.
      Call FSM_CONVERSATION(R, DEALLOCATE_ABEND_RC, RCB).
   Otherwise (as an implementation-dependent option):
      Call PS_PROTOCOL_ERROR (page 5.0-16) with FMH-7 Sense Data.
```

## TEST_FOR_POST_SATISFIED

FUNCTION: Tests whether the post conditions specified in the RCB have been satisfied.

INPUT: The entry in the RCB_LIST corresponding to the resource to be tested.

OUTPUT: The state of FSM_POST is set to POSTED if the post conditions are satisfied.

NOTES:

1. If there are no entries on the HS_TO_PS_BUFFER_LIST, the resource cannot be posted.
2. Receipt of a CONFIRM, PREPARE_TO_RCV, or DEALLOCATE indicator causes the conversation to be posted. A later (optional) receive check determines if the received indicator invalidly truncates a logical record and, if so, appropriate action is taken at that time. Similarly, receipt of a logical record with associated LL field containing an invalid value or indicating a PS header causes posting to occur. Processing of the invalid LL or PS header takes place at a later time.
3. The high-order bit of the LL field is a Continuation bit, which is on unless this logical record is the final one in the current GDS variable. (Continued GDS variables and the information in this bit are specific to "Chapter 5.2. Presentation Services--Mapped Conversation Verbs" in Chapter 5.2).
4. If more than one entry is on the RCB.HS_TO_PS_BUFFER_LIST, one of the entries must be a CONFIRM, PREPARE_TO_RCV, or DEALLOCATE indicator, which causes the conversation to be posted.

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_POST | page 5.1-62 |
| RCB | page A-7 |
| BUFFER_ELEMENT | page A-8 |

```
Select based on number of entries in the RCB.HS_TO_PS_BUFFER_LIST:
   When 0
      Do nothing (see Note 1).
   When 1
      Select based on the type of the first buffer in the list:
         When CONFIRM | PREPARE_TO_RCV_CONFIRM | PREPARE_TO_RCV_FLUSH |
          DEALLOCATE_CONFIRM | DEALLOCATE_FLUSH | FMH7
            Call FSM_POST (page 5.1-62) and pass it a POST signal (see Note 2).
         When DATA
            Select based on RCB.POST_CONDITIONS.FILL:
               When BUFFER
                  If length of buffer data ≥ maximum length in POST_CONDITIONS,
                   or if PS header or invalid length present (see Appendix H) then
                     Call FSM_POST (page 5.1-62) and pass it a POST signal.
               When LL
                  If RCB.RECEIVE_LL_REMAINDER = 0 and length of buffer data  ≥ 2 then
                     If PS header or invalid LL (see Appendix H and Note 3) then
                        Call FSM_POST (page 5.1-62) and pass it a POST signal.

                     Else
                        Calculate RCB.RECEIVE_LL_REMAINDER from LL in the buffer
                         and high order bit forced to 0.
                        If length of buffer ≥ RCB.RECEIVE_LL_REMAINDER
                         or ≥ RCB.POST_CONDITIONS.MAX_LENGTH then
                           Call FSM_POST (page 5.1-62) and pass it a POST signal.
                        Else
                           Do nothing.
                  Else
                     If length of buffer data ≥ RCB_RECEIVE_LL_REMAINDER
                      or ≥ RCB.POST_CONDITIONS.MAX_LENGTH then
                        Call FSM_POST (page 5.1-62) and pass it a POST signal.
   Otherwise (number of entries is ≥ 2, see Note 4)
    Call FSM_POST (page 5.1-62) and pass it a POST signal.
```

## WAIT_FOR_CONFIRMED_PROC

| | | |
|---|---|---|
| FUNCTION: | | Invoked after a SEND_DATA record indicating CONFIRM has been sent to HS and a CONFIRMED record is expected in reply.<br><br>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 record 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:

| | |
|---|---|
| PS | page 5.0-5 |
| HS | page 6.0-3 |
| RM | page 3-17 |
| RECEIVE_RM_OR_HS_TO_PS_RECORD | page 5.1-47 |
| CONVERSATION_FAILURE_PROC | page 5.1-29 |
| POST_AND_WAIT_PROC | page 5.1-37 |
| DEQUEUE_FMH7_PROC | page 5.1-33 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |
| HS_TO_PS_RECORD | page A-12 |
| RM_TO_PS_RECORD | page A-31 |

```
Set CONTINUE to true.
Do while CONTINUE is true:
   Wait for record to arrive.
   If record arrived from RM then
      Call CONVERSATION_FAILURE_PROC with record (see page 5.1-29).
      If state of FSM_ERROR_OR_FAILURE (page 5.1-61) 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 5.1-59).
      Set CONTINUE to false.
```

```
     Else (i.e., record arrived from HS)
        Select based on record type:
           When REQUEST_TO_SEND
              Record in the RCB that request to send was received on the conversation.
           When RECEIVE_ERROR
              Call FSM_ERROR_OR_FAILURE(RECEIVE_ERROR, RCB) (page 5.1-61).
              Call POST_AND_WAIT_PROC(RCB, LL, X'7FFF') (page 5.1-37).
              Call DEQUEUE_FMH7_PROC(CONFIRM verb parameters, RCB) (page 5.1-33).
              Set CONTINUE to false.
           When CONFIRMED
              Set RETURN_CODE parameter of the verb to OK.
              If state of FSM_CONVERSATION is PEND_DEALL then
                 Call FSM_CONVERSATION(R, DEALLOCATION_INDICATOR, RCB) (page 5.1-59).
                 Purge all records from HS to PS process.
                 Create DEALLOCATE_RCB, initialize it, and send it to RM.
              Set CONTINUE to FALSE.
```

## WAIT_FOR_RM_REPLY

| | | |
|---|---|---|
| FUNCTION: | | Waits for an expected reply from the LU resources manager. |
| INPUT: | | None |
| 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:

| | |
|---|---|
| PS | page 5.0-5 |
| RM | page 3-17 |
| CONVERSATION_FAILURE_PROC | page 5.1-29 |
| RM_TO_PS_RECORD | page A-31 |

```
Set CONTINUE to true.
Do while CONTINUE is true:
   Wait until RM_TO_PS_RECORD has arrived from RM.
   If RM_TO_PS_RECORD type is CONVERSATION_FAILURE then
      Call CONVERSATION_FAILURE_PROC(RM_TO_PS_RECORD) (page 5.1-29).

   Else
      Set CONTINUE to false.
      Return with RM_TO_PS_RECORD.
```

## WAIT_FOR_RSP_TO_RQ_TO_SEND_PROC

| | | |
|---|---|---|
| FUNCTION: | | 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.<br><br>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 has no RETURN CODE parameter, 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 record that does have a RETURN_CODE field. |
| | 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 a RECEIVE_DATA record 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:

| | |
|---|---|
| RECEIVE_RM_OR_HS_TO_PS_RECORD | page 5.1-47 |
| CONVERSATION_FAILURE_PROC | page 5.1-29 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |
| BUFFER_ELEMENT | page A-8 |

```
Set CONTINUE to true.
Do while CONTINUE is true:
  Call RECEIVE_RM_OR_HS_TO_PS_RECORD(RCB.RCB_ID, SUSPEND)
   and receive record (page 5.1-47).
  If record arrived from RM then
     Call CONVERSATION_FAILURE_PROC with record (page 5.1-29).
     Set CONTINUE to false.
  If record arrived from HS then

     Select based on the type of the record:
        When REQUEST_TO_SEND (see Note 1)
           Set RCB.RQ_TO_SEND_RCVD to YES.
        When RECEIVE_ERROR (see Note 2)
           Call FSM_ERROR_OR_FAILURE(RECEIVE_ERROR, RCB) (see page 5.1-61)
        When RSP_TO_REQUEST_TO_SEND (see Note 3)
           Set CONTINUE to false.
        When RECEIVE_DATA (see Note 4)
           Enqueue record to RCB.HS_TO_PS_BUFFER_LIST.
           Set BUFFER_ELEMENT to the last entry of HS_TO_PS_RECORD_LIST.
           If BUFFER_ELEMENT type = DEALLOCATE_FLUSH then
              Set CONTINUE to false.
```

## WAIT_FOR_SEND_ERROR_DONE_PROC

FUNCTION: Invoked after a SEND_ERROR record 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 5.1-34) to await the arrival from HS of a record indicating EC. 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 a RECEIVE_DATA with 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 a RECEIVE_DATA with 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_RQS 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_RQS by SET_FMH7_RC (page 5.1-53) when the error is reported to the TP.

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| GET_END_CHAIN_FROM_HS | page 5.1-34 |
| SEND_ERROR_DONE_PROC | page 5.1-49 |
| COMPLETE_DEALLOCATE_ABEND_PROC | page 5.1-28 |
| FSM_CONVERSATION | page 5.1-59 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |
| BUFFER_ELEMENT | page A-8 |

```
Call GET_END_CHAIN_FROM_HS(RCB) (page 5.1-34).

Select based on the state of FSM_ERROR_OR_FAILURE (page 5.1-61):
   When CONV_FAILURE_SON
      Set RETURN_CODE of the verb to RESOURCE_FAILURE_RETRY.
      Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
   When CONV_FAILURE_PROTOCOL_ERROR
      Set RETURN_CODE of the verb to RESOURCE_FAILURE_NO_RETRY.
      Call FSM_CONVERSATION(R, RESOURCE_FAILURE_RC, RCB) (page 5.1-59).
   Otherwise
      Get BUFFER_ELEMENT from HS_TO_PS_BUFFER_LIST.

      Select based on the following conditions:
         When BUFFER_ELEMENT type is DEALLOCATE_FLUSH (see Note 1) and the verb is SEND_ERROR
            Set RETURN_CODE of verb to DEALLOCATE_NORMAL.
            Call FSM_CONVERSATION(R, DEALLOCATE_NORMAL_RC, RCB) (page 5.1-59).
         When BUFFER_ELEMENT type is DEALLOCATE_FLUSH and the verb is DEALLOCATE
            Set RETURN_CODE of the verb to OK.
         When BUFFER_ELEMENT type is DEALLOCATE_CONFIRM, CONFIRM, PREPARE_TO_RCV_CONFIRM,
          or PREPARE_TO_RCV_FLUSH (see Note 2) and the verb is SEND_ERROR
            Call SEND_ERROR_DONE_PROC(transaction program verb parameters, RCB)
             (page 5.1-49).
            Call FSM_CONVERSATION(S, SEND_ERROR, RCB) (page 5.1-59).
         When BUFFER_ELEMENT type is DEALLOCATE_CONFIRM, CONFIRM, PREPARE_TO_RCV_CONFIRM,
          or PREPARE_TO_RCV_FLUSH, and the verb is DEALLOCATE
            Call COMPLETE_DEALLOCATE_ABEND_PROC(transaction program verb parameters, RCB)
             (page 5.1-28).
Call FSM_ERROR_OR_FAILURE (page 5.1-61) and pass it a RESET signal (see Note3).
```

## FSM_CONVERSATION

| | |
|---|---|
| FUNCTION: | This finite-state machine maintains the status of a conversation resource. The states have the following meanings: <br>• RESET = conversation initial state, the program can allocate it <br>• SEND = the program can send data, request confirmation, or request sync point <br>• RCV = receive, the program can receive information from the remote program <br>• RCVD_CONFIRM = received confirm, PS received the confirm indicator from the HS <br>• RCVD_CONFIRM_SEND = received confirm send, PS received the confirm send indicator from HS <br>• RCVD_CONFIRM_DEALL = received confirm deallocate, PS received the confirm deallocate from HS <br>• PREP_TO_RCV_DEFER = prepare to receive defer, the program issued a PREPARE_TO_RECEIVE verb with SYNCPT <br>• DEALL_DEFER = deallocate defer, the program issued DEALLOCATE verb with SYNCPT <br>• 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:

| | |
|---|---|
| PS | page 5.0-5 |
| HS | page 6.0-3 |
| SEND_DATA_TO_HS_PROC | page 5.1-48 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| FSM_ERROR_OR_FAILURE | page 5.1-61 |
| RCB | page A-7 |

| STATE NAMES----> | RESET | SEND | RCV | RCVD CONFIRM | RCVD CONFIRM SEND | RCVD CONFIRM DEALL | PREP TO RCV DEFER | DEALL DEFER | PEND DEALL | END CONV |
|---|---|---|---|---|---|---|---|---|---|---|
| INPUTS STATE NUMBERS--> | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 010 |
| S, ALLOCATE | 2 | / | / | / | / | / | / | / | / | / |
| R, ATTACH | 3 | / | / | / | / | / | / | / | / | / |
| S, SEND_DATA | / | - | >(A) | >(A) | >(A) | >(A) | >(A) | >(A) | / | >(A) |
| S, PREP_TO_RCV_FLUSH | / | 3 | >(A) | >(A) | >(A) | >(A) | >(A) | >(A) | / | >(A) |
| S, PREP_TO_RCV_CONFIRM | / | 3 | >(A) | >(A) | >(A) | >(A) | >(A) | >(A) | / | >(A) |
| S, PREP_TO_RCV_DEFER | / | 7 | >(A) | >(A) | >(A) | >(A) | >(A) | >(A) | / | >(A) |
| S, FLUSH | / | - | >(A) | >(A) | >(A) | >(A) | 3 | 1 | / | >(A) |
| S, CONFIRM | / | - | >(A) | >(A) | >(A) | >(A) | 3 | 9 | / | >(A) |
| S, SEND_ERROR | / | - | 2 | 2 | 2 | 2 | >(A) | >(A) | / | >(A) |
| S, RECEIVE_AND_WAIT | / | 3(B) | - | >(A) | >(A) | >(A) | >(A) | >(A) | / | >(A) |
| S, POST_ON_RECEIPT | / | >(A) | - | >(A) | >(A) | >(A) | >(A) | >(A) | / | >(A) |
| S, WAIT | / | >(A) | - | >(A) | >(A) | >(A) | >(A) | >(A) | / | >(A) |
| S, TEST | / | >(A) | - | >(A) | >(A) | >(A) | >(A) | >(A) | / | >(A) |
| S, REQUEST_TO_SEND | / | >(A) | - | - | >(A) | >(A) | >(A) | >(A) | / | >(A) |
| R, SEND_INDICATOR | / | / | 2 | / | / | / | / | / | / | / |
| R, CONFIRM_INDICATOR | / | / | 4 | / | / | / | / | / | / | / |
| R, CONFIRM_SEND_IND | / | / | 5 | / | / | / | / | / | / | / |
| R, CONFIRM_DEALLOC_IND | / | / | 6 | / | / | / | / | / | / | / |
| S, CONFIRMED | / | >(A) | >(A) | 3 | 2 | 10 | >(A) | >(A) | / | >(A) |
| R, PROGRAM_ERROR_RC | / | 3(C) | -(C) | / | / | / | 3(C) | 3(C) | 3(C) | / |
| R, SERVICE_ERROR_RC | / | 3(C) | -(C) | / | / | / | 3(C) | 3(C) | 3(C) | / |
| R, DEALLOC_NORMAL_RC | / | 10 | 10 | / | / | / | / | / | / | / |
| R, DEALLOC_ABEND_RC | / | 10(C) | 10(C) | / | / | / | 10(C) | 10(C) | 10(C) | / |
| R, RESOURCE_FAILURE_RC | / | 10(C) | 10(C) | / | / | / | 10(C) | 10(C) | 10(C) | / |
| R, ALLOCATION_ERROR_RC | / | 10(C) | 10(C) | / | / | / | 10(C) | 10(C) | 10(C) | / |
| S, DEALLOCATE_FLUSH | / | 1 | >(A) | >(A) | >(A) | >(A) | >(A) | >(A) | / | >(A) |
| S, DEALLOCATE_CONFIRM | / | 9 | >(A) | >(A) | >(A) | >(A) | >(A) | >(A) | / | >(A) |
| S, DEALLOCATE_DEFER | / | 8 | >(A) | >(A) | >(A) | >(A) | >(A) | >(A) | / | >(A) |
| S, DEALLOCATE_ABEND | / | 1 | 1 | 1 | 1 | 1 | 1 | 1 | / | >(A) |
| S, DEALLOCATE_LOCAL | / | >(A) | >(A) | >(A) | >(A) | >(A) | >(A) | >(A) | / | 1 |
| R, DEALLOCATED_IND | / | / | / | / | / | / | / | / | 1 | / |
| S, GET_ATTRIBUTES | / | - | - | - | - | - | - | - | / | - |

| OUTPUT CODE | FUNCTION |
|---|---|
| A | Call DEALLOCATION_CLEANUP_PROC (page 5.0-14). |
| B | If data sent by TP is on a logical record boundary then<br>   Set RCB.PS_TO_HS_RECORD.TYPE to PREPARE_TO_RCV_FLUSH.<br>   Call SEND_DATA_TO_HS_PROC(RCB) (page 5.1-48).<br>Else<br>   Call DEALLOCATION_CLEANUP_PROC (page 5.0-14). |
| C | Call FSM_ERROR_OR_FAILURE with RESET (page 5.1-61). |

## FSM_ERROR_OR_FAILURE

| | |
|---|---|
| FUNCTION: | This finite-state machine remembers if any error or failure records (either HS_TO_PS_RECORDs or RM_TO_PS_RECORDs) 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:<br>• NO_RQS = the initial state of the FSM<br>• RCVD_ERROR = a RECEIVE_ERROR was received<br>• CONV_FAILURE_PROTOCOL_ERROR = a conversation protocol error record was received<br>• CONV_FAILURE_SON = a session outage notification for the conversation was received<br>• ALLOCATE_FAILURE_RETRY = an allocation failure with retry was received<br>• ALLOCATE_FAILURE_NO_RETRY = an allocation failure with no retry allowed was received<br>• SYNCLEVEL_NOT_SUPPORTED = sync level not supported for the conversation is signaled |
| NOTE: | The inputs are the error and failure records from the HS and RM. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| HS | page 6.0-3 |
| RM | page 3-17 |
| RCB | page A-7 |

| STATE NAMES----> | NO RQS | RCVD ERROR | CONV FAILURE PROTOCOL ERROR | CONV FAILURE SON | ALLOCATE FAILURE RETRY | ALLOCATE FAILURE NO_RETRY | SYNCLEVEL NOT SUPPORTED |
|---|---|---|---|---|---|---|---|
| INPUTS STATE NUMBERS--> | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
| SIGNAL(CONV_FAIL_PROTOCOL) | 3 | 3 | / | / | / | / | / |
| SIGNAL(CONV_FAIL_SON) | 4 | 4 | / | / | / | / | / |
| RECEIVE_ERROR | 2(A) | / | - | - | / | / | / |
| SIGNAL(ALLOC_FAIL_RETRY) | 5 | / | / | / | / | / | / |
| SIGNAL(ALLOC_FAIL_NO_RETRY) | 6 | / | / | / | / | / | / |
| SIGNAL(SYNCLEVEL_NOT_SUPPTD) | 7 | / | / | / | / | / | / |
| SIGNAL(RESET) | - | 1 | 1 | 1 | 1 | 1 | 1 |

| OUTPUT CODE | FUNCTION |
|---|---|
| A | Set RCB.PS_TO_HS_RECORD.DATA to null (purge send buffer). |

## 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----> | RESET | PEND POSTED | POSTED |
|---|---|---|---|
| INPUTS STATE NUMBERS--> | 01 | 02 | 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 | - |

## LOCAL DATA STRUCTURES

TEST

TEST contains information that describes the test to be performed on the conversation and the result of the test.

```
TEST
   RESOURCE:  resource identifier
   TEST:  possible values:  POSTED, REQUEST_TO_SEND_RECEIVED
   RETURN_CODE:  possible values:  OK, UNSUCESSFUL, POSTING_NOT_ACTIVE
    RESOURCE_FAILURE_RETRY, RESOURCE_FAILURE_NO_RETRY, ALLOCATION_ERROR
```

# 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 of 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.

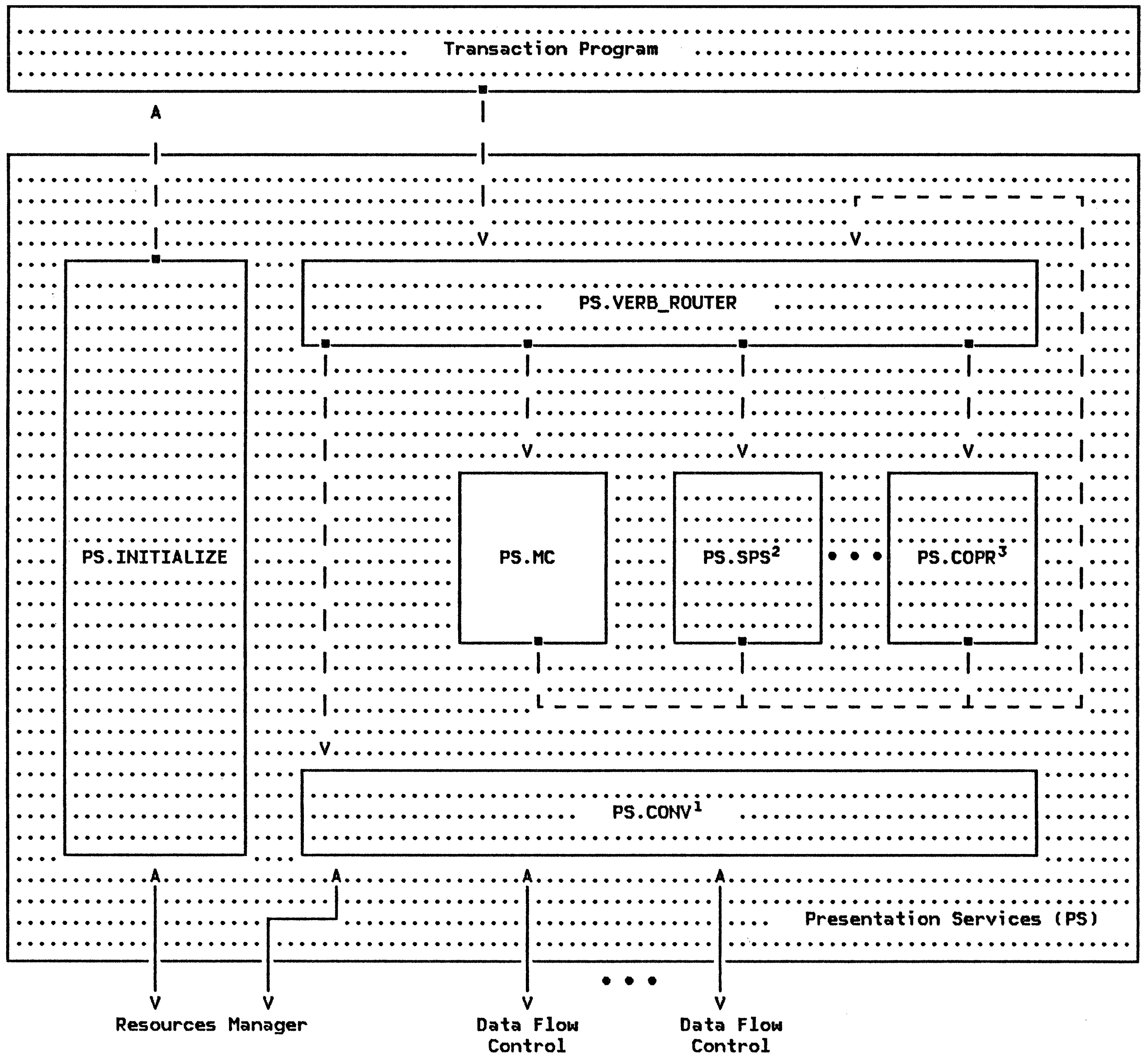
In general, the TP is faced with two formatting problems. The data format that it prefers for computational processing differs from the formats in which data is presented to (or by):

- 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).



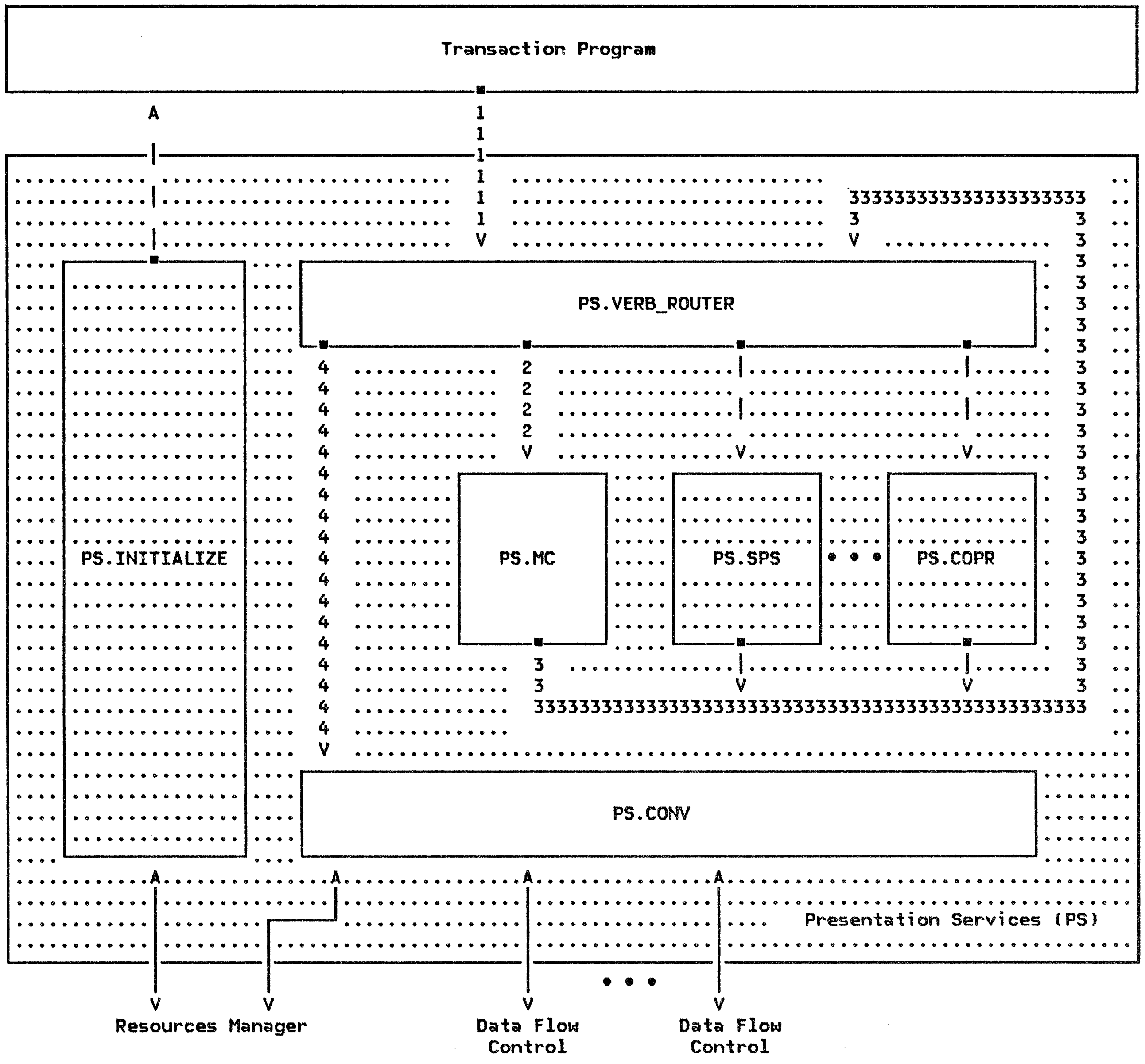
[1] See "Chapter 5.1. Presentation Services--Conversation Verbs"
[2] See "Chapter 5.3. Presentation Services--Sync Point Services Verbs"
[3] See "Chapter 5.4. Presentation Services--Control-Operator Verbs"

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. In both cases, the interaction actually occurs indirectly through PS.VERB_ROUTER (Chapter 5.0). Whenever a verb is issued by any component (for example, the TP), PS.VERB_ROUTER 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 PS.VERB_ROUTER once again gains control, receiving the verb issued by PS.MC (flow 3). This time, PS.VERB_ROUTER discovers that the verb is a basic conversation verb, so it calls PS.CONV and passes the verb to it (flow 4). 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. |

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.

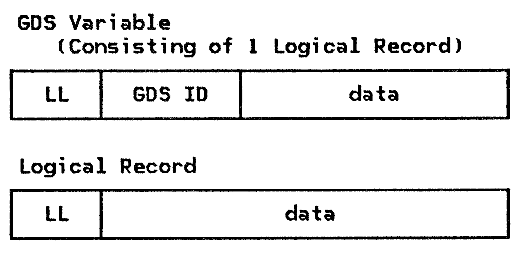


Figure 5.2-3. GDS Variables and Logical Records

The following types of GDS variables flow on mapped conversations:

- Map Name
- Application Data
- User Control Data
- Error Data
- Error Log
- Null Structured Data

(See Appendix H for descriptions of all the valid types of GDS variables and their GDS ID values.)

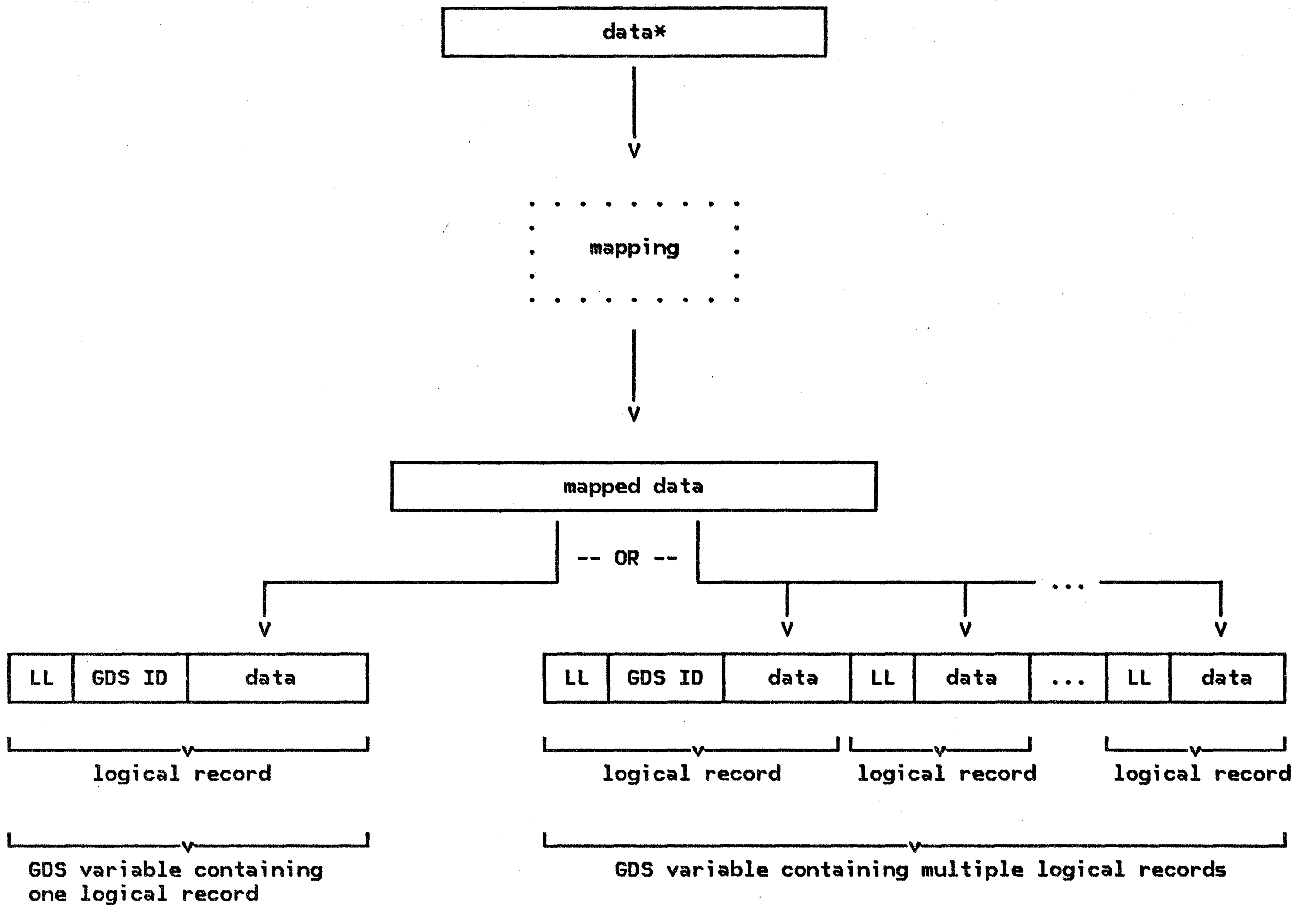
Map Name GDS variables are generated from the MAP_NAME parameter of MC_SEND_DATA (see "Map Names" on page 5.2-8 for details). Application Data and User Control Data GDS variables (collectively called data GDS variables) are generated from data supplied via the DATA parameter of MC_SEND_DATA. If this 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 Structured Data GDS variables are 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. However, the RH that conveys this state-change notification (CD for MC_PREPARE_TO_RECEIVE, or RQD2 for MC_CONFIRM) can flow to the partner only as a prefix to some GDS variable created by PS.MC. PS.MC cannot create a data GDS variable with a null data field 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. To solve this problem, PS.MC sends a Null Structured Data GDS variable.

### 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 SNA Transaction Programmer's Reference Manual for LU Type 6.2) 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.

## EXAMPLES OF MAPPED CONVERSATION VERB PROCESSING

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 attached (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 SNA Transaction Programmer's Reference Manual for LU Type 6.2), 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.

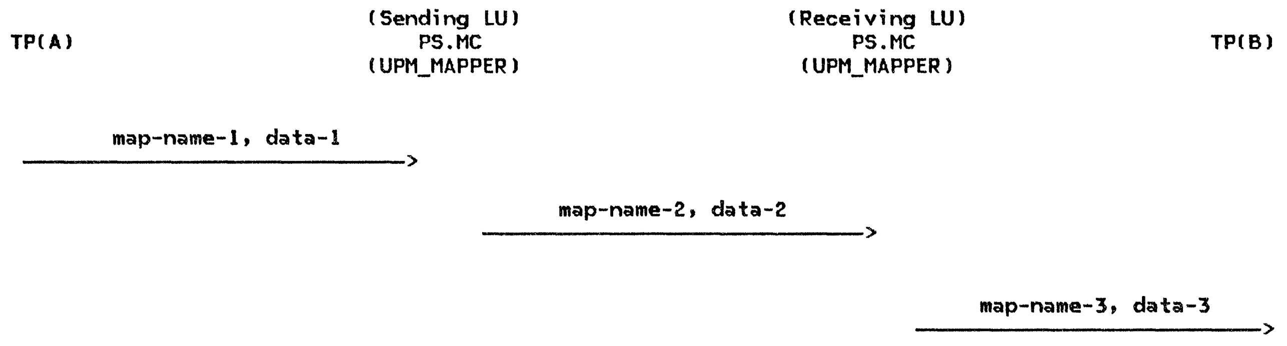
The output from the mapper is map-name-2 and data-2. Data-2 may be a different size from data-1 and may be in an entirely different format. After reformatting data-2 into a GDS variable (by breaking it into logical records according to MC_MAX_SEND_SIZE, and prefixing a GDS ID), PS.MC sends map-name-2 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-



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 ouput 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 _SNA Transaction Programmer's Reference Manual for LU Type 6.2_) 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 _off_, 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.

## _MC TEST PROC_

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.

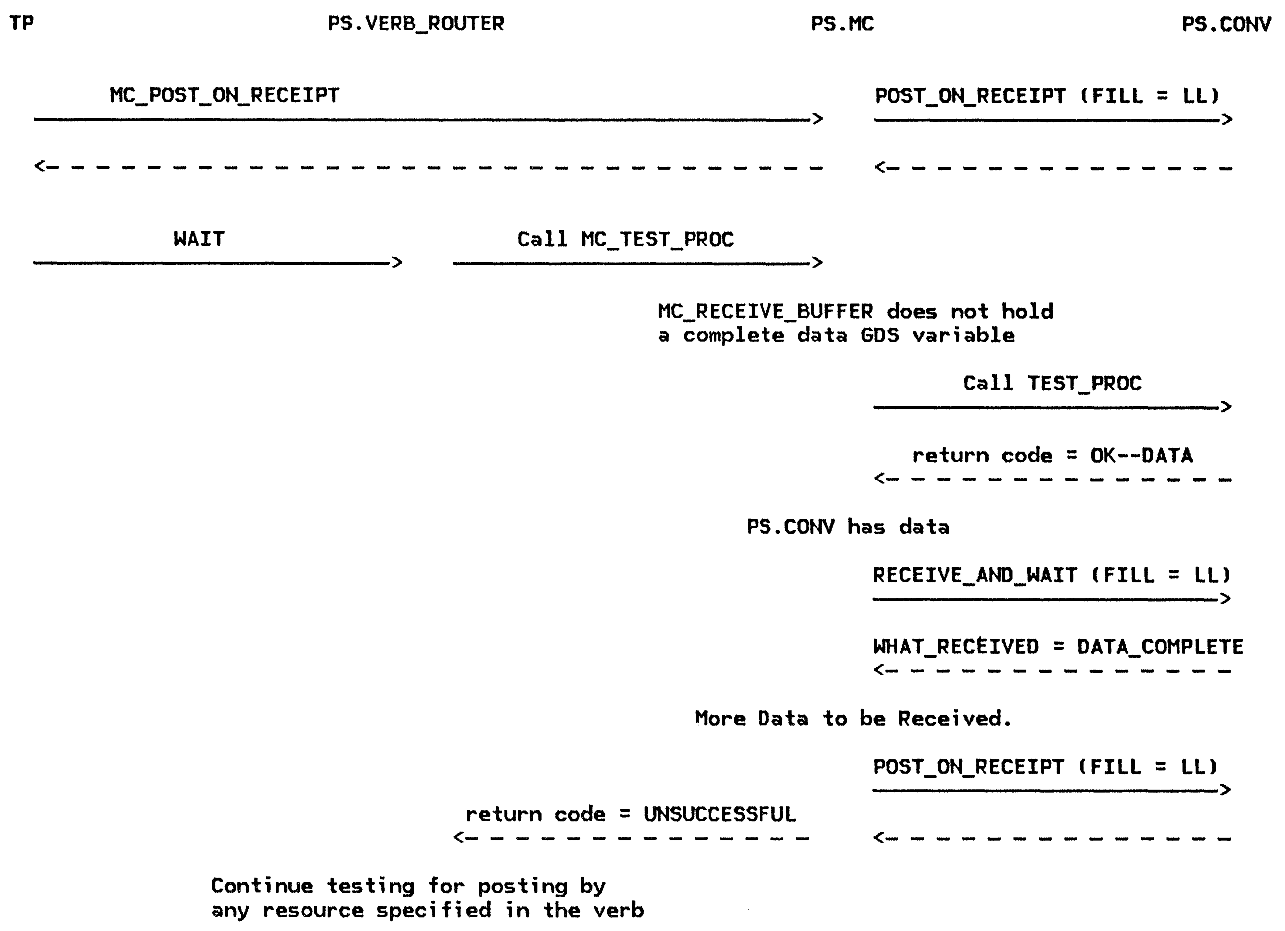
Once a complete data GDS variable has been received, PS.MC requests no more information from PS.CONV until it passes to the transaction program the data already in MC_RECEIVE_BUFFER.

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 <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>. 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


TP
PS.VERB_ROUTER
PS.MC
PS.CONV
MC_POST_ON_RECEIPT
POST_ON_RECEIPT (FILL = LL)
WAIT
Call MC_TEST_PROC
MC_RECEIVE_BUFFER does not hold a complete data GDS variable
Call TEST_PROC
return code = OK--DATA
PS.CONV has data
RECEIVE_AND_WAIT (FILL = LL)
WHAT_RECEIVED = DATA_COMPLETE
More Data to be Received.
POST_ON_RECEIPT (FILL = LL)
return code = UNSUCCESSFUL
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

Appendix H 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 SYNPCT. 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.

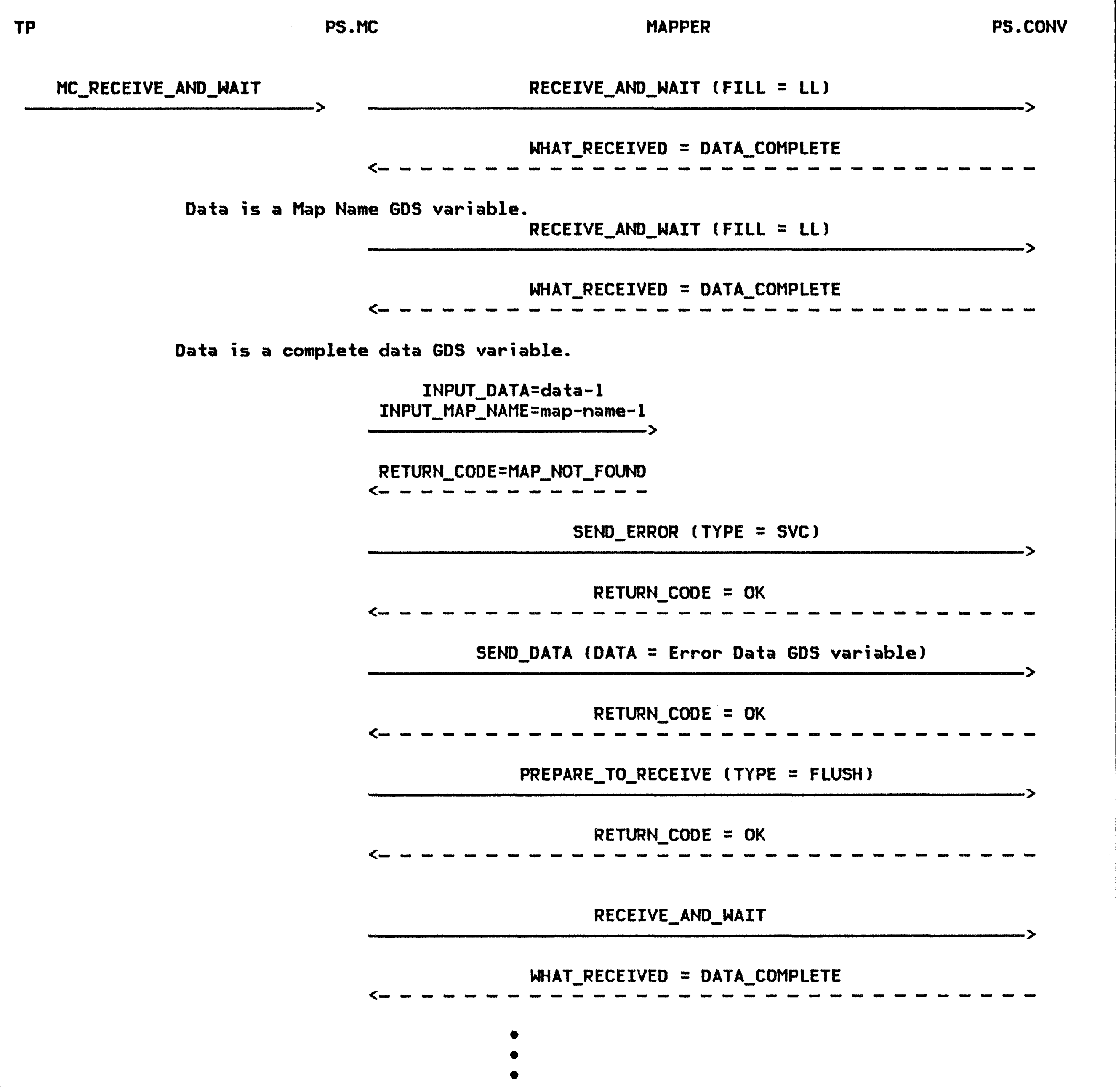
PS.MC also handles receipt of a service error notification from a partner LU when it is the partner that discovered the service error.

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 PS.MC a return code indicating that the map name received is invalid. The mapper has


TP
PS.MC
MAPPER
PS.CONV
MC_RECEIVE_AND_WAIT
RECEIVE_AND_WAIT (FILL = LL)
WHAT_RECEIVED = DATA_COMPLETE
Data is a Map Name GDS variable.
RECEIVE_AND_WAIT (FILL = LL)
WHAT_RECEIVED = DATA_COMPLETE
Data is a complete data GDS variable.
INPUT_DATA=data-1
INPUT_MAP_NAME=map-name-1
RETURN_CODE=MAP_NOT_FOUND
SEND_ERROR (TYPE = SVC)
RETURN_CODE = OK
SEND_DATA (DATA = Error Data GDS variable)
RETURN_CODE = OK
PREPARE_TO_RECEIVE (TYPE = FLUSH)
RETURN_CODE = OK
RECEIVE_AND_WAIT
WHAT_RECEIVED = DATA_COMPLETE

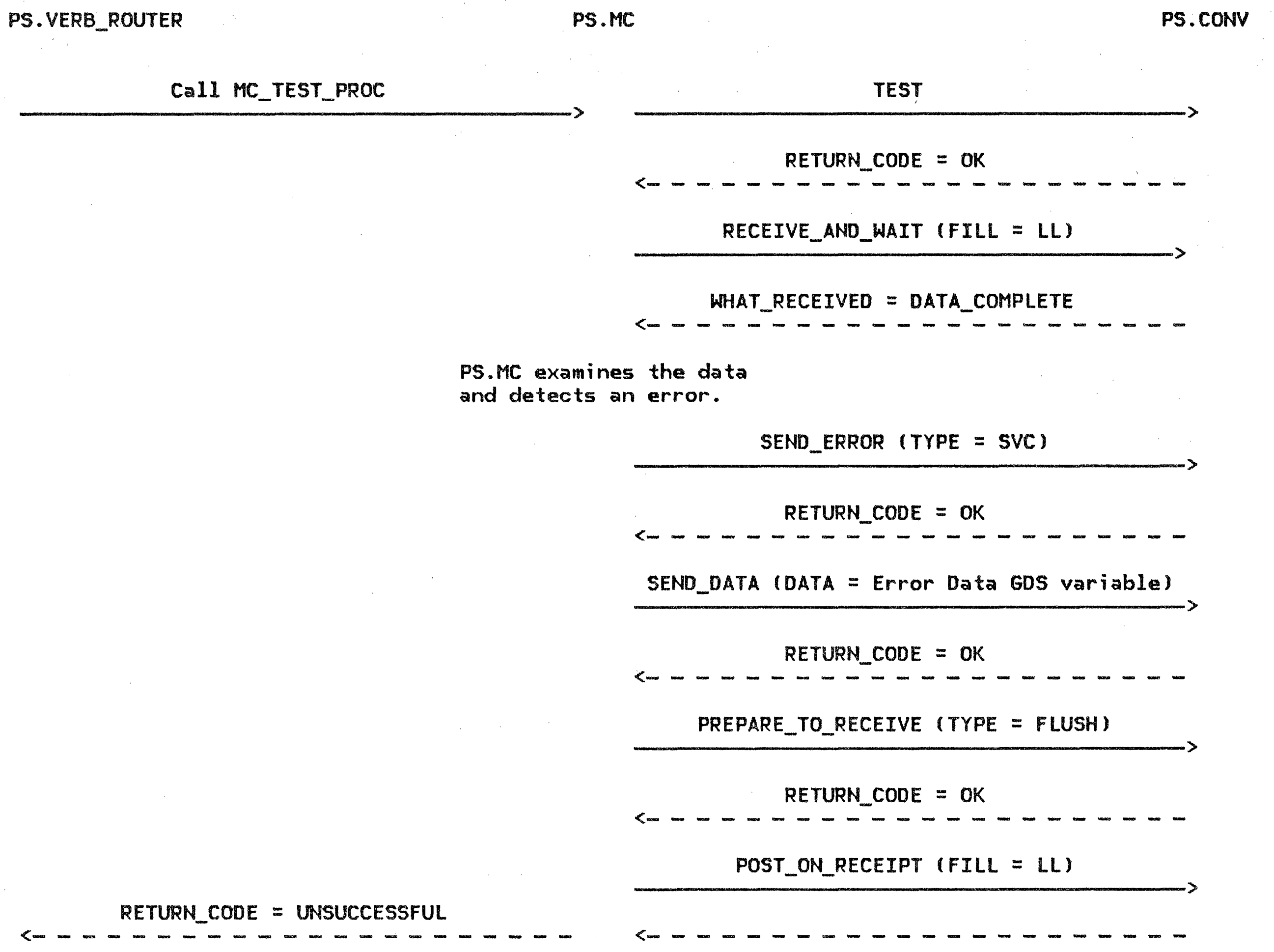
See "Service Errors Detected in Received Data" for an explanation of the flows shown in this figure.

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

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 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


PS.VERB_ROUTER
PS.MC
PS.CONV
Call MC_TEST_PROC
TEST
RETURN_CODE = OK
RECEIVE_AND_WAIT (FILL = LL)
WHAT_RECEIVED = DATA_COMPLETE
PS.MC examines the data and detects an error.
SEND_ERROR (TYPE = SVC)
RETURN_CODE = OK
SEND_DATA (DATA = Error Data GDS variable)
RETURN_CODE = OK
PREPARE_TO_RECEIVE (TYPE = FLUSH)
RETURN_CODE = OK
POST_ON_RECEIPT (FILL = LL)
RETURN_CODE = UNSUCCESSFUL

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

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

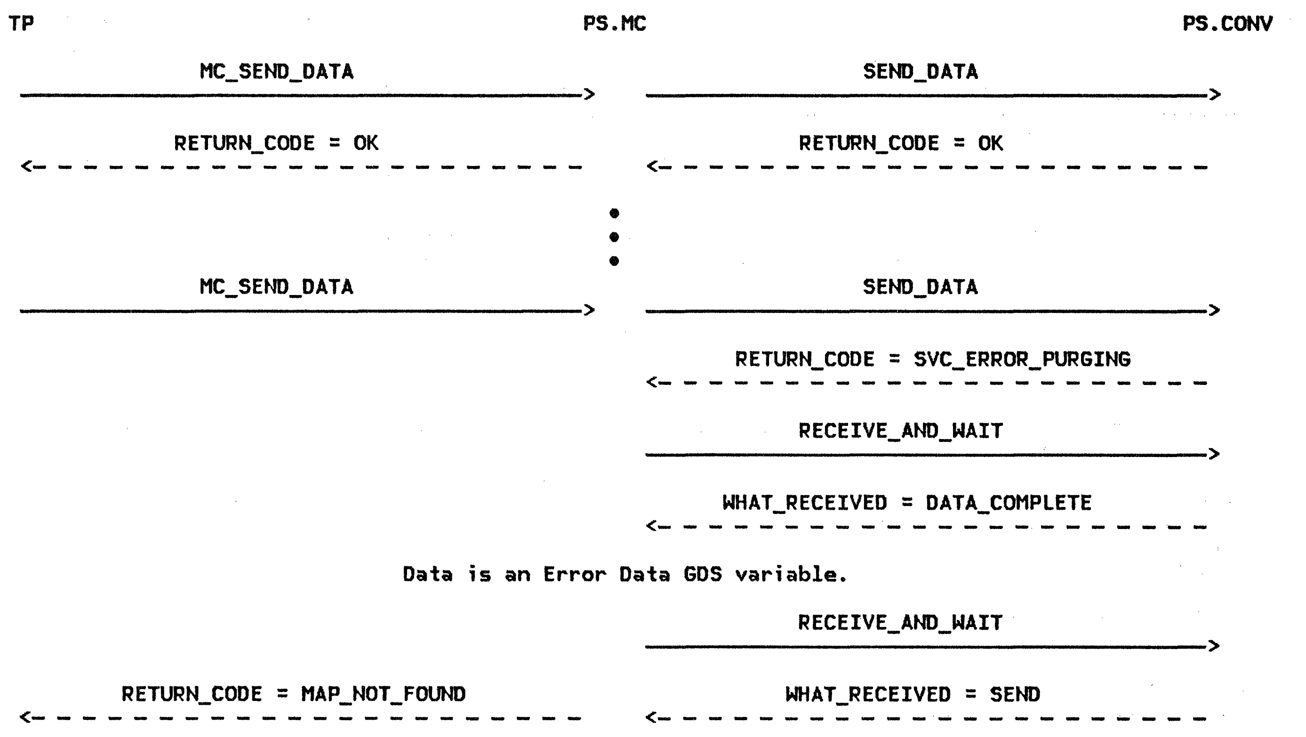
PS.MC also handles service errors that are detected by the partner LU. The error could have been detected in data sent to the partner LU by the local TP. Alternatively, the partner LU may have detected an error while sending data to PS.MC. Figure 5.2-9 on page 5.2-18 and Figure 5.2-10 on page 5.2-19 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.

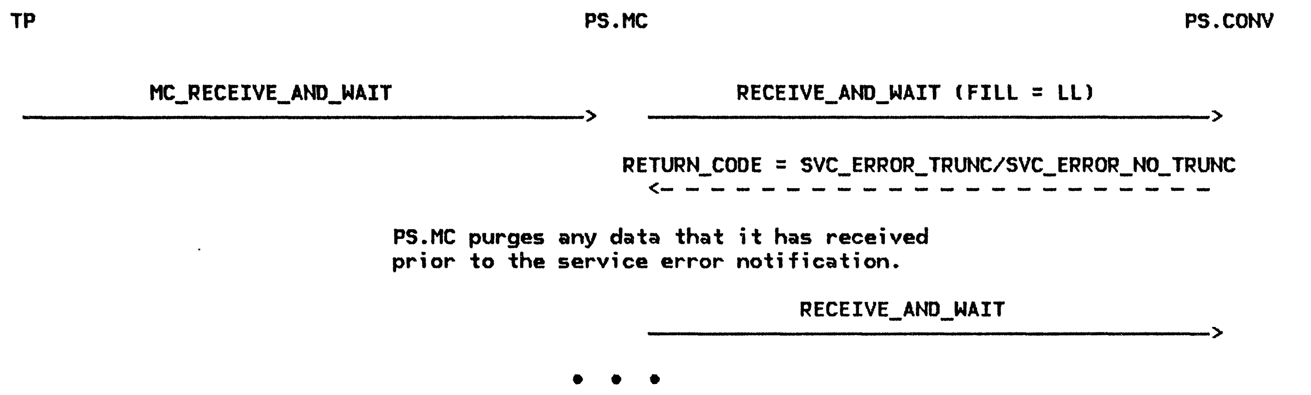


See "Processing of a Service Error Detected by Partner LU" for an explanation of the flows that are shown in this figure.

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)
RETURN_CODE = SVC_ERROR_TRUNC/SVC_ERROR_NO_TRUNC
PS.MC purges any data that it has received prior to the service error notification.
RECEIVE_AND_WAIT
• • •

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 or SVC_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.<br><br>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 5.2-22 |
| MC_CONFIRMED_PROC | page 5.2-23 |
| MC_DEALLOCATE_PROC | page 5.2-23 |
| MC_FLUSH_PROC | page 5.2-24 |
| 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 |
| | |
| PS_PROCESS_DATA | page 5.0-20 |

```
Select based on the mapped conversation verb (issued by the TP):
  When ALLOCATE
    Call MC_ALLOCATE_PROC (page 5.2-21).
  When  CONFIRM
    Call MC_CONFIRM_PROC (page 5.2-22).
  When  CONFIRMED
    Call MC_CONFIRMED_PROC (page 5.2-23).
  When  DEALLOCATE
    Call MC_DEALLOCATE_PROC (page 5.2-23).
  When  FLUSH
    CALL MC_FLUSH_PROC (page 5.2-24).
  When  GET_ATTRIBUTES
   Call MC_GET_ATTRIBUTES_PROC (page 5.2-24).
  When  POST_ON_RECEIPT
   Call MC_POST_ON_RECEIPT_PROC (page 5.2-25).
  When  PREPARE_TO_RECEIVE
   Call MC_PREPARE_TO_RECEIVE_PROC (page 5.2-26).
  When  RECEIVE_AND_WAIT
   Call MC_RECEIVE_AND_WAIT_PROC (page 5.2-27).
  When  REQUEST_TO_SEND
   Call MC_REQUEST_TO_SEND_PROC (page 5.2-37).
  When  SEND_DATA
    Call MC_SEND_DATA_PROC (page 5.2-38).
  When  SEND_ERROR
    Call MC_SEND_ERROR_PROC (page 5.2-40).
```

## 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 initializes the mapped conversation fields in the RCB that is created by the ALLOCATE verb and 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 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:

| | |
|---|---|
| PS_VERB_ROUTER | page 5.0-12 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| RCB | page A-7 |

```
If the transaction program supports mapped conversations and
 the mode name is not SNASVCMG (see Note 1) then
   Call PS_VERB_ROUTER (Chapter 5.0) to issue an ALLOCATE
    verb using the parameters given with the MC_ALLOCATE verb and
    specifying that the conversation type is mapped.
   Set the return code to the value returned by the ALLOCATE verb.
   If the return code from ALLOCATE was OK or ALLOCATION_ERROR then
      Prepare to return the ID of the RCB created by the ALLOCATE verb.
      Initialize RCB.MAPPER_SAVE_AREA as required by the implementation.
      Set RCB.MC_MAX_SEND_SIZE to the implementation limit on the
       length of RUs that can be sent to the partner LU.

Else (allocation of a conversation is not allowed)
   Call DEALLOCATION_CLEANUP_PROC (Chapter 5.0).
```

## MC_CONFIRM_PROC

| | |
|---|---|
| FUNCTION: | This procedure processes MC_CONFIRM verbs. |
| INPUT: | MC_CONFIRM verb parameters (See <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>.) |
| OUTPUT: | A return code as described in <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>. 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:

| | |
|---|---|
| RCVD_SVC_ERROR_PURGING | page 5.2-42 |
| PS_SPS | page 5.3-20 |
| UPM_MAPPER | page 5.2-46 |
| PS_VERB_ROUTER | page 5.0-12 |
| RCB | page A-7 |

```
Find the RCB for the specified conversation (resource).
Call PS_VERB_ROUTER (Chapter 5.0) to issue a CONFIRM verb
 for the current conversation.

Select based on the code returned by CONFIRM:
   When OK
      Set the return code to the value returned by the CONFIRM verb.
      Call UPM_MAPPER (page 5.2-46) to record a positive confirmation.
   When PROG_ERROR_PURGING
      Set the return code to the value returned by the CONFIRM verb.
      Call UPM_MAPPER (page 5.2-46) to record a remotely detected
       error of the type indicated by the return code from CONFIRM.
   When ALLOCATION_ERROR, RESOURCE_FAILURE_RETRY, or RESOURCE_FAILURE_NO_RETRY
      Set the return code to the value returned by CONFIRM.
   When DEALLOCATE_ABEND_PROG
      Set the 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).
      Set the return code to the value returned by CONFIRM.
   When SVC_ERROR_PURGING
      Call RCVD_SVC_ERROR_PURGING (page 5.2-42) to
       get and process error data from the remote TP.
      Set the 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 <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>.) |
| 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:

| | |
|---|---|
| PS_VERB_ROUTER | page 5.0-12 |

```
Call PS_VERB_ROUTER (Chapter 5.0) to issue a CONFIRMED
 verb for the current conversation.
```

## MC_DEALLOCATE_PROC

| | |
|---|---|
| FUNCTION: | This procedure handles the deallocation of mapped conversation resources. |
| INPUT: | MC_DEALLOCATE verb parameters (See <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>.) |
| OUTPUT: | A return code as described in <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>. |
| 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:

| | |
|---|---|
| RCVD_SVC_ERROR_PURGING | page 5.2-42 |
| PS_VERB_ROUTER | page 5.0-12 |
| UPM_MAPPER | page 5.2-46 |
| RCB | page A-7 |

```
Find the RCB for the specified conversation (resource).
If the deallocation type is ABEND then
   Clear RCB.MC_RECEIVE_BUFFER.
   Call PS_VERB_ROUTER (Chapter 5.0) to issue a DEALLOCATE
    verb for the current conversation with no error data and indicating
    that the type of deallocation is ABEND_PROG.
Else
   Call PS_VERB_ROUTER (Chapter 5.0) to issue a DEALLOCATE
    verb for the current conversation with no error data and the
    specified deallocation type.

 Select based on the return code from DEALLOCATE:
   When OK, ALLOCATION_ERROR, RESOURCE_FAILURE_RETRY, or RESOURCE_FAILURE_NO_RETRY
      Set the return code to the code returned by DEALLOCATE.
   When PROG_ERROR_PURGING
      Set the 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 the return code to DEALLOCATE_ABEND.
   When DEALLOCATE_ABEND_SVC or DEALLOCATE_ABEND_TIMER
      Set the 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:

| | |
|---|---|
| PS_VERB_ROUTER | page 5.0-12 |

Call PS_VERB_ROUTER (Chapter 5.0) to issue a FLUSH verb for the current conversation.

## 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 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. |

Issue a basic GET_ATTRIBUTES verb on the current conversation.

Return the attributes of the mapped conversation, returned from the GET_ATTRIBUTES verb, to the TP, such as the fully qualified LU names of both LUs of the conversation, the mode name, and the synchronization level.

## 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:

| | |
|---|---|
| RCB | page A-7 |

```
If the RCB.MC_RECEIVE_BUFFER for the current conversation is empty then
   Issue a basic POST_ON_RECEIPT verb 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.
```

## MC_PREPARE_TO_RECEIVE_PROC

| | |
|---|---|
| FUNCTION: | This procedure processes MC_PREPARE_TO_RECEIVE verbs.<br><br>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 <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>.) |
| 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 |
| RCVD_SVC_ERROR_PURGING | page 5.2-42 |
| PS_VERB_ROUTER | page 5.0-12 |
| RCB | page A-7 |

```
Find the RCB for the current conversation.
Call the PS_VERB_ROUTER (Chapter 5.0) to issue a
 PREPARE_TO_RECEIVE verb, specifying a LOCKS value and verb type,
 for the current RCB.
Select based on the returned PREPARE_TO_RECEIVE return code:
   When (OK, ALLOCATION_ERROR, RESOURCE_FAILURE_RETRY, RESOURCE_FAILURE_NO_RETRY)
     Set the MC_PREPARE_TO_RECEIVE return code to the PREPARE_TO_RECEIVE return code.
   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 the MC_PREPARE_TO_RECEIVE return code to DEALLOCATE_ABEND.
   When (DEALLOCATE_ABEND_SVC, DEALLOCATE_ABEND_TIMER)
      Set the MC_PREPARE_TO_RECEIVE 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 _SNA Transaction Programmer's Reference Manual for LU Type 6.2_.)

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 ( 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-7 |

```
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 the MC_RECEIVE_AND_WAIT return code to OK.
      When the buffer element contains a return code
         Set the MC_RECEIVE_AND_WAIT 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 is used by the WAIT verb to check that an incoming GDS variable is complete.

INPUT: The RCB for the current conversation and the type of test: POSTED or REQUEST_TO_SEND_RECEIVED

OUTPUT: A return code value of OK, OK--DATA, or UNSUCCESSFUL, the return code stored in 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 |
| PROTOCOL_ERROR_PROC | page 5.2-47 |
| PROCESS_ERROR_OR_FAILURE_RC | page 5.2-31 |
| PS_VERB_ROUTER | page 5.0-12 |
| RCB | page A-7 |

```
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_PROC 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_PROC in RCB.MC_RECEIVE_BUFFER.
            Otherwise
               Call PROCESS_ERROR_OR_FAILURE_RC (page 5.2-31)
                to process the return code from TEST_PROC.
         If RCB.MC_RECEIVE_BUFFER is empty or contains a map name or
          unmapped data (see Note 2) then
            Set the code to be returned by this routine to UNSUCCESSFUL.
            Call PS_VERB_ROUTER (Chapter 5.0) 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 code returned by this routine to OK--DATA.
               When it is a return code
                  Set the code returned by this routine to the return
                   code found 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 the code to be returned by this routine to OK--DATA.
         Else
            Set the code to be returned 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.<br><br>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-7 |

```
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 5.3-20 |
| 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-7 |

```
Select based on the RECEIVE_AND_WAIT return code being processed:
   When ALLOCATION_ERROR (see Note 2)
      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 parameter 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 code it returns 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.<br><br>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.<br><br>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.<br><br>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.<br><br>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-7 |

```
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-7 |

```
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.<br><br>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 5.0-12 |
| RCB | page A-7 |

```
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.<br><br>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:

| | |
|---|---|
| PS_VERB_ROUTER | page 5.0-12 |

Call PS_VERB_ROUTER (Chapter 5.0) to issue a REQUEST_TO_SEND verb for the current conversation.

## 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 <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>.) |
| 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:

| | |
|---|---|
| RCVD_SVC_ERROR_PURGING | page 5.2-42 |
| PS_SPS | page 5.3-20 |
| RCB | page A-7 |
| PS_VERB_ROUTER | page 5.0-12 |
| UPM_MAPPER | page 5.2-46 |
| SEND_BUFFER | page 5.2-48 |

```
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 the PS_VERB_ROUTER (Chapter 5.0) to issue a
         SEND_DATA verb, specifying the SEND_BUFFER and the length of the data
         to send, for the current RCB.
        If the SEND_DATA verb processing resulted in a saved request in the
         current RCB, from the remote TP, to send data then
           Save this request to be returned to the local TP on the MC_SEND_DATA verb.
        Select based on the SEND_DATA return code:
           When OK do nothing.
           When ALLOCATION_ERROR, RESOURCE_FAILURE_RETRY, or RESOURCE_FAILURE_NO_RETRY
              Set the MC_SEND_DATA return code to the SEND_DATA return code.
           When DEALLOCATE_ABEND_PROG
              Set the MC_SEND_DATA return code to DEALLOCATE_ABEND.
           When DEALLOCATE_ABEND_SVC or DEALLOCATE_ABEND_TIMER
              Set the MC_SEND_DATA reurn code to RESOURCE_FAILURE_NO_RETRY.
           When PROG_ERROR_PURGING
              Set the MC_SEND_DATA return code to the SEND_DATA return code.
              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).
              Set the MC_SEND_DATA return code to the SEND_DATA return code.
           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:

| | |
|---|---|
| RCVD_SVC_ERROR_PURGING | page 5.2-42 |
| PS_SPS | page 5.3-20 |
| PS_VERB_ROUTER | page 5.0-12 |
| UPM_MAPPER | page 5.2-46 |
| RCB | page A-7 |

```
Find the RCB for the specified conversation.
Clear RCB.MC_RECEIVE_BUFFER.
Call PS_VERB_ROUTER (Chapter 5.0) to issue a SEND_ERROR
 verb for the current conversation without data for the system error log
 and indicating that the request originated from the transaction program.

Select based on the return code from SEND_ERROR:
   When OK
      Set the return code to the code returned by 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 the return code to the code returned by 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, RESOURCE_FAILURE_RETRY, or
    RESOURCE_FAILURE_NO_RETRY
      Set the return code to the code returned by SEND_ERROR.
   When DEALLOCATE_ABEND_PROG
      Set the 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 (Chapter 5.3).
      Set the return code to the code returned by SEND_ERROR.
   When SVC_ERROR_PURGING
      Call RCVD_SVC_ERROR_PURGING (page 5.2-42).
      Set the return code to the code returned by 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 date 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 |
| PS_VERB_ROUTER | page 5.0-12 |
| PROTOCOL_ERROR_PROC | page 5.2-47 |
| RCB | page A-7 |
| 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 PS_VERB_ROUTER (Chapter 5.0) to issue a RECEIVE_AND_WAIT
 verb for the current conversation, specifying a wait for
 the receipt of a complete logical record.
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 Appendix H), and ERROR_DATA_STRUCTURE.ERROR_CODE
       indicates a map execution failure (see Appendix H) 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 |
| PS_VERB_ROUTER | page 5.0-12 |
| PROCESS_ERROR_DATA | page 5.2-43 |
| GET_SEND_INDICATOR | page 5.2-44 |
| PROTOCOL_ERROR_PROC | page 5.2-47 |
| RCB | page A-7 |
| 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 PS_VERB_ROUTER (Chapter 5.0) to issue a RECEIVE_AND_WAIT
 verb for the current conversation, specifying a wait for
 the receipt of a complete logical record.
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 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 the return code to the code returned by PROCESS_ERROR_DATA.
         If the return code is not RESOURCE_FAILURE_NO_RETRY then
            Call GET_SEND_INDICATOR (page 5.2-44).
      Else (Note 1)
         Call PROTOCOL_ERROR_PROC (page 5.2-47)
          to deallocate the current conversation.
         Set the return code to RESOURCE_FAILURE_NO_RETRY.
   When RESOURCE_FAILURE_RETRY or RESOURCE_FAILURE_NO_RETRY (Note 2)
      Set the return code to the code returned by RECEIVE_AND_WAIT.
   When PROG_ERROR_NO_TRUNC, SVC_ERROR_NO_TRUNC, or BACKED_OUT (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 (Note 1)
      Optionally log implementation-dependent error data.
      Set the 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 Appendix H)
      Select based on the GDS_ID in ERROR_DATA_STRUCTURE.ERROR_PARM:
         When it indicates user control data (see Appendix H)
            Set the return code to FMH_DATA_NOT_SUPPORTED.
            Optionally log implementation-dependent error data.
         When it indicates map name (see Appendix H)
            Set the 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 Appendix H)
      Set the return code to MAP_NOT_FOUND.
      Optionally log implementation-dependent error data.
   When it indicates map execution failure (see Appendix H)
      Set the 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:

| | |
|---|---|
| PS_VERB_ROUTER | page 5.0-12 |
| PROTOCOL_ERROR_PROC | page 5.2-47 |
| RCB | page A-7 |

```
Call PS_VERB_ROUTER (Chapter 5.0) to issue a RECEIVE_AND_WAIT
 verb for the current conversation, specifying a wait for
 the receipt of a complete logical record.
Select based on the return code from 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 the return code to RESOURCE_FAILURE_NO_RETRY.
   When RESOURCE_FAILURE_RETRY or RESOURCE_FAILURE_NO_RETRY
      Set the return code to the code returned by RECEIVE_AND_WAIT.
   When DEALLOCATE_NORMAL, DEALLOCATE_ABEND_PROG,
    DEALLOCATE_ABEND_SVC, or DEALLOCATE_ABEND_TIMER
    (optional check when receiving data)
      Set the 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 the 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 corresponding to 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 |
| PS_VERB_ROUTER | page 5.0-12 |
| PROTOCOL_ERROR_PROC | page 5.2-47 |
| RCB | page A-7 |
| ERROR_DATA_STRUCTURE | page 5.2-48 |

```
If the input error code indicates an invalid GDS_ID (see Appendix H) and the GDS_ID in the
 input error parameter does not indicate a map name (see Appendix H) 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 PS_VERB_ROUTER (Chapter 5.0) to issue a SEND_ERROR verb for the current
 conversation, specifying error type SVC and implementation-dependent error log data.
Select based on the return code from 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 PS_VERB_ROUTER (Chapter 5.0) 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 PS_VERB_ROUTER (Chapter 5.0) 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 transaction program determines the kind of mapping that occurs.
- 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. FMH data indicator indicates whether the passed data includes function management (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 transaction program 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 only used if the mapper invocation is for data mapping.

5. Input data contains the data supplied by the transaction program on the MC_SEND_DATA verb for SEND mapping, or data that flows in a data GDS variable for RECEIVE mapping. Again, 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 transaction program 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 transaction program for receive mapping. Again, 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:

| | |
|---|---|
| PS_VERB_ROUTER | page 5.0-12 |
| RCB | page A-7 |

```
Call PS_VERB_ROUTER (Chapter 5.0) to issue a DEALLOCATE verb for the
 current conversation, 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 Appendix H)
   DATA
      ERROR_CODE (see Appendix H)
      ERROR_PARM (see Appendix H)
```

| 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 service.[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.
- 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 and the physical components over which the sessions are carried. The reactivation of failed sessions is handled by system logic; see "Chapter 4. LU Network Services" 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 the sync point service.
- 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 5.3-13.

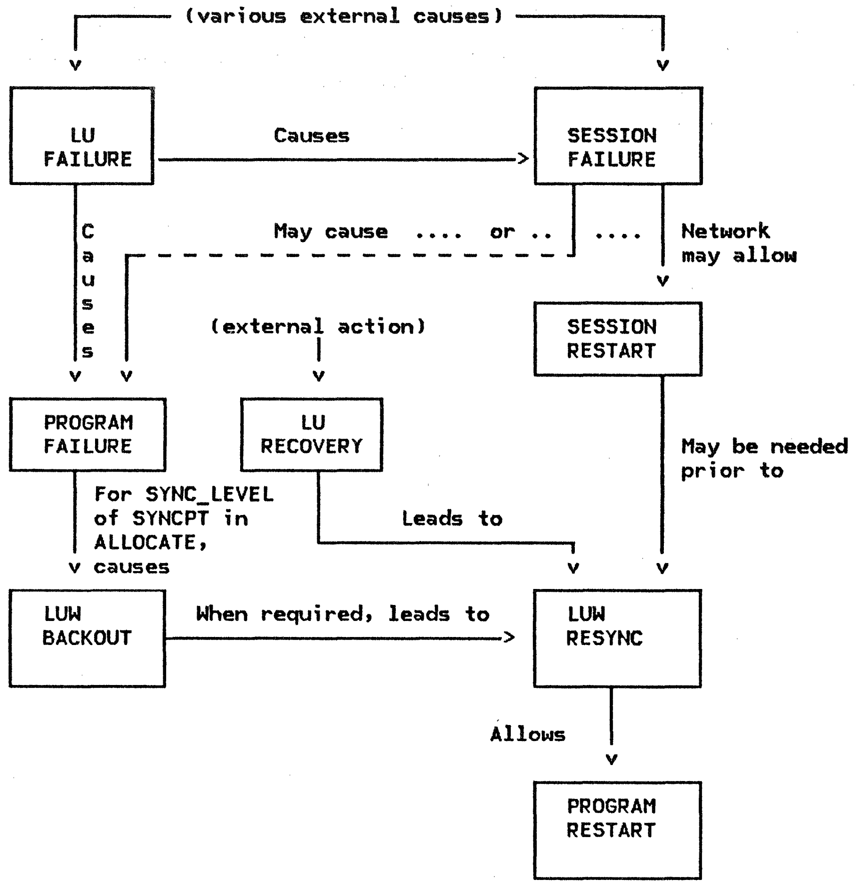


Figure 5.3-1. Relationships among Failures and Recovery

## 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 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 encodings for transmission of these commands are described in "Appendix H. FM Header and LU Services Commands" under PS headers for the first four, and FMH-7 sense data for Backed Out.

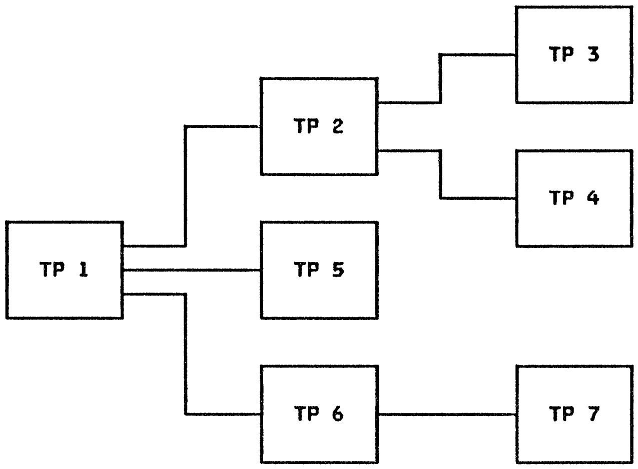


Figure 5.3-2. A Typical Sync Point Tree

## 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 nodes. 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, the distributed LUW state is a vector with seven components:

LUW = [LUW1,LUW2, ... LUW7]

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 on page 5.3-4 shows these waves as they occur between the root and one of the nodes adjacent to the root.

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.

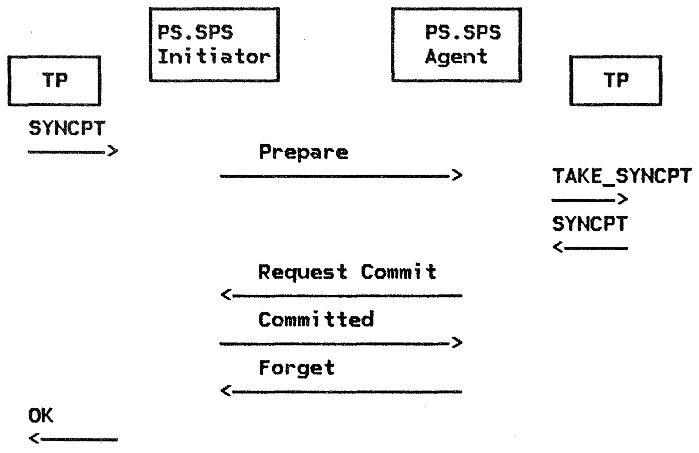


Figure 5.3-3. Basic Sync Point Flows

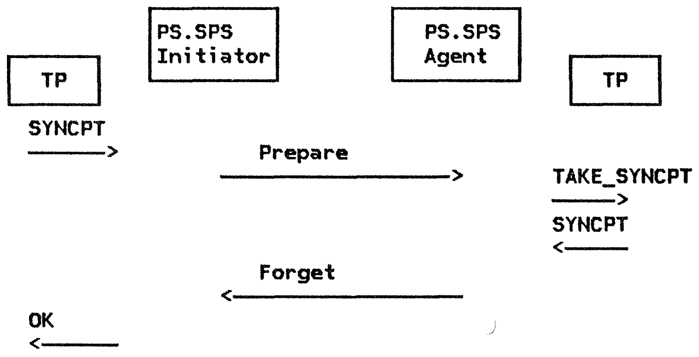


Figure 5.3-4. Optimized Flow: No Resource Changed

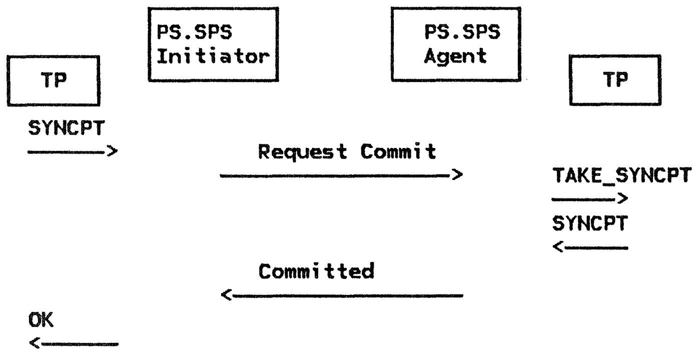


Figure 5.3-5. Optimized Flow: Last Resource

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 on page 5.3-4 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.

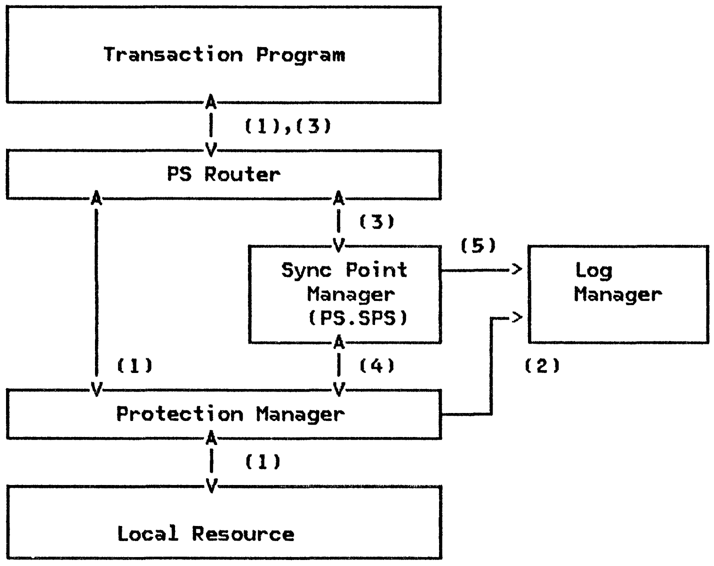


Figure 5.3-6. Sync Point Services for Local (Nonconversational) Resources, Such as Files

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.

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 signals indicating Prepare, Request Commit, Committed, and Forget to coordinate commitment, or a signal 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

[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 forced writes) to the log must

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 5.3-15.

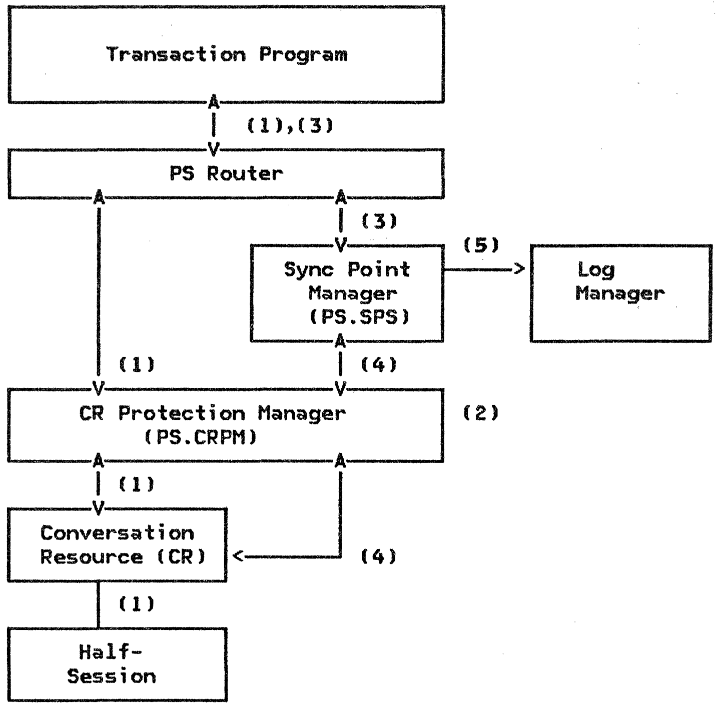

Transaction Program
(1),(3)
PS Router
(3)
(5)
Sync Point Manager (PS.SPS)
Log Manager
(1)
(4)
CR Protection Manager (PS.CRPM)
(2)
(1)
Conversation Resource (CR)
(4)
(1)
Half-Session

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 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 recreate the resource control blocks (RCB) and their relationship to transaction control blocks (TCB) following an LU failure. See RCB on page A-7 for details of the RCB and TCB on page A-10 for details of the 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 signals 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 local/nonlocal[6] indicator in the RCB. A Backed Out signal can be received from nonlocal resources. Compare States signals 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.

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 _forcing the log_.

[6] Local resources are those that share the sync point manager's log.

Also, additional forces of the log are required. Finally, the sync point manager attempts resynchronization with an exchange of Compare States signals with its partner sync point manager after resource failures.

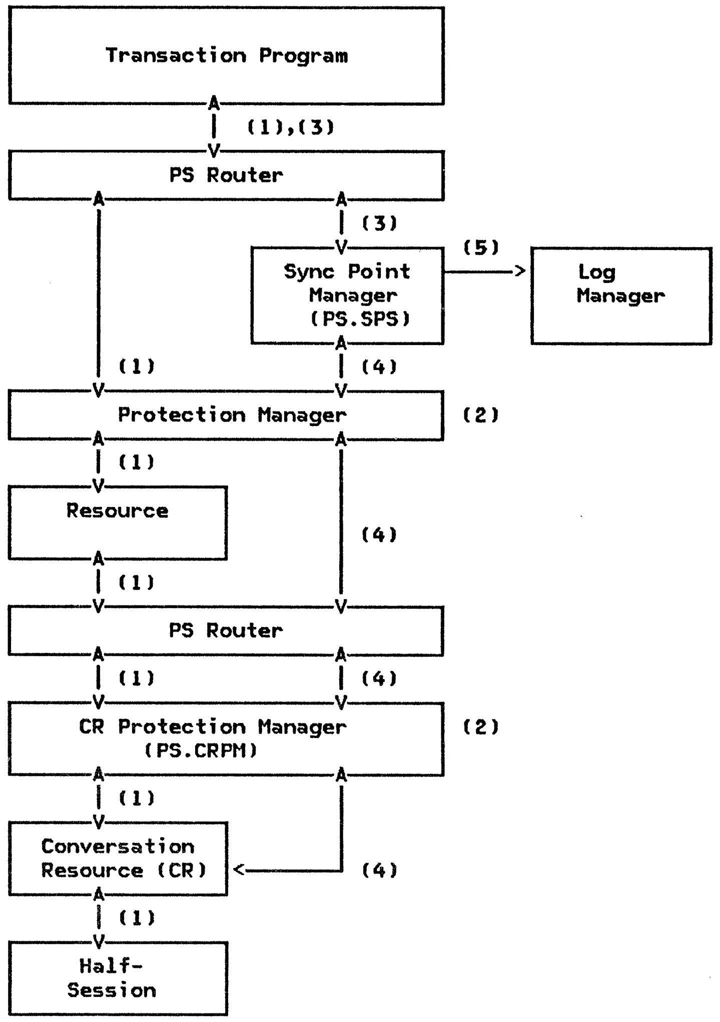


Figure 5.3-8. Sync Point Services for Function Shipping

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 recreate 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 signals 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 signals 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 sync point signals 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 signals 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 SYNCPT_REQUIRED 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 commits 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, a state error is recognized and the TP is abnormally terminated. Otherwise, PS.SPS advances to the Prepare phase.

### PREPARE PHASE

PS.SPS now issues Prepare to all nonlast 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 5.3-13. If no nonlast 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 nonlast resources, PS.SPS issues Request Commit to the last resource, and waits for a reply, thus entering the Committed phase.

### COMMITTED PHASE

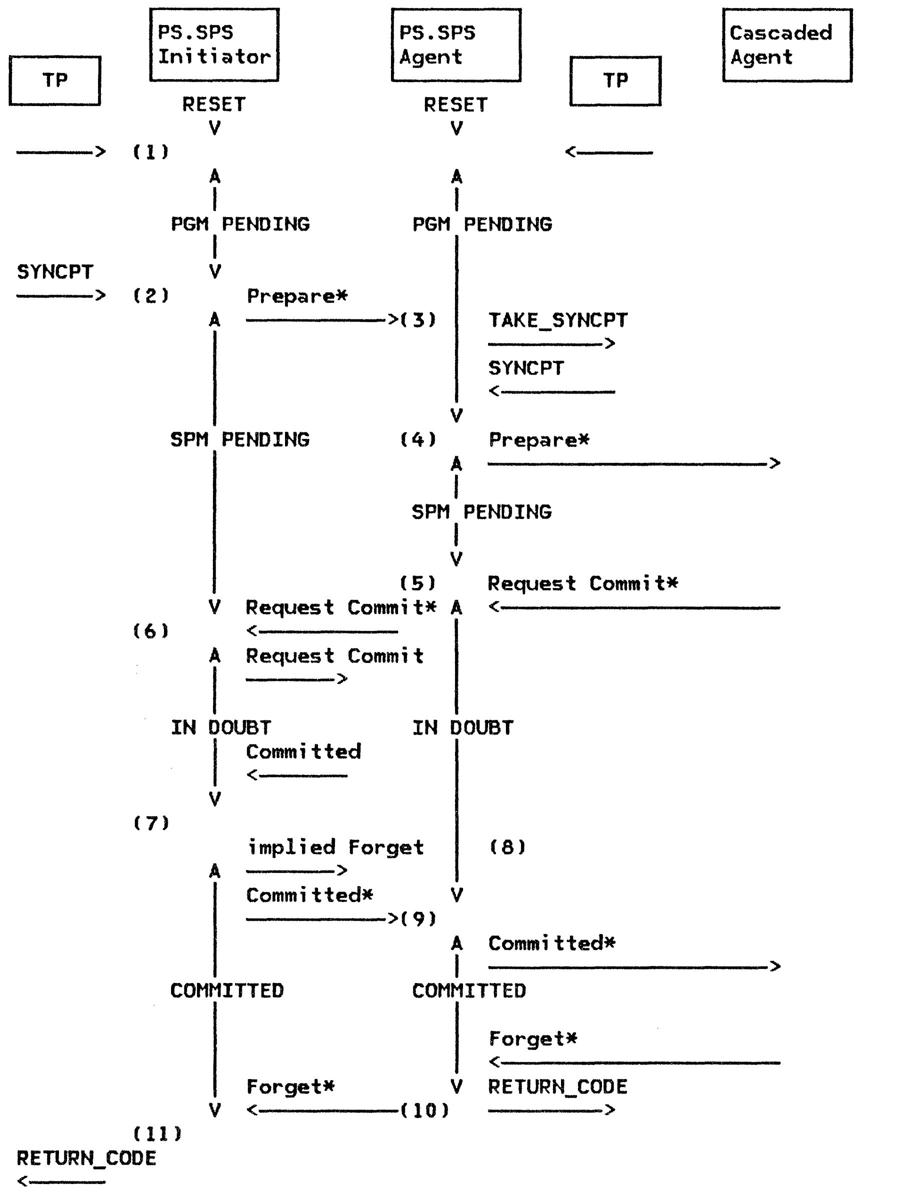
PS.SPS completes sync point processing after receiving Committed from the last resource, by sending Committed to all nonlast resources, thus entering the Forget phase.

### FORGET PHASE

In the Forget phase, PS.SPS waits for Forgets from all the nonlast 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.



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 presents a return code of TAKE_SYNCPT to its transaction program. All TPs agree by issuing SYNCPT.

4. The agent 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. [AGEBT, IN DOUBT] is placed on the log and the log is forced.

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 last agent is using the optimized flow). [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 5.3-15 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
```

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 heuristic damage that may occur.

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 are collected. See Figure 5.3-9 on page 5.3-9 for this type of sequence.

9. IMPLIED_FORGET is sent to the last agent with the aid of the session process.

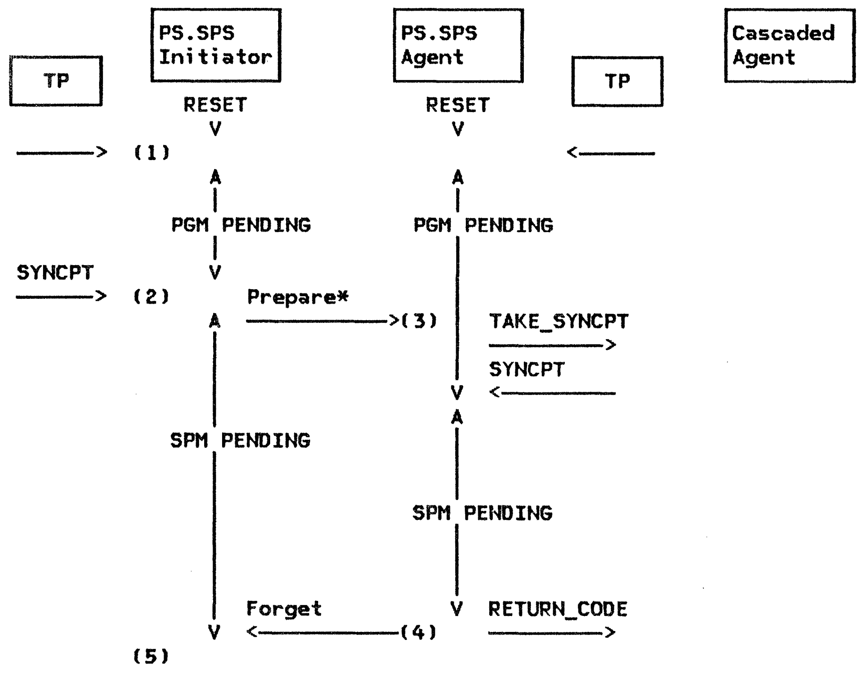


Figure 5.3-11. Illustrative Sync Point Flow: No Resources Changed

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.

## 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 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. Contrary to 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 must propagate 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, an attempt will be made to recover from the failure by exchanging COMPARE_STATES GDS variables with the remote sync point manager. PS.SPS does this by issuing ALLOCATE to the LU resync service TP X'06F2'. See "Resynchronization Logic" on page 5.3-15 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. PS gives the RESOURCE_FAILURE_* return code to the TP on the next verb issued against the failing conversation, thus making the sync point verb and the resource failure appear to have occurred in the reverse order. 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.

   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 in the TP's entry in the TRANSACTION_PROGRAM list in RM 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. As PS.SPS makes this decision with only partial information, it is called a heuristic decision. 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 will 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.

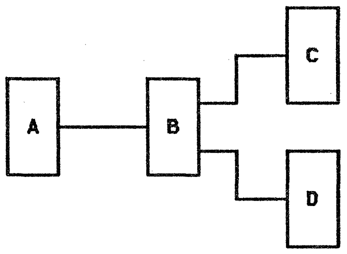
## BACKOUT PROCESSING

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.

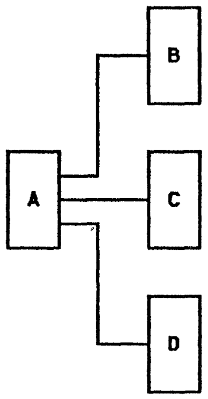
When a transaction program is notified of a BACKOUT initiated by another transaction program, the remote BACKOUT is complete. That is, the conversation resource that reports BACKED_OUT has already done so. The return code indicating this, BACKED_OUT, may be returned on several of the verbs. No backout 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

PS.SPS includes resynchronization logic for these cases:

- 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, 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.

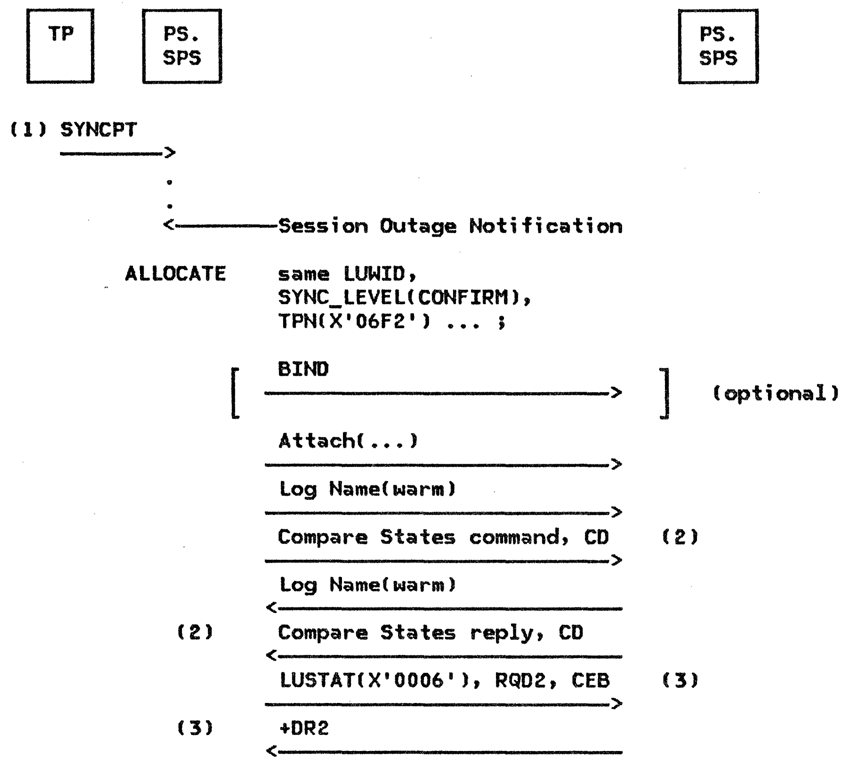
- Since it can happen that multiple conversations connect TCBs with the same LUW IDs in two separate LUs, resynchronization uses the conversation correlator field carried in Attach. to uniquely identify the states to be compared.

The decision to initiate resync by either end is dependent 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 |
|---|---|
| Not Found........... | No Action |
| Agent, not last .... | Wait for resync |
| Agent, last ........ | Resync after timeout |
| 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 the resync logic performed by the sync point manager.



TP
PS. SPS
PS. SPS
(1) SYNCPT
Session Outage Notification
ALLOCATE
same LUWID, SYNC_LEVEL(CONFIRM), TPN(X'06F2') ... ;
BIND
(optional)
Attach(...)
Log Name(warm)
Compare States command, CD
(2)
Log Name(warm)
(2)
Compare States reply, CD
LUSTAT(X'0006'), RQD2, CEB
(3)
(3)
+DR2

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 to the resync transaction program, X'06F2'. The optional BIND may flow as a result of RM logic; PS.SPS does not know if it flows. PS retrieves the LUWID carried in this Attach from the TCB.

2. PS.SPS validates the logname 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.

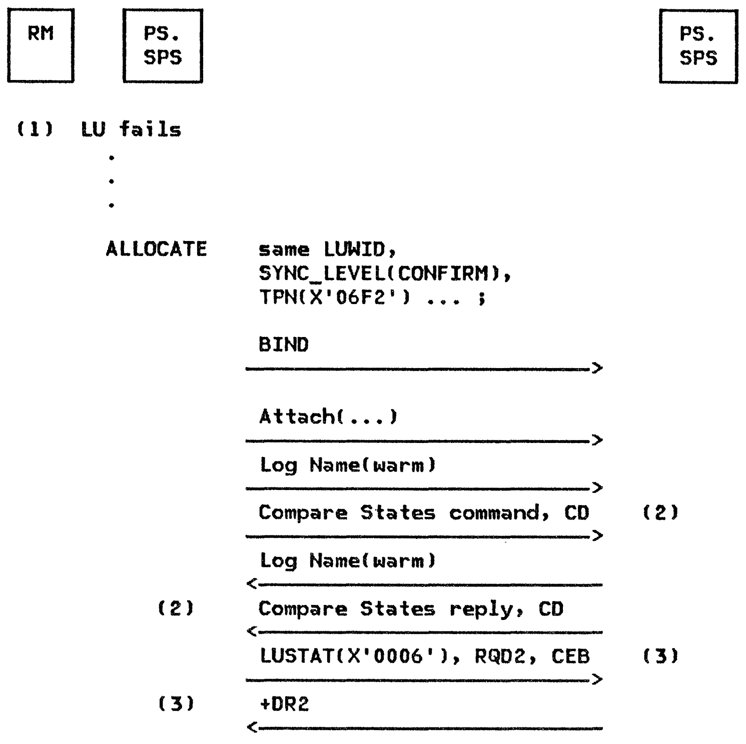

RM
PS. SPS
PS. SPS
(1) LU fails
ALLOCATE same LUWID, SYNC_LEVEL(CONFIRM), TPN(X'06F2') ... ;
BIND
Attach(...)
Log Name(warm)
Compare States command, CD (2)
Log Name(warm)
(2) Compare States reply, CD
LUSTAT(X'0006'), RQD2, CEB (3)
(3) +DR2

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 (a UPM determines the degree of parallelism), 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.

2. PS.SPS validates the logname and then executes resync logic.

3. PS.SPS erases the log. If a conversation or LU failure occurs during resynhronization, PS.SPS repeats resynchronization until both logs are erased.

## LOG NAME PROCESSING

The following two figures illustrate the processing of log names so that log mismatches do not occur.

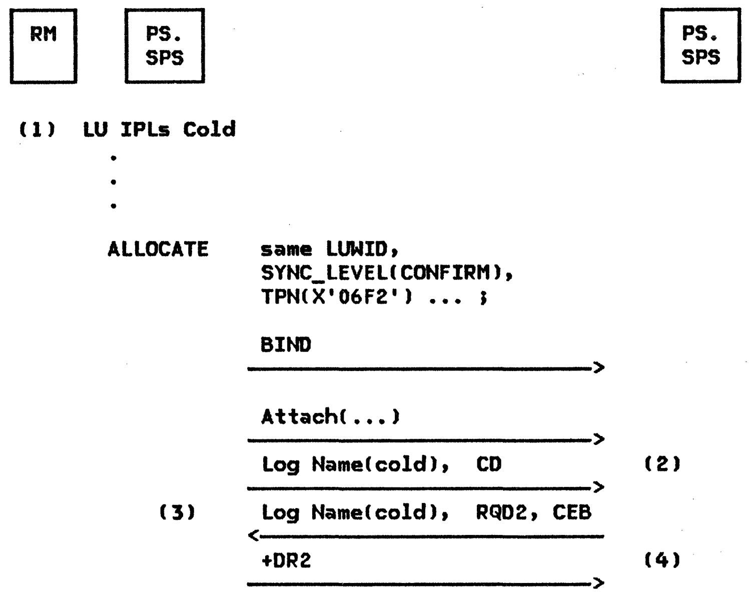


Figure 5.3-16. Cold Start of an LU

The following notes correspond to the numbers in Figure 5.3-16.

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.

2. The resync TP, X'06F2', accepts the cold log name and returns its own LU's log name.

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.

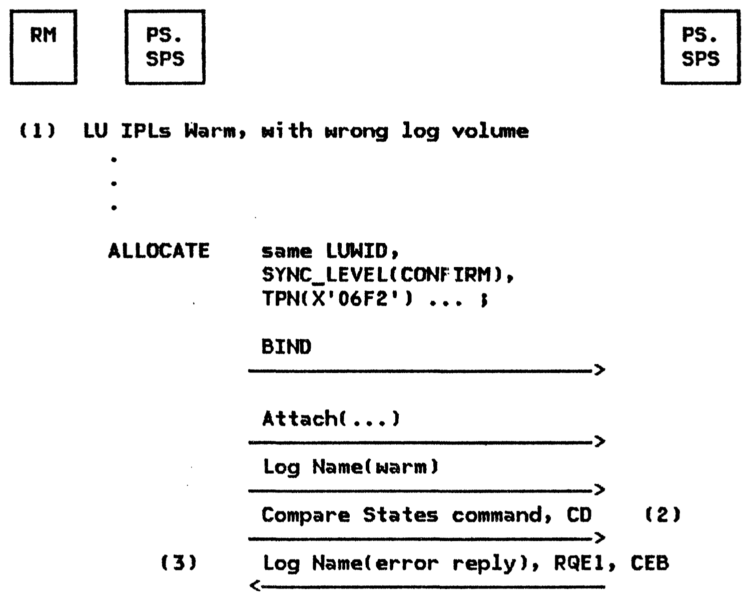


Figure 5.3-17. Log Name Mismatch during Resync

The following notes correspond to the numbers in Figure 5.3-17.

1. The LU IPLs warm, but the wrong log volume is active. However, RM and PS.SPS do not know this at first, so proceed with resync processing.

2. The partner PS.SPS detects the mismatch of log names, notifies its control operator, and returns an error reply.

3. PS.SPS sees the error reply and notifies its control operator of the mismatch. 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. |

# 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").

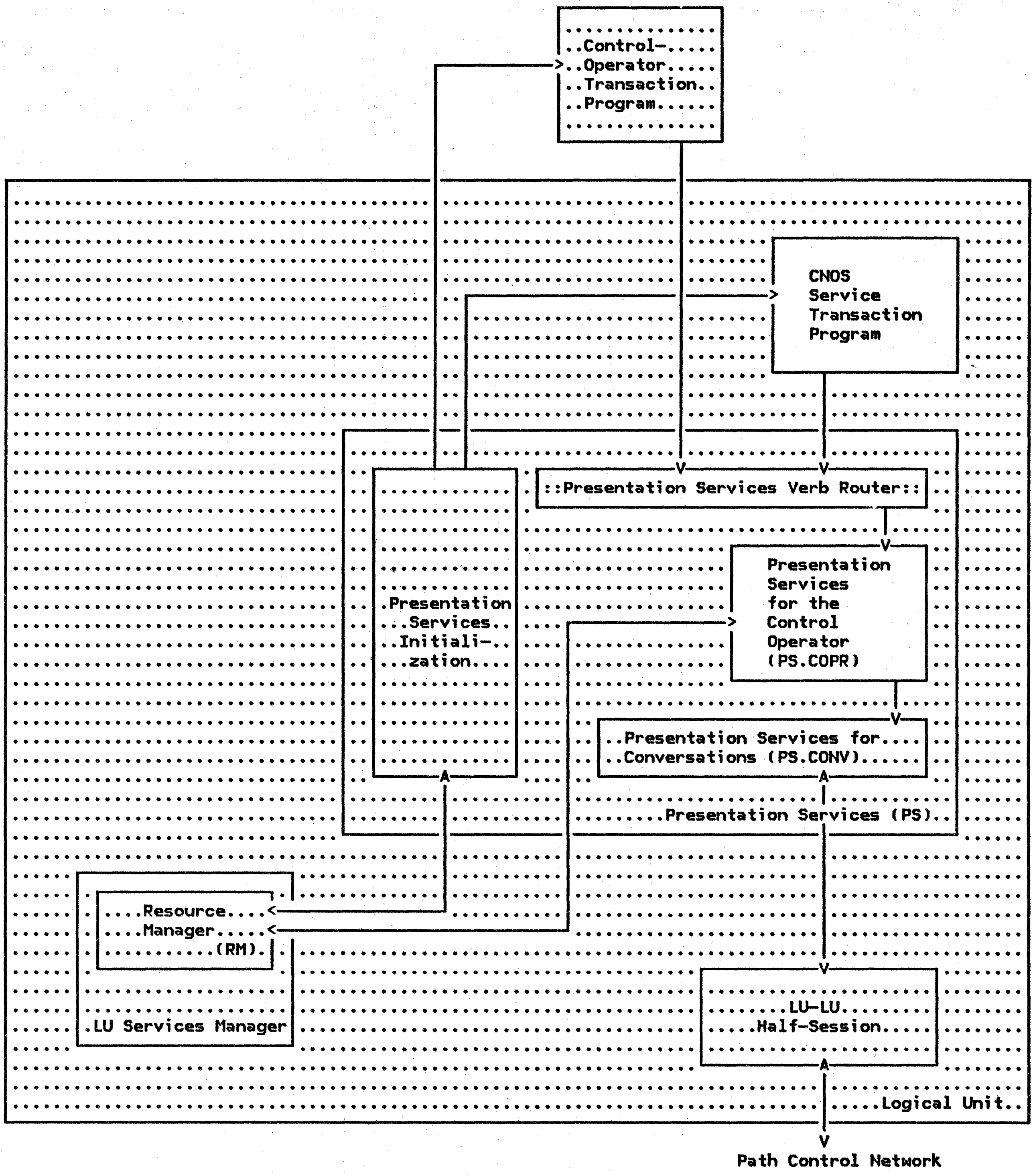
Figure 5.4-1 on page 5.4-2 shows the structural relationship of these components (see Chapter 2 for the complete structure of the LU).

## CONCEPTS AND TERMS

This section discribes 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.


Control-Operator Transaction Program
CNOS Service Transaction Program
Presentation Services Verb Router
Presentation Services Initialization
Presentation Services for the Control Operator (PS.COPR)
Presentation Services for Conversations (PS.CONV)
Presentation Services (PS)
Resource Manager (RM)
LU Services Manager
LU-LU Half-Session
Logical Unit
Path Control Network

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 <u>control-operator verb</u> is a privileged verb that may be issued by the control-operator transaction program to convey the operator's request to the internal components of the LU. Control-operator verbs are described in <u>SNA</u>

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
- A control point e.g., an SSCP, that provides session services during session initiation
- 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 LU parameters that characterize 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 at a time; (it can have successive sessions with different partner LUs selected from a group of LUs known to it). A parallel-session LU can have one or more concurrently active sessions with each of one or more LUs, subject to session limits discussed below. No middle capability exists, i.e., no LU supports concurrent sessions to multiple single-session LUs without also supporting multiple concurrent sessions (or parallel sessions) with any other parallel-session LU.

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 INITEATE (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 parameter 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-PARAMETER VERBS

LU-parameter verbs, DEFINE and DISPLAY, 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
- Transaction programs: the transaction program name, its availability, and the optional functions that it supports such as map names and sync point.
- Control point: the allowed formats of network addresses and session-services RUs used on the CP-LU session
- 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.
- 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.

### 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-parameter 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

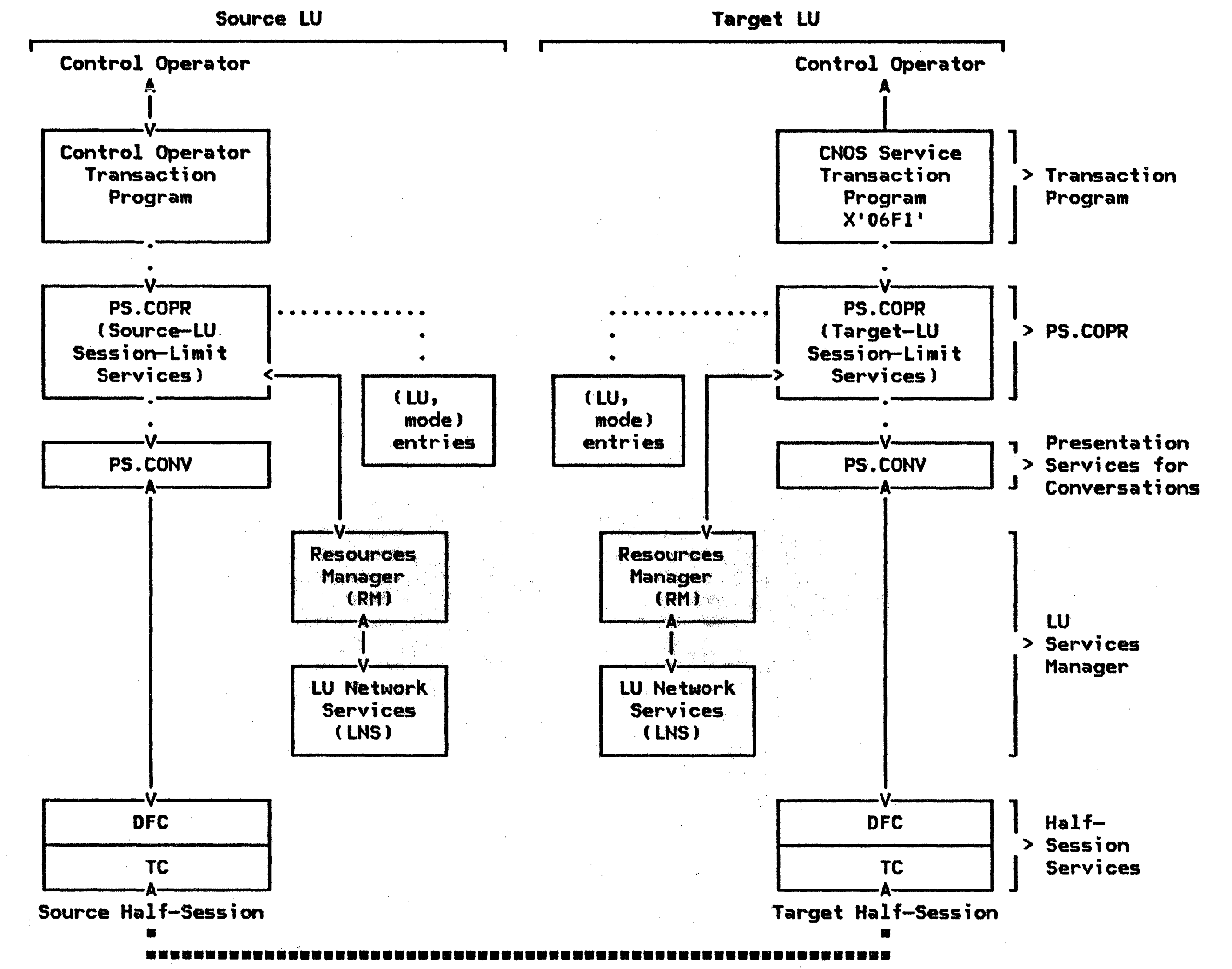
(The INITIALIZE_SESSION_LIMIT and RESET_SESSION_LIMIT verbs are included in both the local verbs and CNOS verbs. They are distinguished by the characteristics of their specified mode name.)

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 (<u>CNOS command</u>) 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 LU network services. 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 is 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 LU network services 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--LU Network Services

   Whenever LU network services 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, LNS issues the appropriate BIND or response to BIND; otherwise, it rejects the request.

   Whenever LU network services 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          Information          PS.COPR       CNOS
      Operator      Source LU        Exchanged            Target LU     Service
      Transaction                                                       Transaction
      Program                                                           Program
      o----------o-----------------------------------------------o---------o
      .          .                                               .         .
(1)   o *_SESSION_LIMIT                                          .         .
      .          .                                               .         .
(2)   .          o ALLOCATE                                      .         .
      .          .  TPN(X'06F1')  >- - attach - - - - - - - -┬- - - - - - > o X'06F1': PROCEDURE
      .          .                                          │    .         .
(3)   .          .                                          │    .         o PROCESS_SESSION_LIMIT
      .          .                                          │    .         .
(4)   .          .                                          ├ - > o GET_TYPE
      .          .                                          │    .         .
(5)   .          .                                          └ - > o GET_ATTRIBUTES
      .          .                                               .         .
(6)   .          o SEND_DATA                                     o RECEIVE_AND_WAIT
      .          .  DATA(command)  >- - command - - - - - - - - > .  DATA=command
      .          .                                               .         .
(7)   .          .                                               o RECEIVE_AND_WAIT
      .          o RECEIVE_AND_WAIT >- - change direction - - - -> .  WHAT_RECEIVED=SEND
      .          .                                               .         .
(8)   .          .                                               o (negotiate limits)
      .          .                                               .         .
(9)   .          .                                               o SEND_DATA
      .          .  DATA=reply  <- - - - - - - - - - - - reply - -< .  DATA(reply)
      .          .                                               .         .
(10)  .          o RECEIVE_AND_WAIT                              o DEALLOCATE
      .          .  RETURN_CODE=  <- - - - deallocate normal - -< .  TYPE(NORMAL)
      .          .    DEALLOCATE_NORMAL                          .         .
      .          .                                               .         .
(11)  .          o DEALLOCATE                                    .         .
      .          .  TYPE(LOCAL)                                  .         .
      .          .                                               .         .
(12)  .          o (update Limits;                               o (update limits;
                    inform resources manager)                       inform resources manager)
      .                                                                    .
(13)  o (inform operator)                                                  o (inform 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 lines 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.

3. When the target LU receives the Attach, it initiates the CNOS service transaction program. This program 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. TSLS issues the GET_TYPE verb to verify that this is a basic conversation.

5. 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 concurrent CNOS commands.

6. SSLS issues SEND_DATA to send the CNOS command to TSLS.

   Meanwhile, TSLS issues RECEIVE_AND_WAIT to receive the command.

7. 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.

8. TSLS negotiates the proposed session limit parameters and builds the CNOS reply.

9. 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.

10. 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.

11. SSLS issues DEALLOCATE to complete its processing of the conversation.

12. 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.

13. When SSLS and TSLS have finished processing the CNOS reply, they return 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
```

<u>Notes:</u>
- 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.

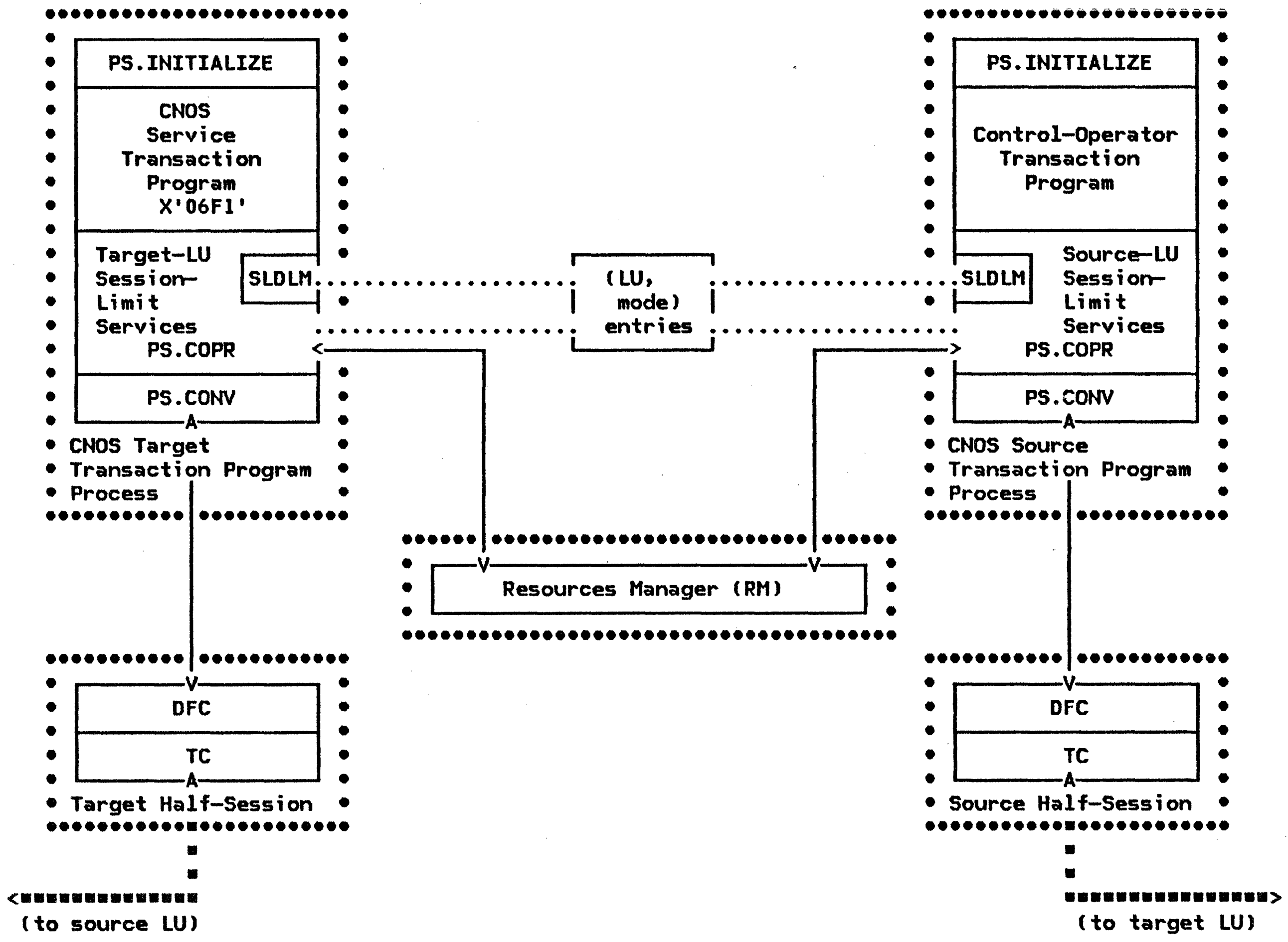
Figure 5.4-4. CNOS External Message-Unit Flows

## 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. 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.


PS.INITIALIZE
CNOS Service Transaction Program X'06F1'
Target-LU Session-Limit Services
SLDLM
PS.COPR
PS.CONV
CNOS Target Transaction Program Process
(LU, mode) entries
PS.INITIALIZE
Control-Operator Transaction Program
Source-LU Session-Limit Services
SLDLM
PS.COPR
PS.CONV
CNOS Source Transaction Program Process
Resources Manager (RM)
DFC
TC
Target Half-Session
DFC
TC
Source Half-Session
(to source LU)
(to target LU)

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 |

<u>Note:</u> Verb routers have been omitted.

Figure 5.4-5. CNOS Process Interactions at a Single LU

## THE CNOS PROCESS RELATIONSHIPS

### <u>Processes</u>

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.

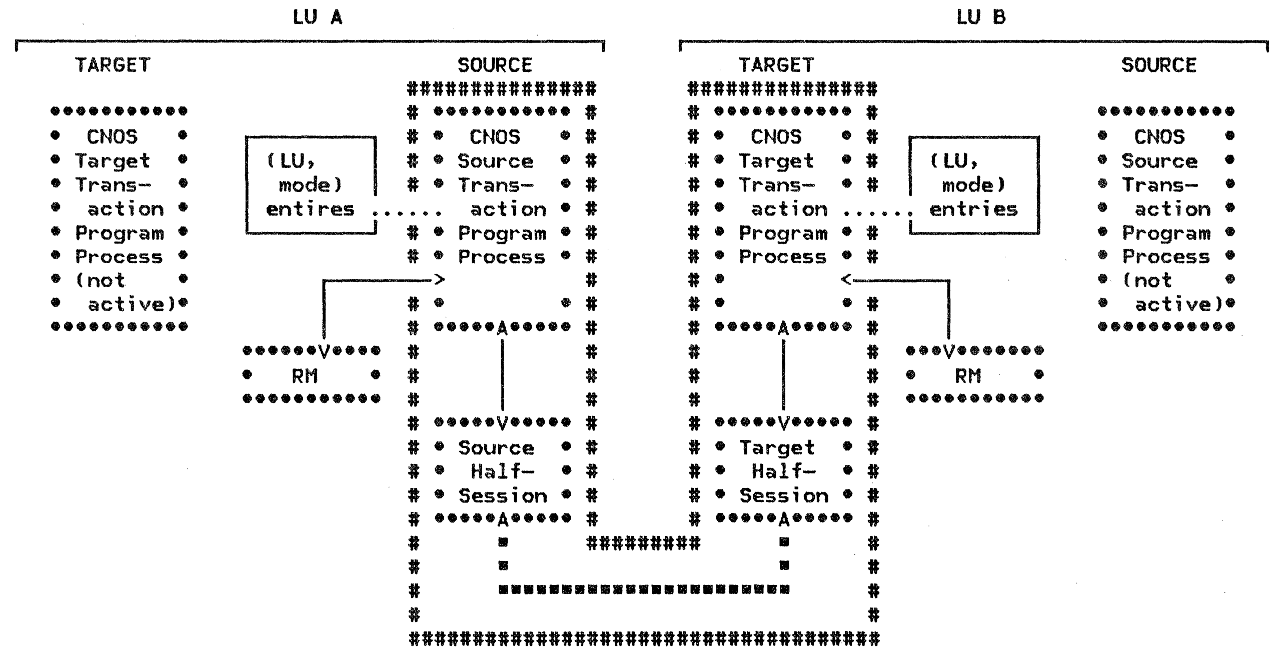
Multiple CNOS transaction-program processes, and corresponding half-session processes, can be active concurrently at any LU. For example, both the local control operator and a remote control operator might issue a CNOS verb at about the same time. Or two remote operators might both issue CNOS to the 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.


LU A
LU B
TARGET
SOURCE
TARGET
SOURCE
CNOS Target Trans- action Program Process (not active)
(LU, mode) entires
CNOS Source Trans- action Program Process
RM
Source Half- Session
CNOS Target Trans- action Program Process
(LU, mode) entries
RM
Target Half- Session
CNOS Source Trans- action Program Process (not active)

LEGEND:

| | |
|---|---|
| <———> | Send/receive relationship (between processes) |
| ...... | Access to shared data (within the LU) |
| ###### | Transaction—handling boundaries |
| ●●●●●● | Process boundaries |
| <■■■■> | Transaction program interaction (between LUs) |

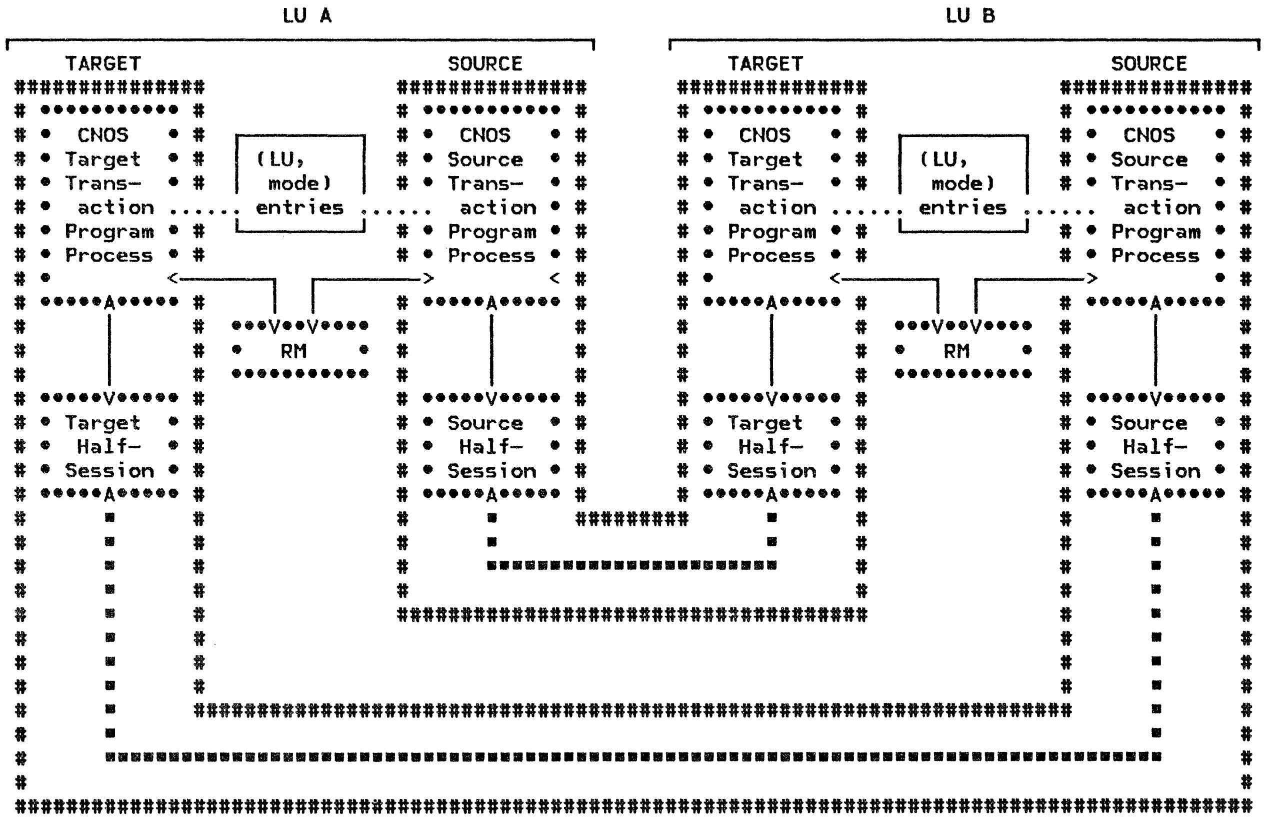
Figure 5.4-6. Transaction Handling Component Relationships--Case 1: Single Verb Issuance

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.

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.

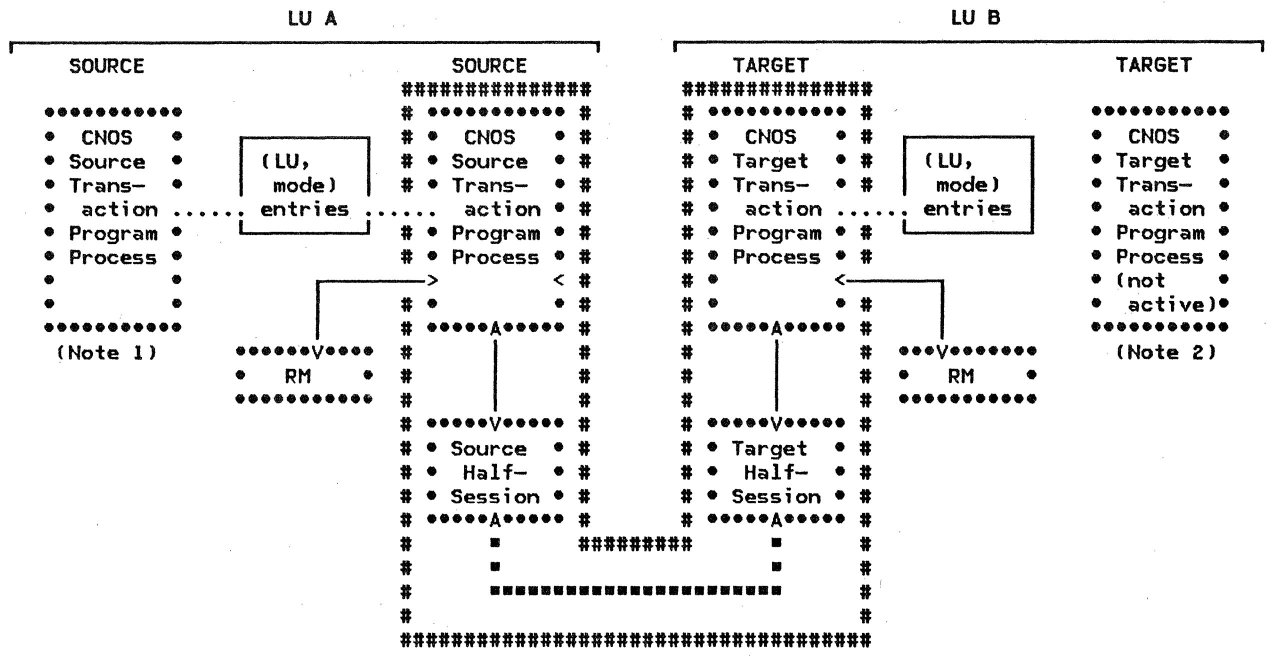

LU A
LU B
TARGET
SOURCE
CNOS Target Trans- action Program Process
CNOS Source Trans- action Program Process
(LU, mode) entries
RM
Target Half- Session
Source Half- Session

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

Simultaneous Verb Issuances at Partner LUs: 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.

Simultaneous Verb Issuances at the Same LU: 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 on page 5.4-14. 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, with the roles of source-LU and target-LU appropriately reversed.


LU A
LU B
SOURCE
SOURCE
TARGET
TARGET
CNOS Source Trans- action Program Process
(LU, mode) entries
CNOS Source Trans- action Program Process
CNOS Target Trans- action Program Process
(LU, mode) entries
CNOS Target Trans- action Program Process (not active)
(Note 1)
RM
RM
(Note 2)
Source Half- Session
Target Half- Session

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 in the LU_MODE_LIST 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 <u>command race</u> 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 LU network services have not yet completed adjustments to the session count, an <u>action race</u> has occurred and neither verb fails. For details, see <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u> 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 (abbreviated SLDLM hereafter), 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

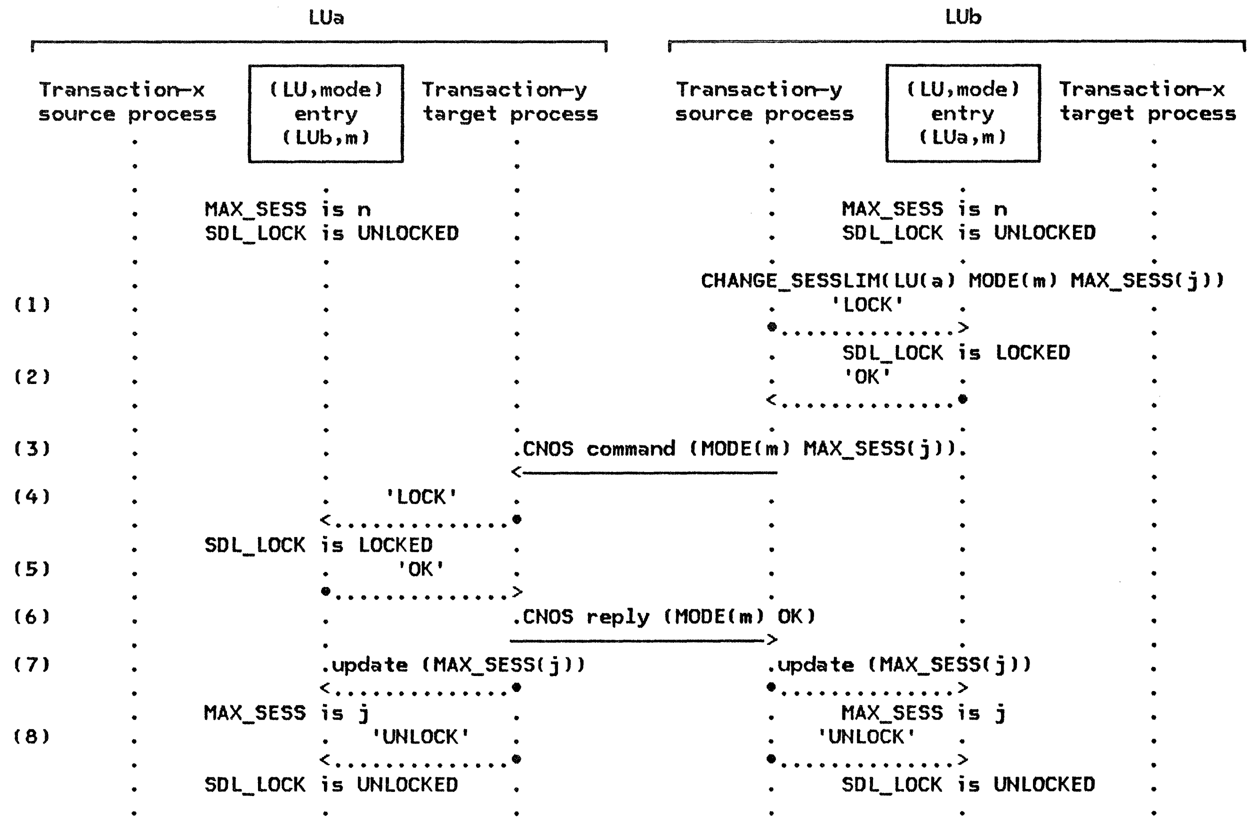
Example flows for the types of command races that can occur are shown in Figure 5.4-10 on page 5.4-17, Figure 5.4-11 on page 5.4-18, and Figure 5.4-12 on page 5.4-19. The flows for the no-race case are shown in Figure 5.4-9 on page 5.4-16 for comparison.

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



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 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.

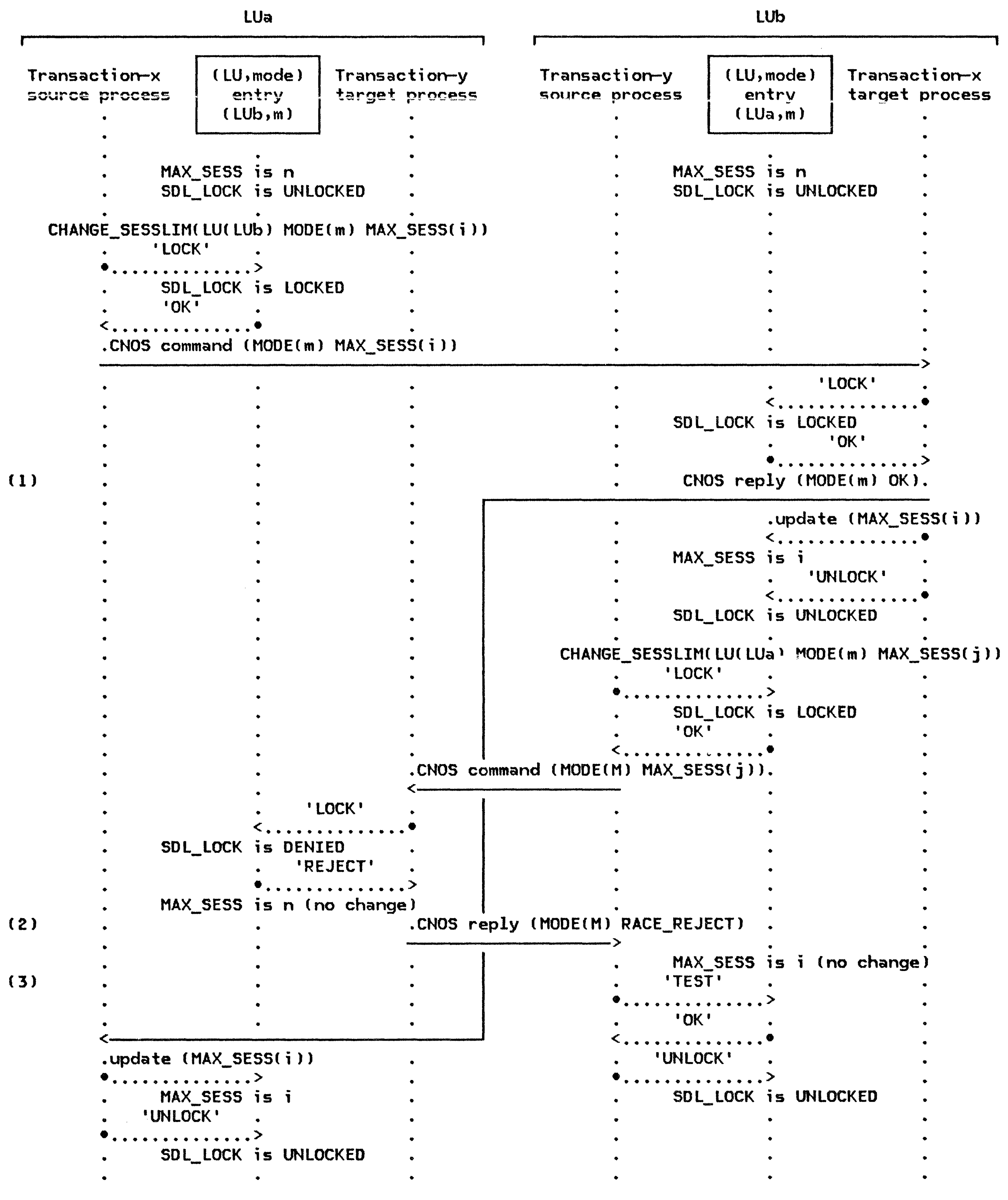
Single-Failure Races: In the single-failure cases (Figure 5.4-10 on page 5.4-17 and Figure 5.4-11 on page 5.4-18), one transaction fails; it does not modify the session-limit parameters in the (LU,mode) entry. The other transaction succeeds and changes the session-limit parameters.

Figure 5.4-10 on page 5.4-17 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



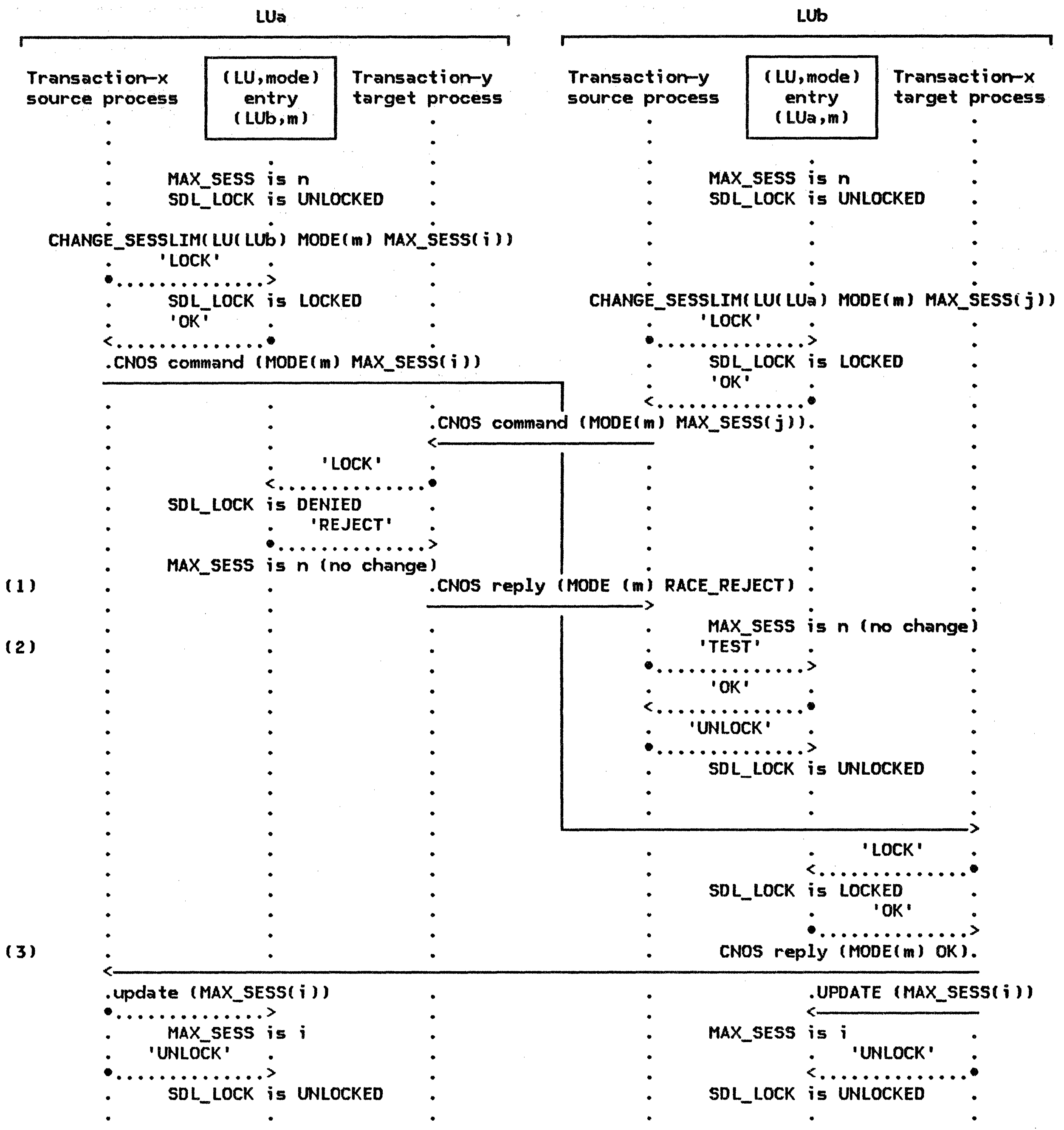
Note: Numbers in the left column refer to explanations in the text.

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 on page 5.4-18 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.

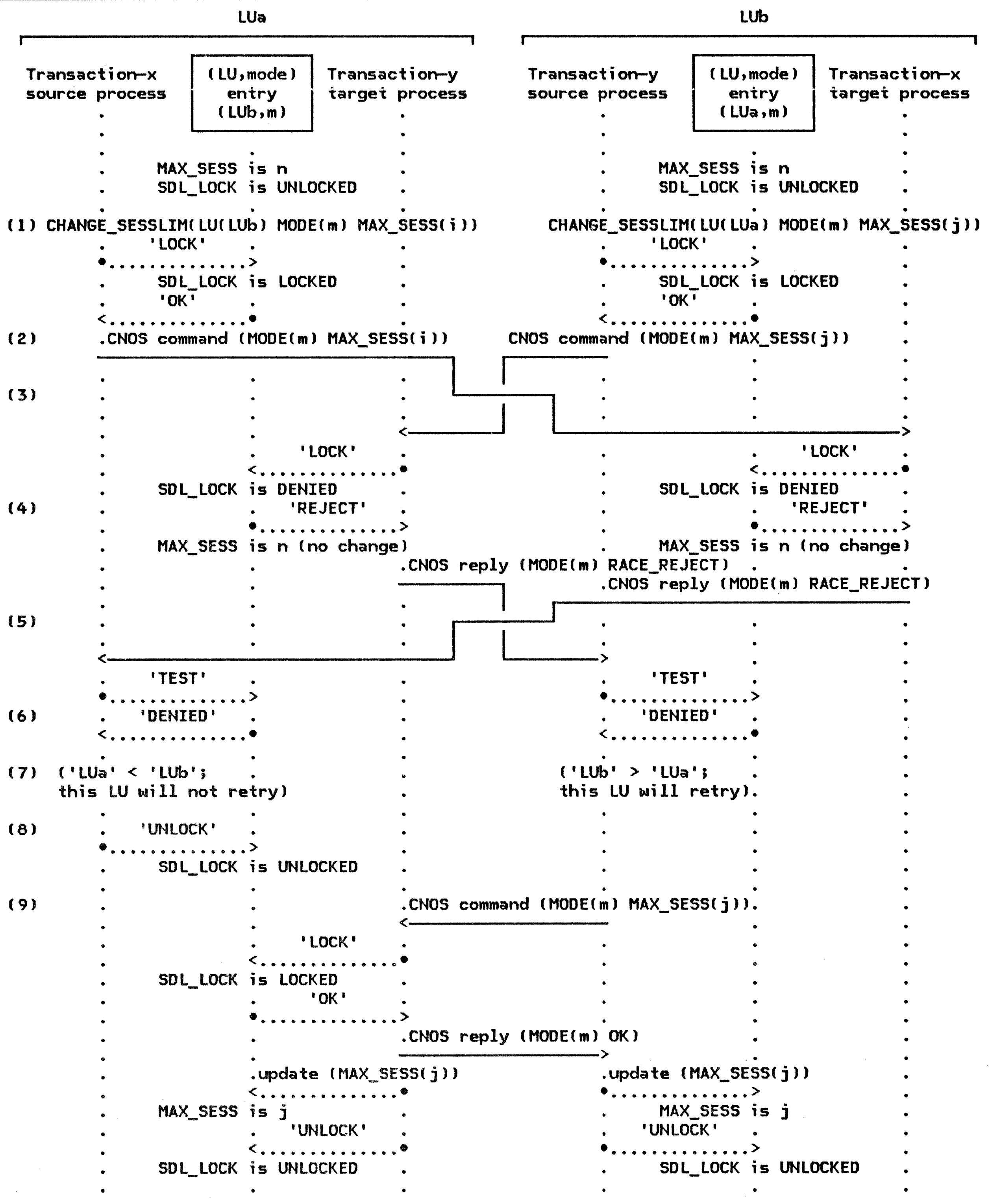


NOTE: Numbers in left column refer to the explanations in the text.

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 source process
(LU,mode) entry (LUb,m)
Transaction-y target process
Transaction-y source process
(LU,mode) entry (LUa,m)
Transaction-x target process
MAX_SESS is n
SDL_LOCK is UNLOCKED
MAX_SESS is n
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'
SDL_LOCK is LOCKED
SDL_LOCK is LOCKED
'OK'
'OK'
(2) CNOS command (MODE(m) MAX_SESS(i))
CNOS command (MODE(m) MAX_SESS(j))
(3)
'LOCK'
'LOCK'
SDL_LOCK is DENIED
SDL_LOCK is DENIED
(4) 'REJECT'
'REJECT'
MAX_SESS is n (no change)
MAX_SESS is n (no change)
CNOS reply (MODE(m) RACE_REJECT)
CNOS reply (MODE(m) RACE_REJECT)
(5)
'TEST'
'TEST'
(6) 'DENIED'
'DENIED'
(7) ('LUa' < 'LUb'; this LU will not retry)
('LUb' > 'LUa'; this LU will retry)
(8) 'UNLOCK'
SDL_LOCK is UNLOCKED
(9) CNOS command (MODE(m) MAX_SESS(j))
'LOCK'
SDL_LOCK is LOCKED
'OK'
CNOS reply (MODE(m) OK)
update (MAX_SESS(j))
update (MAX_SESS(j))
MAX_SESS is j
MAX_SESS is j
'UNLOCK'
'UNLOCK'
SDL_LOCK is UNLOCKED
SDL_LOCK is UNLOCKED

<u>Note:</u> Numbers in left column refer to explanations in the text.

Figure 5.4-12. Double-Failure Race Condition: Command Crosses Command, Reply Crosses Reply

<u>Double-Failure Race</u>: In the double-failure case (Figure 5.4-12), 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.

## BASIC 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-parameter 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 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 RESPONSIBLE(SOURCE) DRAIN_SOURCE(NO) DRAIN_TARGET(YES) DRAIN_SOURCE(YES)

  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"

- 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 LU network services 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

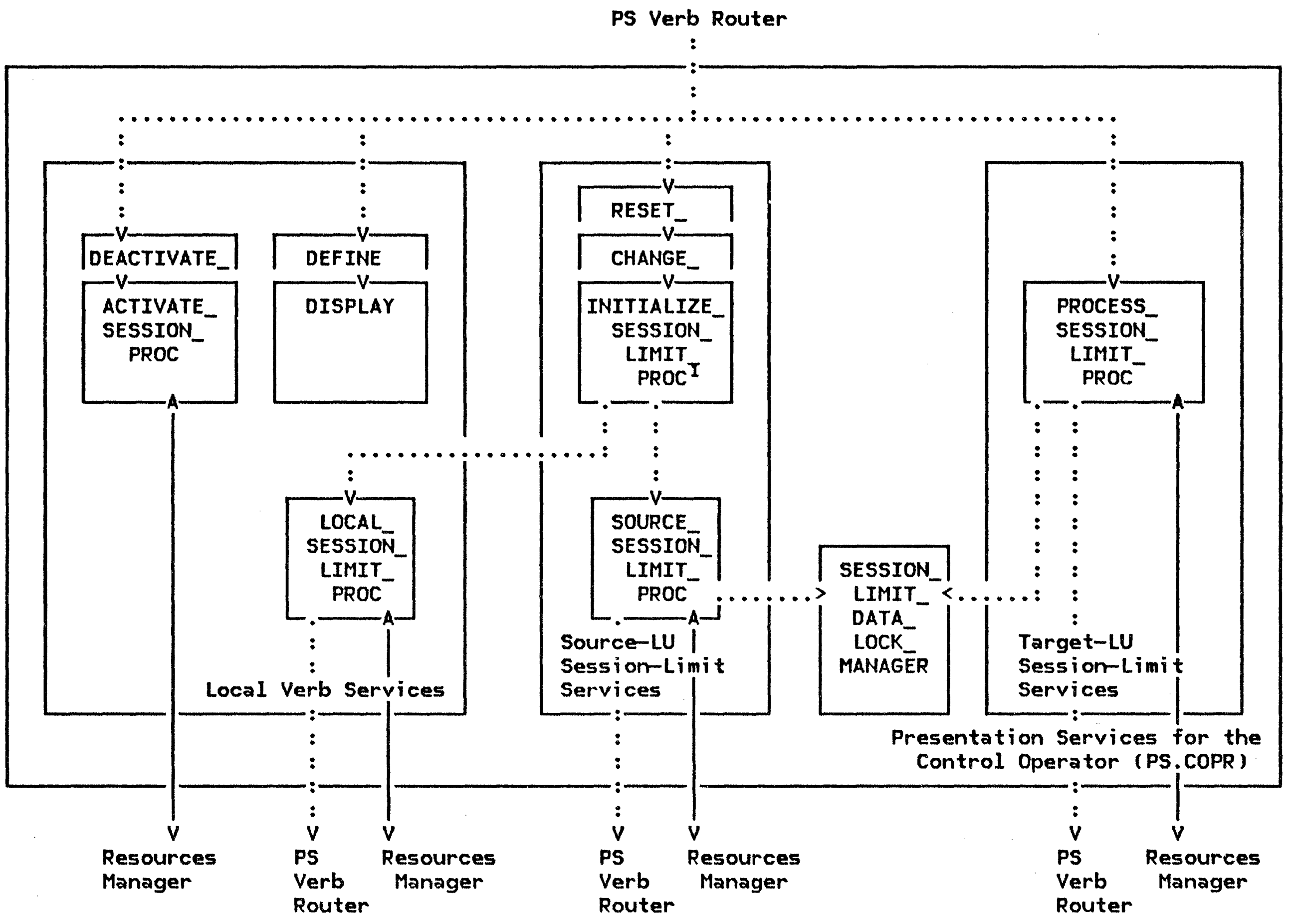
The CNOS service transaction program is that SNA-defined transaction program with transaction-program name (TPN) X'06F1'. It represents the control operator at the target LU. It is invoked by presentation services (PS.INITIALIZE) when the target LU receives the Attach FM header that resulted from the 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") to allow that program to examine the new session-limit parameters and display them for the operator.


PS Verb Router
DEACTIVATE_
ACTIVATE_ SESSION_ PROC
DEFINE
DISPLAY
RESET_
CHANGE_
INITIALIZE_ SESSION_ LIMIT_ PROC[1]
PROCESS_ SESSION_ LIMIT_ PROC
LOCAL_ SESSION_ LIMIT_ PROC
SOURCE_ SESSION_ LIMIT_ PROC
SESSION_ LIMIT_ DATA_ LOCK_ MANAGER
Local Verb Services
Source-LU Session-Limit Services
Target-LU Session-Limit Services
Presentation Services for the Control Operator (PS.COPR)
Resources Manager
PS Verb Router
Resources Manager
PS Verb Router
Resources Manager
PS Verb Router
Resources Manager

[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-parameter verbs and local session-control verbs.

LU-PARAMETER VERB PROCESSING

The LU-parameter 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 inter-process (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 mode name reserved for use 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 inter-process (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, 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, 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

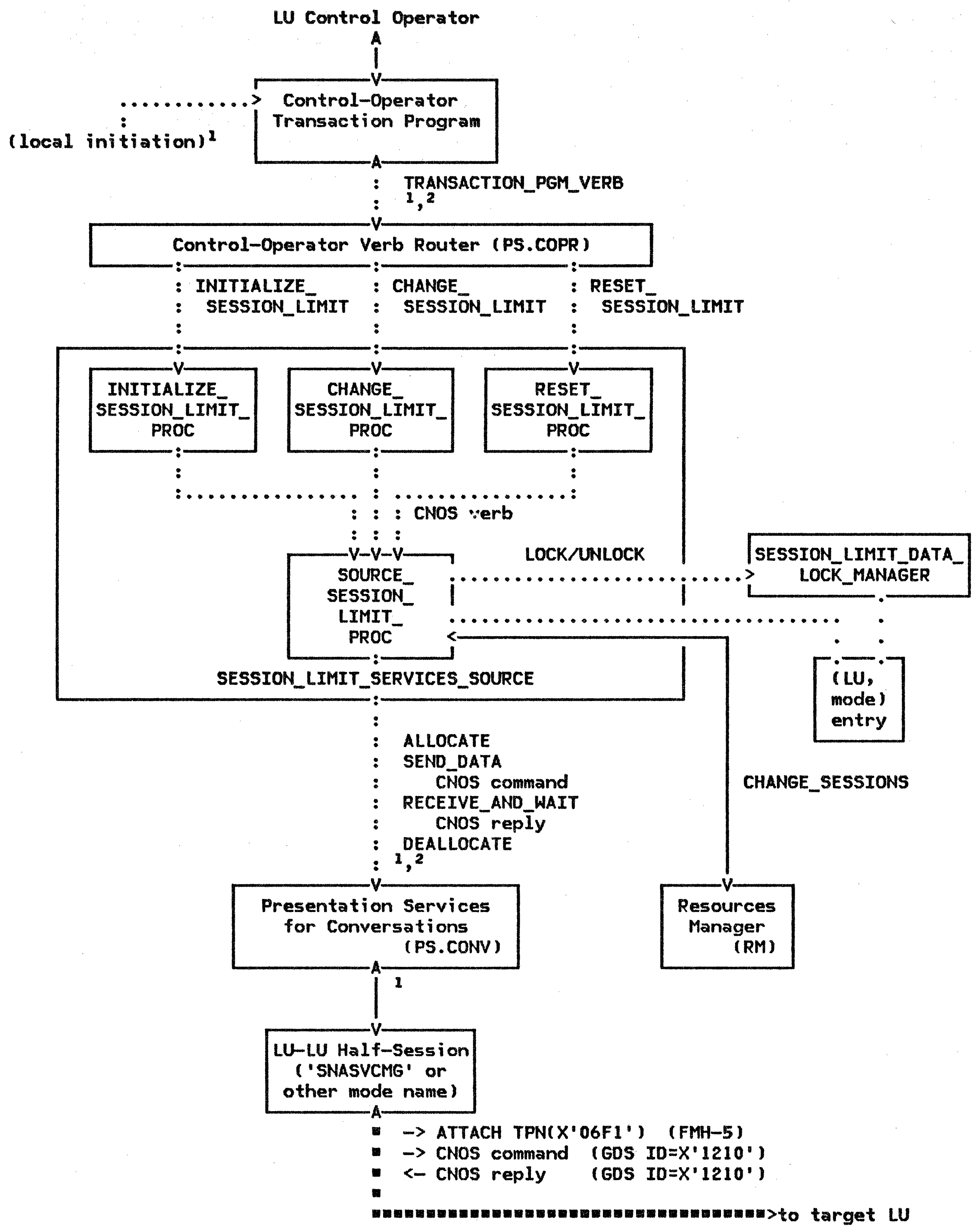
Source-LU session-limit services (SSLS) processes CNOS verbs issued at the source LU. It forms a part of the source-LU transaction-program process that includes the control-operator transaction program. It is invoked via the presentation services (PS) verb router and PS.COPR when the control-operator 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 on page 5.4-26).

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.


LU Control Operator
Control-Operator Transaction Program
(local initiation)[1]
TRANSACTION_PGM_VERB [1,2]
Control-Operator Verb Router (PS.COPR)
INITIALIZE_ SESSION_LIMIT
CHANGE_ SESSION_LIMIT
RESET_ SESSION_LIMIT
INITIALIZE_ SESSION_LIMIT_ PROC
CHANGE_ SESSION_LIMIT_ PROC
RESET_ SESSION_LIMIT_ PROC
CNOS verb
LOCK/UNLOCK
SOURCE_ SESSION_ LIMIT_ PROC
SESSION_LIMIT_DATA_ LOCK_MANAGER
(LU, mode) entry
SESSION_LIMIT_SERVICES_SOURCE
ALLOCATE
SEND_DATA
CNOS command
RECEIVE_AND_WAIT
CNOS reply
DEALLOCATE
1,2
CHANGE_SESSIONS
Presentation Services for Conversations (PS.CONV)
Resources Manager (RM)
1
LU-LU Half-Session ('SNASVCMG' or other mode name)
-> ATTACH TPN(X'06F1') (FMH-5)
-> CNOS command (GDS ID=X'1210')
<- CNOS reply (GDS ID=X'1210')
to target 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)
```

[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: Conversation-level security is not used for CNOS transactions.

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 "Appendix H. FM Header and LU Services Commands". 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 LU network services (see "Chapter 4. LU Network Services").

### 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 LU network services 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 on page 5.4-29).

- 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 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.

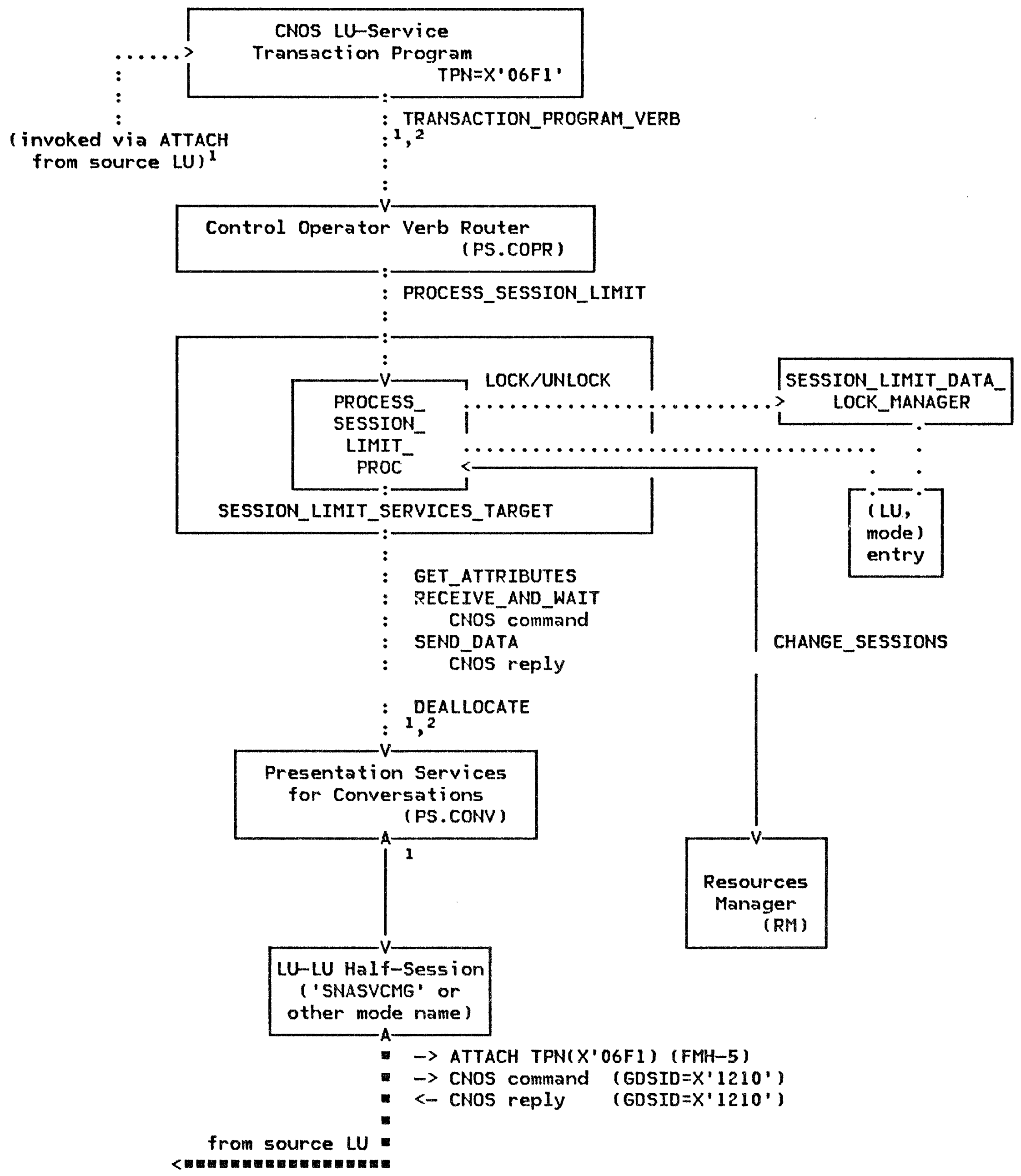
### CNOS Reply

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"--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 the all 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



CNOS LU-Service Transaction Program TPN=X'06F1'
(invoked via ATTACH from source LU)[1]
TRANSACTION_PROGRAM_VERB [1,2]
Control Operator Verb Router (PS.COPR)
PROCESS_SESSION_LIMIT
LOCK/UNLOCK
PROCESS_SESSION_LIMIT_PROC
SESSION_LIMIT_DATA_LOCK_MANAGER
(LU, mode) entry
SESSION_LIMIT_SERVICES_TARGET
GET_ATTRIBUTES
RECEIVE_AND_WAIT
CNOS command
SEND_DATA
CNOS reply
DEALLOCATE
1,2
CHANGE_SESSIONS
Presentation Services for Conversations (PS.CONV)
1
Resources Manager (RM)
LU-LU Half-Session ('SNASVCMG' or other mode name)
-> ATTACH TPN(X'06F1) (FMH-5)
-> CNOS command (GDSID=X'1210')
<- CNOS reply (GDSID=X'1210')
from source 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) |

[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

these parameters as modified by the target LU and returned in the CNOS reply record.

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. 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), the target LU may set its local drain state for any mode names to either YES or NO, regardless of the previous drain state.

- The target LU does not change DRAIN_SOURCE.

### <u>Errors</u>

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 "Appendix H. FM Header and LU Services Commands" 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.

### <u>Other Interactions</u>

Other TSLS interactions are similar to the corresponding interactions of SSLS.

## SESSION-LIMIT DATA LOCK MANAGER

### <u>Locking the (LU,mode) Entry</u>

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 |

```
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
      Call DEFINE_PROC with the verb parameters (page 5.4-38).
   When DISPLAY
      Call DISPLAY_PROC with the verb parameters (page 5.4-39).
```

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. If an ABEND condition occurs, it calls PS to abnormally terminate the transaction-program process. |
| 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 5.4-40 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| LUCB | page A-1 |

```
If this transaction program is not authorized to issue the CNOS verb then
   Call DEALLOCATION_CLEANUP_PROC (page 5.0-14)
    to abnormally terminate this instance of the transaction program
    (control is not returned).

Else
   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 5.4-40),
          with the verb parameters, to perform the CNOS action solely at the
          local LU.
```

## 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. If an ABEND condition occurs, it calls PS to abnormally terminate the transaction-program process. |
| 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 5.4-40 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| LUCB | page A-1 |

```
If this transaction program is not authorized to issue the CNOS verb then
   Call DEALLOCATION_CLEANUP_PROC (page 5.0-14)
    to abnormally terminate this instance of the transaction program
    (control is not returned).

Else
   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 DRAIN_TARGET and RESPONSIBLE parameters are not applicable. These attributes cannot be modified for the SNA-defined mode name.

```
   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 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 (local control-operator verb)
         Call LOCAL_SESSION_LIMIT_PROC (page 5.4-40),
          with the verb parameters, to perform the CNOS action solely at the local LU.
```

## 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. If an ABEND condition occurs, it calls PS to abnormally terminate the transaction-program process. |
| 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 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| LUCB | page A-1 |

```
If the control-operator transaction program, at the source LU,
 is not authorized to issue the CNOS verb then
  Call DEALLOCATION_CLEANUP_PROC (page 5.0-14)
   to abnormally terminate this instance of the transaction program
   (control is not returned).

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 <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u>.

```
  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 the
      Call SOURCE_SESSION_LIMIT_PROC (page 5.4-45),
       with the verb parameters, to begin the negotiation phase of the CNOS process.

    Else
      Call DEALLOCATION_CLEANUP_PROC (page 5.0-14)
       to abnormally terminate this instance of the transaction program
       (control is not returned).
```

## 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:

| | |
|---|---|
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| PS_PROCESS_DATA | page 5.0-20 |
| RM_ACTIVATE_SESSION | page A-27 |
| RM_SESSION_ACTIVATED | page A-33 |

```
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 result of parameter verification:
   When an ABEND condition is identified
      Call DEALLOCATION_CLEANUP_PROC (page 5.0-14) to abnormally
       terminate this instance of the transaction program (control is not returned).
   When a parameter error is identified
      Set the CNOS RETURN_CODE to PARAMETER_ERROR.
   When all parameters are correct
      Create an 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 ACTIVATE_SESSION request to RM.
      Receive SESSION_ACTIVATED reply from RM.
      Set CNOS RETURN_CODE according to the return code in the
       SESSION_ACTIVATED reply received from RM.
```

## 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 activate a session. |
| INPUT: | The CNOS 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:

| | |
|---|---|
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| PS_PROCESS_DATA | page 5.0-20 |
| RM_DEACTIVATE_SESSION | page A-27 |

```
If this transaction program is not authorized
 to issue the CNOS verb then
   Call DEALLOCATION_CLEANUP_PROC (page 5.0-14)
    to abnormally terminate this instance of the transaction program
    (control is not returned).

Else
   Set the CNOS RETURN_CODE to OK.
   Create a 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 DEACTIVATE_SESSION request to RM.
```

## DEFINE_PROC

| | |
|---|---|
| FUNCTION: | This procedure is called by PS_COPR, the control-operator-verb router, when a transaction program issues a DEFINE verb. It is used to initialize or modify system definition parameters in the LUCB, PARTNER_LU and MODE data structures. |
| INPUT: | The DEFINE verb parameters |
| OUTPUT: | The system definition parameters are defined with the specified values |
| NOTE: | This verb may be used to define any other system definition parameters 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 |

```
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
   Call DEALLOCATION_CLEANUP_PROC (page 5.0-14) to abnormal
    this instance of the transaction program (control is not returned).

Else

     ┌──────────────────────────────────────────────────────────────────┐
     │ The parameters specified are all valid fields of the LUCB, PARTNER_LU, │
     │ and MODE data structures.                                        │
     └──────────────────────────────────────────────────────────────────┘

   Assign values to the system definition parameters 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 a DISPLAY verb. It is used to display system definition parameters in the LUCB, PARTNER_LU and MODE data structures. |
| INPUT: | The DISPLAY verb parameters |
| OUTPUT: | The specified system definition parameters are displayed for the user. |
| NOTE: | This verb may be used to display any other system definition parameters 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 |

```
Verify that the verb parameters specified satisfy the
 parameter values for the DISPLAY verb in
 SNA Transaction Programmer's Reference Manual for LU Type 6.2.

If an ABEND condition is identified then
   Call DEALLOCATION_CLEANUP_PROC (page 5.0-14) to abnormal
    this instance of the transaction program (control is not returned).

Else

     The parameters specified are all valid fields of the LUCB, PARTNER_LU,
     and MODE data structures.

   Display the requested LUCB, PARTNER_LU, and MODE attributes as they
    are currently defined.
```

## 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:

| | |
|---|---|
| LOCAL_VERB_PARAMETER_CHECK | page 5.4-41 |
| SNASVCMG_VERB_PARAMETER_CHECK | page 5.4-42 |
| CHANGE_ACTION | page 5.4-43 |
| LUCB | page A-1 |

```
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 5.4-41),
    with the CNOS verb parameters, to verify the verb parameters.

Else
   Call SNASVCMG_VERB_PARAMETER_CHECK (page 5.4-42), 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:

| | |
|---|---|
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| LUCB | page A-1 |
| PARTNER_LU | page A-2 |
| MODE | page A-3 |

```
Verify that the verb parameters specified satisfy the single-session
 parameter values as described for this verb 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 based on result of parameter verification:
   When all parameters are correct
      Set the CNOS RETURN_CODE to OK--AS_SPECIFIED.
   When an ABEND condition is identified
      Call DEALLOCATION_CLEANUP_PROC (page 5.0-14)
       to abnormally terminate this instance of the transaction program
       (control is not returned).

      If the single-session  CNOS optional function set is  supported then a
      mode name  value of ALL constitutes  an ABEND condition when  the CNOS
      verb is RESET_SESSION_LIMIT.

   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 the session limit specified exceeds the session limit (in the LUCB) for the LU
    or would exceed the session limit for the LU in the process of adding
    a single session with the specified LU name and mode name (page 5.4-4) then
      Set the CNOS RETURN_CODE for this verb to REQUEST_EXCEEDS_MAX_ALLOWED.
```

## SNASVCMG_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:

| | |
|---|---|
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| LUCB | page A-1 |
| PARTNER_LU | page A-2 |
| 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 ABEND condition is identified
      Call DEALLOCATION_CLEANUP_PROC (page 5.0-14)
       to abnormally terminate this instance of the transaction program
       (control is returned).
   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 REQUEST_EXCEEDS_MAX_ALLOWED.
```

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 <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u> 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:

| | |
|---|---|
| CHANGE_SESSIONS | page A-26 |
| 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.TCB_ID to PS_PROCESS_DATA.TCB_ID to identify the transaction
          control block describing this instance of PS.
         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 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 SESSION_LIMIT, MIN_CONWINNERS_LIMIT and MIN_CONLOSERS_LIMIT to 0.
         If this LU is responsible for session deactivation 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; 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
          (Appendix H).

         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 Appendix H.

               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
                   fully-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.

         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:

| | |
|---|---|
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| LUCB | page A-1 |
| PARTNER_LU | page A-2 |
| MODE | page A-3 |

```
Verify that the verb parameters specified 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 an ABEND error condition is identified
      Call DEALLOCATION_CLEANUP_PROC (page 5.0-14)
       to abnormally terminate this instance of the transaction program
       (control is not returned).
   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 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 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:<br>• OK: conversation completed successfully<br>• SON: session outage occurred; retry for the same mode name might succeed<br>• NO_SESSION: no session is available for this mode name; retry for another mode name might succeed<br>• 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 fully-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:<br>• OK: conversation completed successfully<br>• SON: session outage occurred; retry for the same mode name might succeed<br>• NO_SESSION: no session is available for this mode name; retry for another mode name might succeed<br>• FAILED: conversation or transaction failure; retry is not likely to succeed<br><br>The SOURCE_CONVERSATION_CONTROL procedure will make a retry decision based on this information. |
| NOTE: | See <u>SNA Transaction Programmer's Reference Manual for LU Type 6.2</u> 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 |
| LUCB | page A-1 |
| PARTNER_LU | page A-2 |

```
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 DEALLOCATE 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 fully qualified LU names for
    this LU and the partner 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:<br>• OK: conversation completed successfully<br>• SON: session outage occurred; retry for the same mode name might succeed<br>• NO_SESSION: no session is available for this mode name; retry for another mode name might succeed<br>• FAILED: conversation or transaction failure; retry is not likely to succeed<br>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:<br>• OK: conversation completed successfully<br>• SON: session outage occurred; retry for the same mode name might succeed<br>• NO_SESSION: no session is available for this mode name; retry for another mode name might succeed<br>• FAILED: conversation or transaction failure; retry is not likely to succeed<br><br>This information will later be used by SOURCE_CONVERSATION_CONTROL to make a retry decision. |
| NOTE: | Checks are required unless designated optional. |

```
If the REQUEST_TO_SEND_RECEIVED parameter returned from the SEND_DATA verb is NO then

     Select, in order, based on the RETURN_CODE parameter from the SEND_DATA verb:
        When OK
           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.

  Else
     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:<br>• OK: conversation completed successfully<br>• SON: session outage occurred; retry for the same mode name might succeed<br>• NO_SESSION: no session is available for this mode name; retry for another mode name might succeed<br>• FAILED: conversation or transaction failure; retry is not likely to succeed<br><br>This information will later be used by SOURCE_CONVERSATION_CONTROL to make a retry decision. |
| NOTE: | Checks are required unless designated optional. |

```
If the REQUEST_TO_SEND_RECEIVED parameter from the RECEIVE_AND_WAIT verb is NO then

      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
               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.

         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.

   Else
      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_IMMEDIATE verb:<br>• OK: conversation completed successfully<br>• SON: session outage occurred; retry on the same mode name might succeed<br>• NO_SESSION: no session is available for this mode name; retry on another mode name might succeed<br>• FAILED: conversation or transaction failure; retry is not likely to succeed |

```
If the REQUEST_TO_SEND_RECEIVED parameter returned from the DEALLOCATE verb is NO then

      Select based on the RETURN_CODE value returned from the DEALLOCATE verb:
         When DEALLOCATE_NORMAL
            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.

   Else
      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.<br><br>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 fully-qualified LU names for the source and target LUs
       (returned from the GET_ATTRIBUTES verb 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. If an ABEND condition occurs, it calls PS to abnormally terminate the transaction-program process. |
| 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:

| | |
|---|---|
| CHECK_CNOS_COMMAND | page 5.4-62 |
| CHANGE_ACTION | page 5.4-43 |
| TARGET_COMMAND_CONVERSATION | page 5.4-59 |
| TARGET_REPLY_CONVERSATION | page 5.4-64 |
| SESSION_LIMIT_DATA_LOCK_MANAGER | page 5.4-66 |
| DEALLOCATION_CLEANUP_PROC | page 5.0-14 |
| 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
   Call DEALLOCATION_CLEANUP_PROC (page 5.0-14) to abnormally
    terminate this instance of the transaction program (control is not returned).

Else
   Call TARGET_COMMAND_CONVERSATION (page 5.4-59)
    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 Appendix H.

      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 Appendix H).
   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.
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 5.4-60 |
| RESULT_CHECK_RECEIVE_SEND | page 5.4-61 |
| RESULT_CHECK_SEND_REPLY | page 5.4-65 |

```
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
    "Appendix H. FM Header and LU Services Commands".

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 5.4-60).

   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. |

```
If the REQUEST_TO_SEND_RECEIVED parameter returned from the
 RECEIVE_AND_WAIT verb is NO then

   Select based on the RETURN_CODE parameter returned from
    RECEIVE_AND_WAIT:
      When OK
         If WHAT_RECEIVED = DATA_COMPLETE then
            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.

Else (REQUEST_TO_SEND_RECEIVED=YES--an 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. |

```
If the REQUEST_TO_SEND_RECEIVED parameter returned from the RECEIVE_AND_WAIT
 is NO then

   Select based on the RETURN_CODE parameter returned from the RECEIVE_AND_WAIT:
      When OK
         If WHAT_RECEIVED = SEND 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.

  Else
     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:

| | |
|---|---|
| LUCB | page A-1 |
| PARTNER_LU | page A-2 |
| 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.<br><br>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.

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)
      For MODE_NAME(ALL), only RESPONSIBLE may be negotiated.
      Negotiate the RESPONSIBLE parameter from TARGET to
       SOURCE.

If any of these parameters is negotiated then
   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:

| | |
|---|---|
| PARTNER_LU | page A-2 |
| 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 |

```
If the REQUEST_TO_SEND_RECEIVED parameter returned from the SEND_DATA
 verb is NO then

   Select based on the RETURN_CODE parameter returned from the SEND_DATA verb:
      When OK
         Issue a DEALLOCATE verb with TYPE=SYNC_LEVEL to deallocate
          the conversation normally.
      When RESOURCE_FAILURE_RETRY or DEALLOCATE_ABEND_PROG
         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
         Issue a DEALLOCATE verb with TYPE=ABEND_PROG to deallocate
          the conversation.

Else
   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.

# CHAPTER 6.0. HALF-SESSION

## GENERAL DESCRIPTION

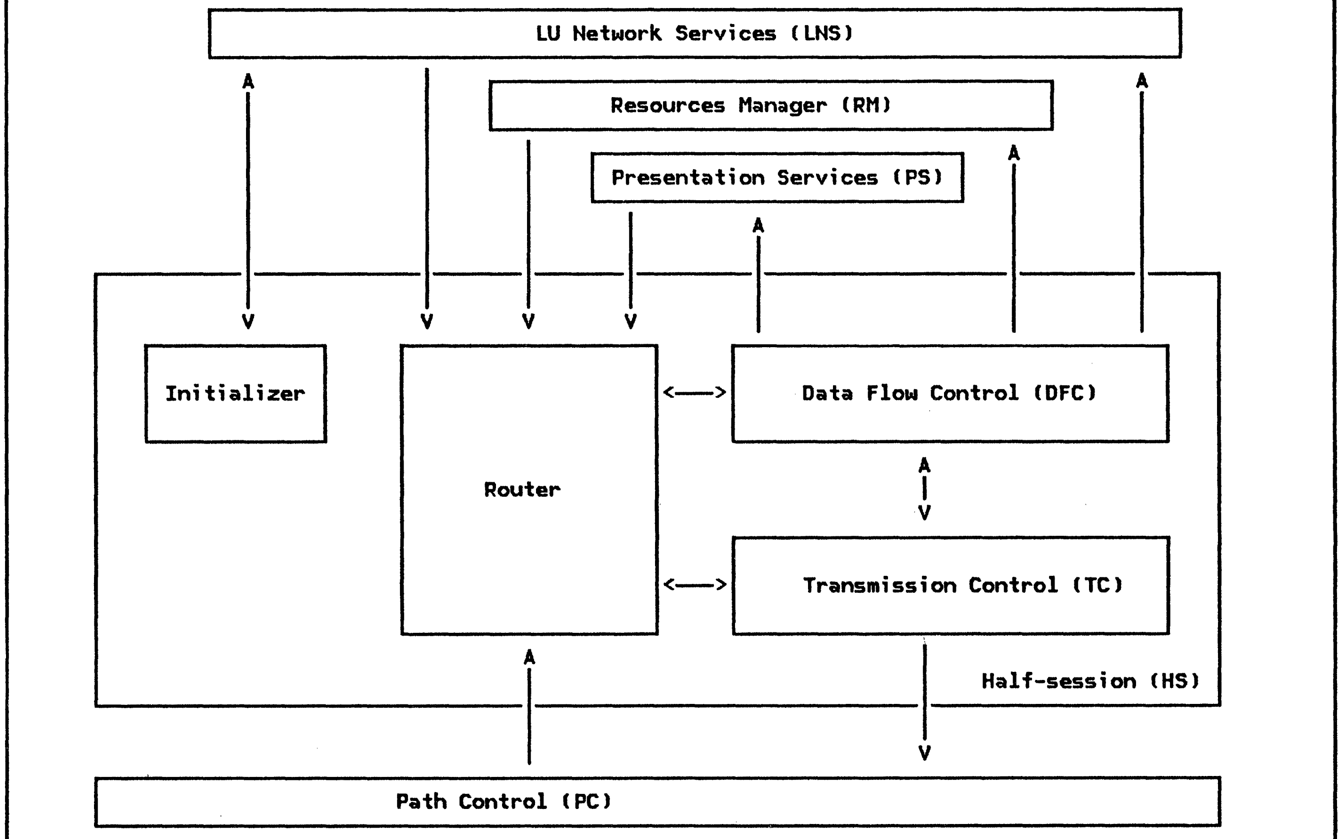


Figure 6.0-1. Overview of Half-Session

The half-session component (see Figure 6.0-1) resides in the LU and represents a session with another LU or with a control point (e.g., an SSCP). 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. Message units received from LU network services (LNS--see "Chapter 4. LU Network Services"), resources manager (RM--see "Chapter 3. LU Resources Manager"), and 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 BIUs and records produced from transaction program verbs and to control the flow of data between the half-session and PS, RM, and LNS. 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 LNS when a session-activation request (BIND or ACTLU) has been successfully processed. The half-session is destroyed by LNS when (1) a session-deactivation request (UNBIND or DACTLU) has been processed, (2) a session route outage has occurred, or (3) a control point session has been deactivated and requires a heirarchical reset of all related sessions (e.g., the PU-CP session has been deactivated).

The half-session, RM, PS, LNS, and PC are all separate processes. Message units are sent to HS by RM, PS, LNS, 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, LNS, and PC.

## PROTOCOL BOUNDARIES BETWEEN HS AND OTHER COMPONENTS

Message units that flow from HS to RM:
- ATTACH_HEADER
- BID
- BID_RSP
- FREE_SESSION
- BIS_RQ
- BIS_REPLY
- RTR_RQ
- RTR_RSP

Message units that flow from RM to HS:
- BID_WITHOUT_ATTACH
- BID_WITH_ATTACH
- BID_RSP
- YIELD_SESSION
- BIS_RQ
- BIS_REPLY
- RTR_RQ
- RTR_RSP
- HS_PS_CONNECTED

Message units that flow from HS to PS:
- RECEIVE_DATA
- CONFIRMED
- RECEIVE_ERROR
- REQUEST_TO_SEND
- RSP_TO_REQUEST_TO_SEND

Message units that flow from PS to HS:
- SEND_DATA_RECORD
- CONFIRMED
- SEND_ERROR
- REQUEST_TO_SEND

Message units that flow from HS to LNS:
- INIT_HS_RSP
- Network services BIUs (carried in HS_RCV_RECORD)
- ABORT_HS

Message units that flow from LNS to HS:
- INIT_HS
- Network services BIUs (carried in HS_SEND_RECORD)

Message units that flow from HS to PC:
- PIU information containing a request or response BIU

Message units that flow from PC to HS:
- PIU information containing a request or response BIU

## HS

| | |
|---|---|
| FUNCTION: | This procedure causes the half-session to be initialized and invokes the appropriate router according to the type of half-session (LU-CP or LU-LU). |
| INPUT: | At creation time, HS_ID (half-session identifier) and LU_ID (LU identifier); at run time, INIT_HS received from LNS |
| OUTPUT: | INIT_HS_RSP sent to LNS, HS_ID and LU_ID recorded for other procedures in the half-session. The following are recorded for use by other procedures in the half-session: LOCAL.SENSE_CODE is initialized to 0; the PC_ID of the path control that the half-session uses; the half-session role (PRI or SEC); and the FM and TS profile types. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| TC.INITIALIZE | page 6.2-8 |
| DFC_INITIALIZE | page 6.1-18 |
| PROCESS_LU_LU_SESSION | page 6.0-4 |
| PROCESS_CP_LU_SESSION | page 6.0-5 |
| INIT_HS | page A-16 |
| INIT_HS_RSP | page A-11 |
| LOCAL | page 6.0-6 |

Before the half-session can begin processing, it must be initialized. Therefore, the first thing HS does after creation is to receive an initialization record (INIT_HS) from LNS. The initialization record specifies the rules and parameters that this session will use (this information comes from BIND or ACTLU).

```
Record the HS_ID and LU_ID to make the information available to all
 half-session procedures.

Set LOCAL.SENSE_CODE to 0 (the no error state).
From INIT_HS.TYPE, record an indication that this half-session is
 primary (PRI) or secondary (SEC).
Depending on whether the INIT_HS.DATA_TYPE = ACTLU_IMAGE or BIND_IMAGE,
 record the FM profile and TS profile types from the ACTLU or BIND image.

Initialize the half-session by calling
 TC.INITIALIZE(INIT_HS record) (page 6.2-8) and
 DFC_INITIALIZE(INIT_HS record) (page 6.1-18), passing them
 the INIT_HS record.

If TC and DFC initialization is successful then
   Send a positive INIT_HS_RSP to LNS (use 0 for the SENSE_CODE, POS for TYPE,
    and HS_ID to identify this HS).
   If FM profile is 0 or 6 (CP-LU session) then
      Call PROCESS_CP_LU_SESSION (page 6.0-5).
   Else (FM profile is 19, for an LU-LU session)
      Call PROCESS_LU_LU_SESSION (page 6.0-4).

Else (initialization unsuccessful--LOCAL.SENSE_CODE indicates the type of error)
   Send a negative INIT_HS_RSP to LNS for this LU.  Use
    LOCAL.SENSE_CODE as the INIT_HS_RSP sense code, NEG for TYPE,
    and HS_ID to identify this HS.

Wait to be destroyed.
```

## PROCESS_LU_LU_SESSION

| | |
|---|---|
| FUNCTION: | Does processing for LU-LU 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 LU-LU 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 LNS. This causes LNS to send an UNBIND(protocol error) for this LU-LU session. |
| INPUT: | Message units received from PS, RM, and PC; LOCAL.SENSE_CODE |
| OUTPUT: | ABORT_HS sent to LNS if error |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| DFC_SEND_FROM_RM | page 6.1-20 |
| DFC_SEND_FROM_PS | page 6.1-19 |
| TRY_TO_RCV_SIGNAL | page 6.1-22 |
| TC.RCV | page 6.2-15 |
| TC.TRY_TO_SEND_IPR | page 6.2-19 |
| FSM_BSM_FMP19 | page 6.1-43 |
| FSM_CHAIN_SEND_FMP19 | page 6.1-46 |
| ABORT_HS | page A-11 |
| LOCAL | page 6.0-6 |

```
Do while LOCAL.SENSE_CODE = 0. (Do while no errors.)
   Select based on the source of the record:
      When the record is from PS
         Call DFC_SEND_FROM_PS (page 6.1-19) to route the record to DFC.
          (When the session is between brackets
          [state of FSM_BSM_FMP19 = BETB] or PS is sending data and the
          half-session is expecting either a response or a reply
          [state of FSM_CHAIN_SEND_FMP19 = PEND_RSP or PEND_RCV_REPLY],
          processing of request records from PS is deferred until
          this condition is resolved.)

      When the record is from RM
         Call DFC_SEND_FROM_RM (page 6.1-20) to route the record to DFC.

      When the record is from PC
         Call TC.RCV (page 6.2-15) to route the record to TC.

   The input to those procedures is the received record.
```

```
    If LOCAL.SENSE_CODE ≠ 0 (error found) then
       Send an ABORT_HS record to LNS.  The ABORT_HS.SENSE_CODE comes from
        LOCAL.SENSE_CODE; ABORT_HS.HS_ID is the HS_ID saved during HS initialization.
        (LNS sends an UNBIND.)

    Else (no error found--continue processing)
       Call TRY_TO_RCV_SIGNAL (page 6.1-22) 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 so that it can check
        the current half-session state and process a SIGNAL request if necessary.

       Call TC.TRY_TO_SEND_IPR (page 6.2-19) to
        see if an ISOLATED PACING RESPONSE (IPR) may be sent, depending
        on the pacing state of the half-session.  The TC.TRY_TO_SEND_IPR
        procedure is called so that it can check the current half-session
        pacing state and send an IPR if necessary.
        (The formal description sends IPRs even if a response
        will be the next RU sent.  Implementations may optimize flows
        by setting the Pacing indicator to PAC on the response, rather than
        sending an IPR followed by a response that has the pacing
        indicator set to ¬PAC.)
```

PROCESS_CP_LU_SESSION

| | |
|---|---|
| FUNCTION: | Does processing for CP-LU half-session (FM profiles 0 and 6). Message units received from LNS are routed to DFC. Message units received from PC are routed to TC. The CP-LU half-session continues to operate until the half-session process is destroyed (e.g., because of a DACTLU request).<br><br>If an error condition occurs, LOCAL.SENSE_CODE is set with the sense data indicating the kind of error, the half-session component detecting the error sends a -RSP or logs the error. The CP-LU half-session continues to operate. |
| INPUT: | Message units received from LNS and PC; FM profile type |
| OUTPUT: | LOCAL.SENSE_CODE |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| DFC_SEND_FROM_LNS | page 6.1-22 |
| TC.RCV | page 6.2-15 |
| FSM_IMMEDIATE_RQ_MODE_SEND | page 6.1-48 |
| LOCAL | page 6.0-6 |

```
  Do until HS process is destroyed.
     Set LOCAL.SENSE_CODE to 0.
     Select based on the source of the record:
        When the record is from LNS
           Call DFC_SEND_FROM_LNS (page 6.1-22) to route the record to DFC.
            (When the session is using immediate request mode [FM profile = 0]
            and a request is already outstanding [FSM_IMMEDIATE_RQ_MODE_SEND
            (page 6.1-48) is in the PEND_RSP state], processing of
            request records from LNS is deferred until a response to the outstanding
            request is received.)

         When the record is from PC
           Call TC.RCV (page 6.2-15) to route the record to TC.
```

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
      SENSE_CODE

   DFC:  fields used only by DFC
      LU_LU:  fields used for LU-LU sessions (FM profile 19)
         SQN_SEND_CNT:  contains SNF (see page 6.0-6)
         PHS_BB_REGISTER:  contains SNF (see page 6.0-6)
         SHS_BB_REGISTER:  contains SNF (see page 6.0-6)
         CURRENT_BRACKET_SQN:  contains SNF (see page 6.0-6)
         SEND_ERROR_RSP_STATE:  possible values:  RESET, NEG_OWED (negative
          response owed)
         SIG_RECEIVED: possible values: YES, NO
         SEND_BUFFER:  buffer area for collecting transaction program data until
          the maximum RU size is reached or DFC is instructed to send the data.
   TC:  fields used only by TC
      TCCB
         Q_PAC:  the outbound pacing queue for send pacing.  Holds BIUs
          that were not sent earlier because the send pacing count
          was 0.  They are now waiting for a pacing response to
          let the next pacing group (window) be sent.
         SEND_PACING_COUNT:  the number of requests that the local
          half-session can send before having to wait for a
          pacing response (varies between 2n-1 and 0, where n is the
          send window size)
         RCV_PACING_COUNT:  the number of requests that the local
          half-session can yet receive from the partner half-session
          in the currently allowed windows (varies between 2n-1 and 0,
          where n is the receive window size).
          If a request is received when this count is 0,
          the request is refused with a negative response.
         SQN_RCV_CNT:  the last received sequence number on the normal flow.
          Wraps to zero after 65535.
```

SNF

Defines sequence number field.

```
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.
```

# 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 data flow control component; network control and session control RUs do not. An LU may have a session with another LU or a control point (CP). The protocol rules (e.g., FM profile) to be used on the session are established when the session is activated and differ based on the type of session. LU-LU sessions use FM profile 19; CP-LU sessions use FM profile 0 or 6. Data flow control protocols differ significantly based on the FM profile. Protocols associated with FM profile 19 contain many more functions and capabilities then those associated with FM profile 0 or 6. The following describes the data flow control protocols for LU-LU and CP-LU sessions.

## DFC FOR LU-LU HALF-SESSIONS

### OVERVIEW OF DFC FUNCTIONS

The following functions are done by DFC for LU-LU sessions:

- 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.

### DFC STRUCTURE

The DFC structure is shown in Figure 6.1-1 on page 6.1-2.

#### Initialization

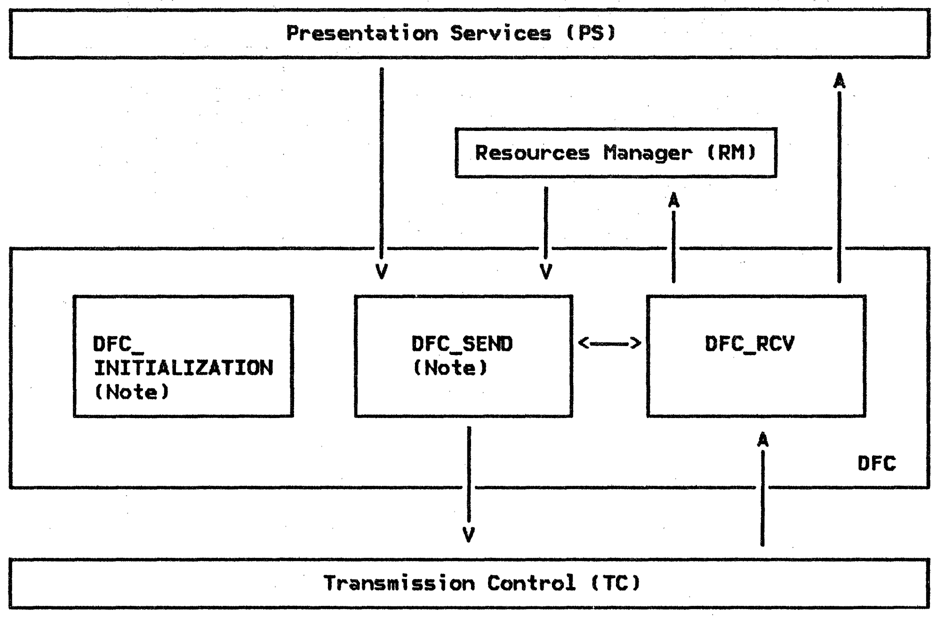
The DFC initialization procedure is called by the half-session router (see "Chapter 6.0. Half-Session") at the activation of each session. It initializes FSMs and other protocol related parameters to be used during the session.

#### Send

The DFC send procedures receive records from presentation services (PS) and from the resources manager (RM). They also receive records from the DFC receive procedure. The send procedures process the records and sends them on to transmission control (TC). The send processing consists of creating corresponding BIU records and updating the states of DFC send FSMs.

#### Receive

The DFC receive procedure (DFC_RCV) receives BIU records from TC, processes them, and sends them on to PS or RM. It also generates BIU records that it sends to the DFC send procedures. DFC_RCV optionally checks the BIU records for receive error conditions. These are conditions that occur only when the other half-session has violated the architec-


Presentation Services (PS)
Resources Manager (RM)
DFC_ INITIALIZATION (Note)
DFC_SEND (Note)
DFC_RCV
DFC
Transmission Control (TC)

Note: Called by half-session router ("Chapter 6.0. Half-Session")

Figure 6.1-1. Overview of DFC for LU-LU Half-Sessions

ture. When DFC_RCV detects an error condition, it sets the sense code (in global process data) 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 processing consists mainly of updating the states of DFC receive FSMs and creating corresponding records to be sent on to PS or RM.

Termination

DFC and other half-session components stay active until a deactivation request (UNBIND or DACTLU) flows. On LU-LU sessions, 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 BIU records suitable for its processing; similarly, it maps BIU records into commands and replies suitable for processing by RM and PS. The records DFC sends to, and receives from, TC are BIU records that represent RU chains.

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, and TC are defined by the individual DFC procedures.

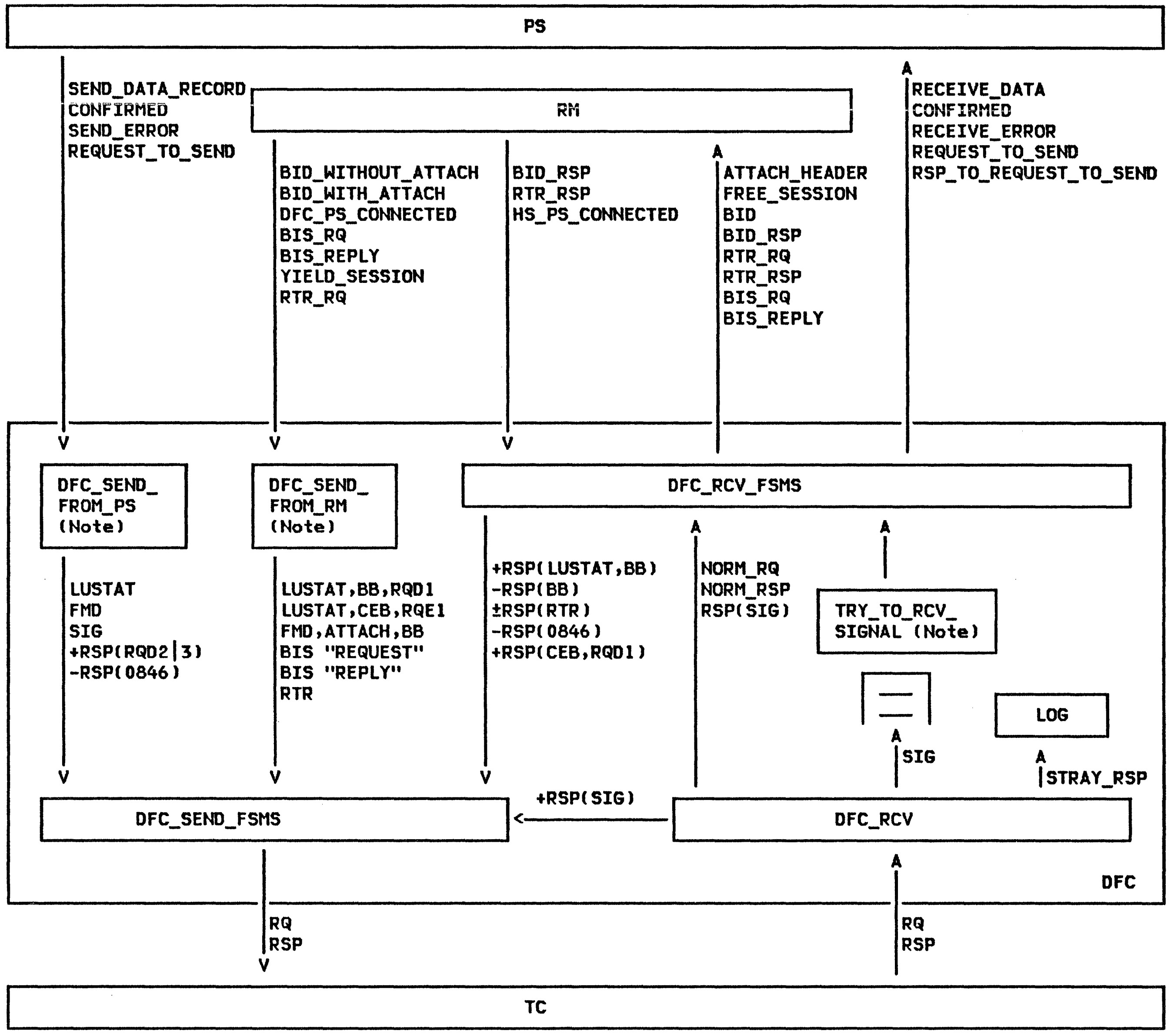
Throughout this chapter, references to request units (requests) and response units (responses) pertain to the BIU records that represent the requests and responses. References to the sending or receiving of requests and responses pertain to the protocol boundary with TC, unless stated otherwise.

## FUNCTION MANAGEMENT PROFILE 19

FM profiles are used to convey information about the protocols used on a session. FM profile 19 is used for LU-LU half-sessions. The DFC requests for this profile are BIS, LUSTAT, RTR, and SIG. These requests are used to control the flow of data between the half-sessions and are described in "DFC Request and Response Descriptions" on page 6.1-14.

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: FM headers (only FMH-5 (Attach) and FMH-7 (Error)) are used.



Note: Called by half-session router ("Chapter 6.0. Half-Session")

Figure 6.1-2. Detailed Structure and Protocol Boundaries of DFC for LU-LU Half-Sessions

- 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 negotiated at BIND time. 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, may be found in the following pages.

## Conditional End Bracket (CEB)

The Conditional End Bracket (CEB) is used to indicate bracket termination. It is allowed only on an RH with EC. The bracket is terminated in all cases except that 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 6.1-8 for more details on bracket termination.)

## FM Header Usage

The Format indicator (FI) in the RH is used to indicate the presence of an FM header as the first byte of FM data following this RH. The FM headers that are indicated by the FI are either FMH-5(Attach) or FMH-7(Error), which are described in "Appendix H. FM Header and LU Services Commands".

The FMH-5(Attach) may be carried only in an RU with the Begin Chain indicator (BCI) set to BC.

The FMH-7(Error) 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.

## 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" on page 6.1-5).

## 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 LUSTAT(0006) (RQE1,CEB)

LU-LU sessions are activated in the in-brackets (INB) state. If, for some reason, RM decides a newly activated session is not needed, it sends YIELD_SESSION to DFC. This results in an (RQE1,CEB) LUSTAT(0006) being sent to terminate the unused bracket.

## Usage of SIGNAL(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 6.1-10). SIGNAL always uses the code Request to Send (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 other 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 6.1-5 illustrates an example of the use of sequence numbers. In

```
Sending   Receiving
Sequence  Sequence
Number    Number    LUa                                          LUb
--------------------------------------------------------------------
(Note 1)            |-------------BIND---------------------------->|
          (Note 2)  |<---------------------RSP(BIND)---------------|
(Note 1)            |-------------CRV----------------------------->|
          (Note 2)  |<---------------------RSP(CRV)----------------|
   1      (Note 3)  |-------------Normal-flow RU ----------------->|
          (Note 4)  |<--------------RSP(Normal-flow RU)------------|
   2                |-------------Normal-flow RU ----------------->|
          (Note 4)  |<--------------RSP(Normal-flow RU)------------|
   3                |-------------Normal-flow RU ----------------->|
          (Note 4)  |<--------------RSP(Normal-flow RU)------------|
(Note 4)            |<------------SIGNAL---------------------------|
          (Note 4)  |----------------------RSP(SIGNAL)------------>|
   4                |-------------Normal-flow RU ----------------->|
          (Note 4)  |<--------------RSP(Normal-flow RU)------------|
(Note 1)            |-------------UNBIND-------------------------->|
          (Note 2)  |<--------------RSP(UNBIND)--------------------|
(Note 1)            |-------------BIND---------------------------->|
          (Note 2)  |<---------------------RSP(BIND)---------------|
   1                |-------------Normal-flow RU ----------------->|
          (Note 4)  |<--------------RSP(Normal-flow RU)------------|
   2                |-------------Normal-flow RU ----------------->|
          (Note 4)  |<--------------RSP(Normal-flow RU)------------|
   3                |-------------Normal-flow RU ----------------->|
          (Note 4)  |<--------------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. This is done for efficiency. The bracket sequence number is 0, the sequence number of the first 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.

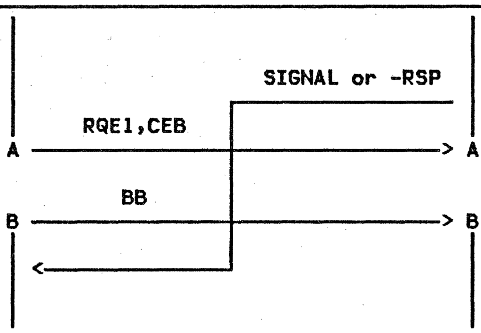
   - The high-order bit is 0 if the bracket was started by the secondary half-session, or 1 if the bracket was started by the primary half-session.

Figure 6.1-3. Use of Sequence Numbers

this figure, some session control RUs (BIND, UNBIND, and CRV) are also illustrated.

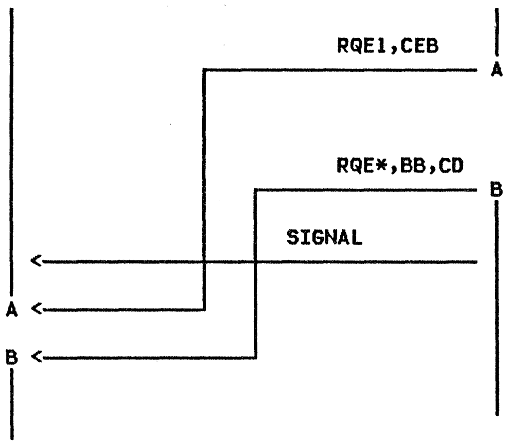
## Stray SIGNALs and Responses

When a request is sent (RQE1,CEB) or (RQD*,CEB) a stray SIGNAL or response can occur. This is a SIGNAL or response that is received outside the bracket it is intended for, and which could be disruptive if not eliminated or not recognized as a stray. SIGNALs received outside the intended (current) bracket may be "early" or "late". "Early" SIGNALs are those received in a bracket that was started prior to the current bracket. "Late" SIGNALs are those received in a bracket that was started after the current bracket. Responses received outside the current bracket are always "late". Examples are shown in the following figures.



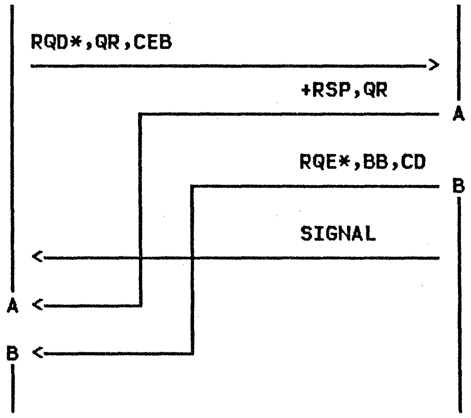
Bracket B gets the SIGNAL intended for A.

Figure 6.1-4. Case 1: "Late" SIGNAL or Response



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<br>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<br>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 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 only gets 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 6.1-6 and Figure 6.1-6 on page 6.1-6), 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--path errors and invalid sense code values are detected and a conversation failure is reported to PS and RM. An UNBIND(X'FE....') with sense code 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). (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--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(0846).

After the EC request is received, PS can send the FMH-7(Error); the FMH-7 includes sense data for PS's use—it 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 "Appendix D. RH Formats" 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 may not be 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.

There are only two expedited RUs used (SIG and CRV) and both use the immediate request mode. The two RUs flow at different times (when in use, the CRV exchange must be completed 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 <u>bracket</u> 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.

The bracket rules regulate the initiation and termination of a bracket.

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 <u>first speaker</u> and the other as <u>bidder</u>. 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 6.1-10 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 6.1-10 for definition of reply) or a positive or negative response (other than 0813, 0814, or 088B) or refuses the attempt via negative response (0813, 0814, or 088B).

A negative response with sense code 0813, 0814, or 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 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 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 0813 (Bracket Bid Reject--No RTR Forthcoming) or 088B (BB Not Accepted--BIS Reply Requested). In the 0813 case, the bidder will send BB again, if it still wants to begin a bracket. In the 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.

The following rules apply to the bracket indicators:

- BB may be indicated only on the first (or only) request of a chain.
- CEB may be indicated only on the last (or only) request of a 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:

- <u>Bracket Termination Rule</u>: Bracket termination is controlled by the form of response requested (definite or exception) for the chain containing CEB. If the chain requests a definite response, the bracket is not terminated until a positive response is processed. A negative response to the last request (marked definite response) causes the bracket to be continued. If the chain requests exception response, the bracket is terminated unconditionally when the request containing CEB is processed. A negative response to an (RQE,CEB) request will not be found in the receive correlation entity, and therefore will be logged and discarded.

No more than one BB can be outstanding from a half-session.

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 (0813, 0814, or 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 6.1-8 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 on requests. The setting of the QRI bit is the same for all RUs in a chain. See "Appendix D. RH 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 RECEIVE_DATA record (sent from HS to PS) are mapped to and from the RH portion of a BIU containing a request. The SEND_DATA_RECORD is used by PS to send data in accordance with the verbs issued by a transaction program. This record (defined using transaction program verb terminology) is mapped into a request BIU by DFC before being sent. The RECEIVE_DATA record is used to inform PS about data received on the half-session. This record (defined using transaction program verb terminology) is mapped from a received BIU containing a request. Figure 6.1-7 on page 6.1-11 summarizes the SEND_DATA_RECORD to RH mapping and Figure 6.1-8 on page 6.1-11 summarizes the RH to RECEIVE_DATA record mapping.

| Parameters in SEND_DATA_RECORD | Request RH indicators |
|---|---|
| ALLOCATE=YES (see Note 1)<br>FMH=YES (see Note 1) | BB<br>FMH |
| NOT_END_OF_DATA<br>FLUSH | ¬EC,RQE1<br>¬EC,RQE1 |
| CONFIRM<br>PREPARE_TO_RECEIVE_CONFIRM_SHORT<br>PREPARE_TO_RECEIVE_CONFIRM_LONG | EC,RQD3<br>EC,CD,RQD3<br>EC,CD,RQE3 |
| PREPARE_TO_RECEIVE_FLUSH<br>DEALLOCATE_CONFIRM<br>DEALLOCATE_FLUSH with DEALLOCATE_ABEND_* FM header (see Note 3)<br>DEALLOCATE_FLUSH without DEALLOCATE_ABEND_* FM header (see Note 3) | EC,CD,RQE1<br>EC,CEB,RQD3<br>EC,CEB, RQD1<br><br>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. SEND_DATA_RECORD to Request RH Mapping

| Request RH indicators | Parameters set in RECEIVE_DATA Record |
|---|---|
| FMH | FMH=YES (see Note 1) |
| ¬EC | NOT_END_OF_DATA |
| EC,RQD2\|3<br>EC,CD,RQ*2\|3<br>EC,CD,RQE1 | CONFIRM<br>PREPARE_TO_RECEIVE_CONFIRM<br>PREPARE_TO_RECEIVE_FLUSH |
| EC,CEB,RQD2\|3<br>EC,CEB,RQE1 or RQD1 | DEALLOCATE_CONFIRM<br>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 RECEIVE_DATA record).

2. Other RH indicators (e.g., QRI) have no effect on the RECEIVE_DATA record parameter settings.

Figure 6.1-8. Request RH to RECEIVE_DATA Record Mapping

## 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 "Appendix E. Request-Response Unit (RU) Formats". Figure 6.1-9 on page 6.1-12 and Figure 6.1-10 on page 6.1-13 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 "Appendix D. RH Formats" for complete RH description.

3. 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. The TH formats are not described in this volume.

Figure 6.1-9. DFC Request Formats

| DFC Response-----> Header Indicators | | | BIS | RTR | LUSTAT | 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 "Appendix D. RH 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. The TH formats are not described in this volume.

Figure 6.1-10. DFC Response Formats

## DFC REQUEST AND RESPONSE DESCRIPTIONS

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 it's principle FSMs are described in more detail 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 encodings 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 transfer 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 it is now allowed to initiate a bracket. An RTR request is sent only by the first speaker (see "Bracket Protocols" on page 6.1-8). The use of RTR and it's principal FSMs are described in more detail in "Chapter 3. LU Resources Manager".

## SIG (SIGNAL)

Flow: Primary to secondary and secondary to primary (Expedited)

Principal FSM:
None

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.

The overview, structure, and protocol boundaries for DFC for CP-LU half-sessions is shown in Figure 6.1-11 on page 6.1-17.

The CP-LU session uses FM profile 0 or 6. Immediate request and immediate response modes are used for FM profile 0. Delayed request and delayed response modes are used for FM profile 6. Chain form-of-response-requested indications for FM profile 0 are restricted to RQD, while FM profile 6 allows any (RQD, RQE, or RQN). FM profiles 0 and 6 share the following properties:

- Only single-RU chains are allowed.
- Compression is not used.
- Brackets are not used.
- FM headers are not used.
- Alternate code is not allowed.
- Send/Receive mode is full duplex (FDX).

DFC is initialized at half-session activation time to the FM profile being used.

## OVERVIEW OF DFC FUNCTIONS

The functions of DFC for CP-LU half-sessions are:

- Enforce correct request and response formats (e.g., RH parameter settings).

- Enforce immediate request and immediate response mode for sending and receiving normal-flow data. No enforcement is necessary for the delayed request or delayed response mode.

### Request/Response Formatting

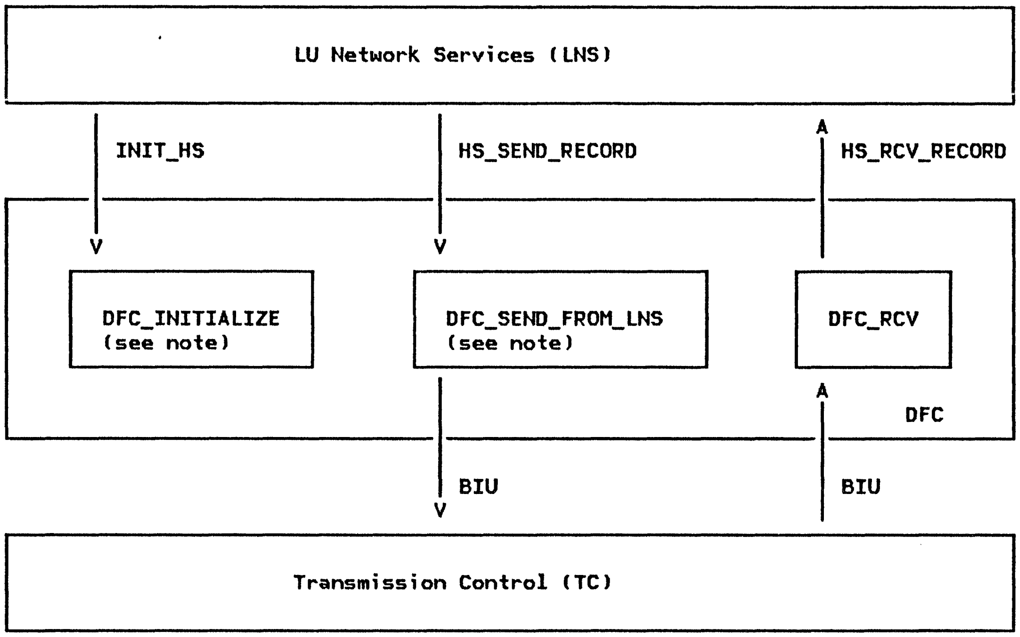
DFC optionally checks that received requests and responses are formatted correctly. For example, all RHs must have BCI=BC and ECI=EC. Formats may vary depending upon the FM profile used on the session. For instance, FM profile 0 allows chains asking only for definite response (RQD).

### Immediate Request and Immediate Response Mode Enforcement

DFC optionally checks that received requests do not violate immediate request mode protocols for sessions using FM profile 0. Once a request asking for a definite response has been received, it must be responded to before another request is received. Immediate response mode is also enforced for those sessions using FM profile 0. Any response received must be to an outstanding RQD request.

## ERROR PROCESSING

If a format error or immediate request or immediate response mode violation occurs, a negative response is sent if possible; otherwise, the error is logged. The half-session then continues to run until it is destroyed (e.g., via DACTLU).



LU Network Services (LNS)
INIT_HS
HS_SEND_RECORD
HS_RCV_RECORD
DFC_INITIALIZE (see note)
DFC_SEND_FROM_LNS (see note)
DFC_RCV
DFC
BIU
BIU
Transmission Control (TC)

<u>Note:</u> Called by half-session router ("Chapter 6.0. Half-Session")

Figure 6.1-11. Overview, Structure, and Protocol Boundaries of DFC for CP-LU Half-Sessions

DFC_INITIALIZE

| | |
|---|---|
| FUNCTION: | Initialize 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 record containing either ACTLU or BIND image; the following information is made available to DFC by the half-session router: FM profile type, indication that half-session is primary or secondary, LU_ID that identifies the LNS and RM associated with this half-session, HS_ID used to indicate to PS, RM, and LNS the half-session associated with a particular request. |
| OUTPUT: | SUCCESSFUL return code and half-session initialized |
| NOTE: | 1) When a half-session is activated it comes up in-brackets. The first BIU sent on the session uses a value of X'0001' in the TH sequence number field and does not carry BB. The BB, in effect, was carried on the session activation request (BIND). Therefore, the current bracket sequence number (LOCAL.CURRENT_BRACKET_SQN) associated with the first bracket on a session is initialized to 0.<br><br>2) The TS and FM profile type are checked for validity prior to calling this procedure. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_BSM_FMP19 | page 6.1-43 |
| FSM_RCV_PURGE_FMP19 | page 6.1-50 |
| FSM_QRI_CHAIN_RCV_FMP19 | page 6.1-49 |
| FSM_CHAIN_RCV_FMP19 | page 6.1-44 |
| FSM_CHAIN_SEND_FMP19 | page 6.1-46 |
| FSM_IMMEDIATE_RQ_MODE_SEND | page 6.1-48 |
| FSM_IMMEDIATE_RQ_MODE_RCV | page 6.1-48 |
| LOCAL | page 6.0-6 |
| INIT_HS | page A-16 |

```
Set all of the FSMs in this chapter to the reset state (state number 1).
If the INIT_HS record is for an LU-LU session (contains a BIND image) then
   Record information from BIND image in the INIT_HS record that will be used by DFC
    throughout the life of this session:
     • First speaker or bidder (contention winner or loser)
     • Maximum send RU size
     • Alternate code set allowed

   Set LOCAL.SQN_SEND_CNT to 0.
   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.CURRENT_BRACKET_SQN.BRACKET_STARTED_BY to PRI.
   Set LOCAL.CURRENT_BRACKET_SQN.NUMBER to 0. (See Note 1)

   Set LOCAL.SEND_ERROR_RSP_STATE to RESET.
   Set LOCAL.SIG_RECEIVED to NO.
```

DFC_SEND_FROM_PS

| | |
|---|---|
| FUNCTION: | Process records received from presentation services (PS). This procedure is called by the half-session router ("Chapter 6.0. Half-Session"). |
| INPUT: | PS_TO_HS_RECORD and the form of response requested for the last chain received |
| OUTPUT: | Indication may be set that a negative response is to be sent to the next chain received (LOCAL.SEND_ERROR_RSP_STATE); a SIGNAL may be sent to the partner half-session. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| PROCESS_SEND_PARM | page 6.1-35 |
| SEND_RSP_BIU | page 6.1-38 |
| DFC_SEND_FSMS | page 6.1-25 |
| FSM_CHAIN_RCV_FMP19 | page 6.1-44 |
| PS_TO_HS_RECORD | page A-24 |
| SEND_DATA_RECORD | page A-24 |
| SEND_ERROR | page A-24 |
| REQUEST_TO_SEND | page A-24 |
| CONFIRMED | page A-24 |
| LOCAL | page 6.0-6 |

```
Select based on PS_TO_HS_RECORD type:
  When SEND_DATA_RECORD
    Call PROCESS_SEND_PARM(SEND_PARM from input record) (page 6.1-35).

  When CONFIRMED
    If last request received was RQD2 or RQD3 then
       Call SEND_RSP_BIU (page 6.1-38) to send normal-flow,
        positive response.  BIU_PTR passed to procedure has null value.

  When SEND_ERROR
    If state of FSM_CHAIN_RCV_FMP19 = BETC (between chains) then
       Set LOCAL.SEND_ERROR_RSP_STATE to NEG_OWED to indicate that a negative
        response should be sent to the next RU received.
    Else (send -RSP to chain currently being processed)
       Call SEND_RSP_BIU (page 6.1-38) to send normal-flow, -RSP with
        sense data X'08460000'. BIU_PTR passed to the procedure has null value.

  When REQUEST_TO_SEND
      Create a BIU and initialize it to all 0's.  See Appendix D.
      Set EFI to indicate expedited.
      Set RH as described in Figure 6.1-9 on page 6.1-12.
      Set RU as described under SIG request in Appendix E.
      Call DFC_SEND_FSMS(BIU) (page 6.1-25).
```

## 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: | RM_TO_HS_RECORD, indication that session just started, first speaker indicator, primary or secondary half-session indicator, and LOCAL.SQN_SEND_CNT.NUMBER<br><br>In addition an HS_PS_CONNECTED record may be received from RM |
| OUTPUT: | The following RUs may be sent: Bid with Attach (an Attach carrying BB), Bid LUSTAT (A LUSTAT carrying BB) BIS, RTR, or a YIELD SESSION LUSTAT (LUSTAT carrying CEB).<br><br>The following fields may be altered: LOCAL.CURRENT_BRACKET_SQN.NUMBER, LOCAL.CURRENT_BRACKET_SQN.BRACKET_STARTED_BY, LOCAL.PHS_BB_REGISTER.NUMBER, LOCAL.SHS_BB_REGISTER.NUMBER,<br><br>In addition, the ID of the PS that is connected to this HS is saved to identify the PS that is using this HS, indication that session just started. |
| NOTE: | This procedure uses the BIU (see Appendix D ). In addition, the EFI field of the TH may be set. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| DFC_SEND_FSMS | page 6.1-25 |
| PROCESS_SEND_PARM | page 6.1-35 |
| FSM_BSM_FMP19 | page 6.1-43 |
| RM_TO_HS_RECORD | page A-28 |
| BID_WITHOUT_ATTACH | page A-29 |
| BID_WITH_ATTACH | page A-28 |
| HS_PS_CONNECTED | page A-29 |
| LOCAL | page 6.0-6 |

```
Select based on RM_TO_HS_RECORD type:
  When BID_WITH_ATTACH
     If the session just started (this is the first conversation on this session) then
        (no need to set current bracket sequence number because it has already
         been properly initialized)
        Receive the HS_PS_CONNECTED record that RM sends immediately after it sends the
         BID_WITH_ATTACH record.  Save the PS identifier (HS_PS_CONNECTED.PS_ID).
        Call FSM_BSM_FMP19 (page 6.1-43) with an INB signal to indicate that
         this half-session is connected to a PS.
        Record that the session did not just start.

     Else (the session did not just start)
        If this half-session is the first speaker then
           Receive the HS_PS_CONNECTED record that RM sends immediately following the
            BID_WITH_ATTACH record.  Save the PS identifier (HS_PS_CONNECTED.PS_ID).
           Call FSM_BSM_FMP19 with an INB signal to indicate that this half-session is
            connected to a PS (page 6.1-43).

           The following sets the current bracket sequence number and LOCAL.*_BB_REGISTER
            before the BB request is sent:
           Set LOCAL.CURRENT_BRACKET_SQN.NUMBER to LOCAL.SQN_SEND_CNT.NUMBER + 1 (taking
            the wrap case into account).
           If the half-session is primary then
              Set LOCAL.CURRENT_BRACKET_SQN.BRACKET_STARTED_BY to PRI.
              Set LOCAL.PHS_BB_REGISTER.NUMBER to LOCAL.CURRENT_BRACKET_SQN.NUMBER.
           Else
              Set LOCAL.CURRENT_BRACKET_SQN.BRACKET_STARTED_BY to SEC.
              Set LOCAL.SHS_BB_REGISTER.NUMBER to LOCAL.CURRENT_BRACKET_SQN.NUMBER.

     Call PROCESS_SEND_PARM(BID_WITH_ATTACH.SEND_PARM) (page 6.1-35).
```

```
When BID_WITHOUT_ATTACH
   Create and send a (LUSTAT, BB, RQD1) request to bid for a conversation on
    this session as follows:
   Create a BIU and initialize it to all 0's.  Set EFI to indicate normal-flow.
    Set the RH to indicate DFC, FMH, BC, EC, RQD1, QR, and BB.  Set the RU to an
    LUSTAT as described in Appendix E.
   Call DFC_SEND_FSMS(BIU) (page 6.1-25).

When BIS_REPLY
   Create a BIU and initialize it to all 0's.  Set EFI to indicate normal-flow.
    Set the RH to indicate DFC, FMH, BC, EC, RQE3, QR, and BB.  Set the RU to
    BIS as described in Appendix E.
   Call DFC_SEND_FSMS(BIU) (page 6.1-25).

When BIS_RQ
   Same as processing for BIS_REPLY (above) except RH indicates RQE1 instead of RQE3.

When HS_PS_CONNECTED
   Save the ID of the PS (HS_PS_CONNECTED.PS_ID) that is connected to this HS.
   Call FSM_BSM_FMP19 (page 6.1-43) with an INB signal to indicate that
    this half-session is connected to a PS.
   If the session did not just start (i.e., this is not the first conversation
    on this session) then
      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
       the wrap case into account).
      If the half-session is primary then
         Set LOCAL.CURRENT_BRACKET_SQN.BRACKET_STARTED_BY to PRI.
         Set LOCAL.PHS_BB_REGISTER.NUMBER to LOCAL.CURRENT_BRACKET_SQN.NUMBER.
      Else
         Set LOCAL.CURRENT_BRACKET_SQN.BRACKET_STARTED_BY to SEC.
         Set LOCAL.SHS_BB_REGISTER.NUMBER to LOCAL.CURRENT_BRACKET_SQN.NUMBER.
   Else (session just started)
      Record that the session did not just start.

When RTR_RQ
   Create a BIU and initialize it to all 0's.  Set EFI to indicate normal-flow.
    Set the RH to indicate DFC, FMH, BC, EC, and RQD1.  Set the RU to RTR as
    described in Appendix E.
   Call DFC_SEND_FSMS(BIU) (page 6.1-25).

When YIELD_SESSION
   If the session just started then
      Record that the session did not just start.
   The following sends an (LUSTAT, RQE1, CEB) to end the current conversation on
    this session.
   Create a BIU and initialize it to all 0's.  Set EFI to indicate normal-flow.
    Set the RH to indicate DFC, FMH, BC, EC, RQE1, and CEB.  Set the RU to LUSTAT as
    described in Appendix E.
   Call DFC_SEND_FSMS(BIU) (page 6.1-25).
```

## DFC_SEND_FROM_LNS

| | |
|---|---|
| FUNCTION: | Process record received from LU network services (LNS). This procedure is called by the half-session router ("Chapter 6.0. Half-Session") and is used for SSCP-LU half-sessions only. |
| INPUT: | HS_SEND_RECORD variant of LNS_TO_HS_RECORD |
| OUTPUT: | TC.SEND procedure is called to send the data that is included in the input record to the partner half-session. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| TC.SEND | page 6.2-13 |
| FSM_IMMEDIATE_RQ_MODE_SEND | page 6.1-48 |
| FSM_IMMEDIATE_RQ_MODE_RCV | page 6.1-48 |
| LOCAL | page 6.0-6 |
| LNS_TO_HS_RECORD | page A-15 |
| HS_SEND_RECORD | page A-16 |

```
If FM profile = 0 (FM profile 0 uses immediate request mode.) then
   Call FSM_IMMEDIATE_RQ_MODE_SEND(the BIU from the HS_SEND_RECORD.PIU) (page 6.1-48).
   Call FSM_IMMEDIATE_RQ_MODE_RCV(the BIU from the HS_SEND_RECORD.PIU) (page 6.1-48).

Call TC.SEND(the BIU from the HS_SEND_RECORD.PIU along with the EFI and SNF) (page 6.2-13).
```

## TRY_TO_RCV_SIGNAL

| | |
|---|---|
| FUNCTION: | Determine 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: | Indication that a SIGNAL has been received (LOCAL.SIG_RECEIVED), the sequence number of the signal, LOCAL.CURRENT_BRACKET_SQN, LOCAL.PHS_BB_REGISTER, LOCAL.SHS_BB_REGISTER |
| OUTPUT: | REQUEST_TO_SEND sent to PS if required, indication that a SIGNAL has been received (LOCAL.SIG_RECEIVED) may be altered |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_BSM_FMP19 | page 6.1-43 |
| REQUEST_TO_SEND | page A-13 |
| LOCAL | page 6.0-6 |

```
If the state of FSM_BSM_FMP19 is INB and LOCAL.SIG_RECEIVED = YES then
   If the sequence number of the received SIGNAL request = LOCAL.CURRENT_BRACKET_SQN then
      (a SIGNAL request has been received for the current bracket).
      Create and send a REQUEST_TO_SEND record to PS.
      Set LOCAL.SIG_RECEIVED to NO.
   Else (the SIGNAL is either stray or future)
      Set BB_REGISTER (see below) to the low-order 15 bits of either LOCAL.PHS_BB_REGISTER or
       LOCAL.SHS_BB_REGISTER according to the value of the high order bit of the SIGNAL
       sequence number (e.g., if it indicates primary (1) use PHS_BB_REGISTER).
      Set SIG_NUMBER (see below) to the low-order 15 bits of the SIGNAL sequence number.

      Calculate (SIG_NUMBER - BB_REGISTER) modulo 2**15.
      If the result is 0 or > 2**14 then
         The SIGNAL is a stray that was intended for a previous conversation.
         Optionally log the condition and set LOCAL.SIG_RECEIVED to NO.
      Else
         The SIGNAL is for a future conversation.  Save it until the bracket in
          which it was sent sent becomes the current bracket.
```

## DFC_RCV

| | |
|---|---|
| FUNCTION: | Process BIUs received from TC. This procedure is called by TC ("Chapter 6.2. Transmission Control"). |
| INPUT: | BIU, FM profile type, LU_ID, and HS_ID |
| OUTPUT: | LOCAL.SIG_RECEIVED is set if SIGNAL is received and the SNF of the SIGNAL is saved. |
| NOTE: | This procedure and the procedures it calls use the BIU see Appendix D and Appendix E . In addition, the EFI and SNF fields of the TH are used. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| DFC_RCV_FSMS | page 6.1-24 |
| FORMAT_ERROR | page 6.1-26 |
| SEND_RSP_BIU | page 6.1-38 |
| STRAY_RSP | page 6.1-41 |
| SEND_NEG_RSP_OR_LOG | page 6.1-37 |
| FORMAT_ERROR_SSCP_LU | page 6.1-30 |
| STATE_ERROR_SSCP_LU | page 6.1-40 |
| FSM_IMMEDIATE_RQ_MODE_SEND | page 6.1-48 |
| FSM_IMMEDIATE_RQ_MODE_RCV | page 6.1-48 |
| LOCAL | page 6.0-6 |
| HS_RCV_RECORD | page A-11 |

```
If FM profile for this session is 19 (X'13'), indicating LU-LU session, then
   If the BIU indicates RQ, FMD, ¬SD, CODE1, and the RU length > 0 then
      Translate the data in the RU from ASCII to EBCDIC.

   Call FORMAT_ERROR(BIU) to perform optional format error checks (page 6.1-26).
   If a format error was found in the BIU then
      Return to the HS router ("Chapter 6.0. Half-Session") 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 RQ then
         If EFI = normal-flow then
            Call DFC_RCV_FSMS(BIU) (page 6.1-24).
         Else (expedited-flow SIGNAL request)
            Save only the latest SIGNAL request received.  Set LOCAL.SIG_RECEIVED = YES and
             save the sequence number.  The sequence number is used in determining the
             bracket the SIGNAL was intended for.
            Call SEND_RSP_BIU (page 6.1-38) to send an expedited positive
             response to the SIGNAL request immediately.
      Else (RSP)
         Call STRAY_RSP(BIU) to determine if response is stray (page 6.1-41).
         If response is not stray then
            Call DFC_RCV_FSMS(BIU) (page 6.1-24).

Else (CP-LU half-session, FM profile 0 or 6.)
   Call FORMAT_ERROR_SSCP_LU(BIU) (page 6.1-30).
   Call STATE_ERROR_SSCP_LU(BIU) (page 6.1-40).
   If there is either a format or state error then
      Call SEND_NEG_RSP_OR_LOG(BIU) (page 6.1-37).
   Else (no errors)
      If FM profile = 0 then (FM profile 0 uses immediate request mode.)
         Call FSM_IMMEDIATE_RQ_MODE_SEND(BIU) (page 6.1-48).
         Call FSM_IMMEDIATE_RQ_MODE_RCV(BIU) (page 6.1-48).

      Incorporate the BIU in an HS_RCV_RECORD and send it to the LNS associated with this HS.
       The HS_RCV_RECORD includes the HS_ID to identify this half-session.
```

## DFC_RCV_FSMS

| | |
|---|---|
| FUNCTION: | Enforce data flow control protocols for received requests and responses. |
| INPUT: | BIU containing request or response, LOCAL.SEND_ERROR_RSP_STATE |
| OUTPUT: | LOCAL.PHS_BB_REGISTER or LOCAL.SHS_BB_REGISTER; LOCAL.SEND_ERROR_RSP_STATE; RSP_TO_REQUEST_TO_SEND record may be sent to PS |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| RCV_STATE_ERROR | page 6.1-36 |
| GENERATE_RM_PS_INPUTS | page 6.1-31 |
| SEND_RSP_TO_RM_OR_PS | page 6.1-39 |
| UPDATE_FSMS | page 6.1-42 |
| SEND_RSP_BIU | page 6.1-38 |
| FSM_RCV_PURGE_FMP19 | page 6.1-50 |
| FSM_CHAIN_RCV_FMP19 | page 6.1-44 |
| FSM_CHAIN_SEND_FMP19 | page 6.1-46 |
| RSP_TO_REQUEST_TO_SEND | page A-13 |
| LOCAL | page 6.0-6 |

```
Call RCV_STATE_ERROR(BIU) (page 6.1-36). These checks are optional.
If a state error is found then
   An error has occurred that will cause this session to be deactivated and the
    half-session process to be destroyed.  LOCAL.SENSE_CODE contains sense data
    indicating the type of error.  The HS router ("Chapter 6.0. Half-Session") will
    cause the abnormal termination of the half-session as a result of LOCAL.SENSE_CODE
    being set.

Else (no state error)
   Select based on RRI and EFI:
      When normal-flow request
         If BBI = BB then
            Set LOCAL.PHS_BB_REGISTER.NUMBER (if this half-session is primary) or
             LOCAL.SHS_BB_REGISTER.NUMBER (if secondary) to the low-order 15 bits
             of request SNF.

         If the state of FSM_RCV_PURGE_FMP19 ≠ PURGE then
            Call GENERATE_RM_PS_INPUTS(BIU) (page 6.1-31).
         Else
            Call UPDATE_FSMS(BIU) (page 6.1-42).

         If LOCAL.SEND_ERROR_RSP_STATE = NEG_OWED, BCI = BC, BBI = ¬BB, and
          (RU category is FMD or this request is an LUSTAT) then
            Call SEND_RSP_BIU(BIU, NORMAL, NEG, X'08460000') (page 6.1-38) to send
             negative response to the chain.
            Set LOCAL.SEND_ERROR_RSP_STATE to RESET.

         If the state of FSM_CHAIN_RCV_FMP19 = PEND_RSP (a response is owed), CEBI = CEB,
          and form of response requested is RQD1 then
            Call SEND_RSP_BIU(BIU, NORMAL, POS, X'00000000') (page 6.1-38) to send
             a positive response.

      When normal-flow response
         Call SEND_RSP_TO_RM_OR_PS(BIU) (page 6.1-39).
         Call FSM_CHAIN_SEND_FMP19(BIU) (page 6.1-46).

      When expedited-flow response (i.e., a positive response to SIGNAL)
         Create and send a RSP_TO_REQUEST_TO_SEND record to PS.
```

## DFC_SEND_FSMS

| | |
|---|---|
| FUNCTION: | Maintain states while sending requests and responses. |
| INPUT: | BIU containing request or response |
| OUTPUT: | TC.SEND is called with the BIU to send. In addition, the following fields may be set: sequence number for request or response, LOCAL.SQN_SEND_CNT, LOCAL.PHS_BB_REGISTER, LOCAL_SHS_BB_REGISTER, and RH fields. |
| NOTE: | This procedure and the procedures it calls use the BIU see Appendix D and Appendix E . In addition, the EFI and SNF fields of the TH are used. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| TC.SEND | page 6.2-13 |
| FSM_CHAIN_RCV_FMP19 | page 6.1-44 |
| FSM_CHAIN_SEND_FMP19 | page 6.1-46 |
| LOCAL | page 6.0-6 |

```
Select based on BIU.RRI and BIU.EFI:
   When normal-flow request
      Increment LOCAL.SQN_SEND_CNT.SQN by 1 (taking the wrap case into account)
       and assign it to the SNF for this request.

      If CEBI = CEB then
         Indicate RQD1 on this CEB request if necessary.  A count is kept of all the
          normal-flow requests sent and received.  When this count exceeds 16384 (2**14)
          the next CEB request sent indicates RQD1 so that any SIGNAL requests or responses
          are flushed (received by this half-session) before the response to the RQD1 request.
          This allows stray SIGNALs and responses to be accurately recognized.

      If the state of FSM_CHAIN_RCV_FMP19 = PEND_SEND_REPLY then
         Call FSM_CHAIN_RCV_FMP19(BIU) (page 6.1-44); this request is an implicit response.

      If BBI = BB then
         Set LOCAL.PHS_BB_REGISTER.NUMBER (if this half-session is primary) or
          LOCAL.SHS_BB_REGISTER.NUMBER (if secondary) to the low-order 15 bits
          of request SNF.

      If BCI = BC then
         Call FSM_CHAIN_SEND_FMP19(BIU, BEGIN_CHAIN) (page 6.1-46).

      If ECI = EC then
         Call FSM_CHAIN_SEND_FMP19(BIU, END_CHAIN).

      If it is specified to send the data as ASCII (implementation-defined) then
         Translate the data in the RU from EBCDIC to ASCII.  Set CSI to CODE1.

   When normal-flow response
      If this is an RTR response then
         Set the SNF to the SNF value received on the RTR request.
      Else (response to FMD or LUSTAT)
         If this is a response to a BB chain then
            Set the high-order bit of the response SNF to indicate the half-session
             that sent the BB chain (if primary sent the BB, then the bit is 1;
             otherwise, it's 0).  Set the low-order 15 bits of the response SNF to
             the low-order 15 bits of the BB request.
         Else
            Set the SNF to LOCAL.CURRENT_BRACKET_SQN.
      Call FSM_CHAIN_RCV_FMP19(BIU) (page 6.1-44).

   When expedited-flow request (i.e., a SIGNAL request)
      Set the SNF to LOCAL.CURRENT_BRACKET_SQN.

   When expedited-flow response (i.e., a SIGNAL response)
      Set the SNF to the SNF value received on the SIGNAL request.

Call TC.SEND(BIU along with the EFI and SNF of the TH) (page 6.2-13).
```

FORMAT_ERROR

| | |
|---|---|
| FUNCTION: | Perform format checks on all requests and responses for LU-LU session. These checks are optional. None, some, or all of these checks may be done. |
| INPUT: | BIU |
| OUTPUT: | TRUE if format error; otherwise, FALSE. If TRUE, LOCAL.SENSE_CODE is set to appropriate sense data. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FORMAT_ERROR_RQ_FMD | page 6.1-29 |
| FORMAT_ERROR_RQ_DFC | page 6.1-28 |
| FORMAT_ERROR_NORM_RSP | page 6.1-27 |
| FORMAT_ERROR_EXP_RSP | page 6.1-27 |
| LOCAL | page 6.0-6 |

```
Select based on one of the following conditions:
   When request with RU category of FMD
      Call FORMAT_ERROR_RQ_FMD(BIU) (page 6.1-29).

   When request with RU category of DFC
      Call FORMAT_ERROR_RQ_DFC(BIU) (page 6.1-28).

   When normal-flow response
      Call FORMAT_ERROR_NORM_RSP(BIU) (page 6.1-27).

   When expedited-flow response
      Call FORMAT_ERROR_EXP_RSP(BIU) (page 6.1-27).

(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 ≠ 0 then
   Return with a value of TRUE (format error found).
Else
   Return with a value of FALSE (no format error found).
```

FORMAT_ERROR_EXP_RSP

| | |
|---|---|
| FUNCTION: | Perform format checks on expedited-flow responses. These checks are optional. |
| INPUT: | BIU containing expedited-flow response |
| OUTPUT: | If error, LOCAL.SENSE_CODE is set to appropriate sense data. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 6.0-6 |

```
Select, in order, based on fields in the BIU:
   When RU category is not 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 request code ≠ SIGNAL
      Set LOCAL.SENSE_CODE to X'40120000'.
   When RTI = NEG (-RSP to expedited request)
      Set LOCAL.SENSE_CODE to BIU.SENSE_CODE.
```

FORMAT_ERROR_NORM_RSP

| | |
|---|---|
| FUNCTION: | Perform format checks on normal-flow responses. These checks are optional. |
| INPUT: | BIU containing normal-flow response |
| OUTPUT: | If error, LOCAL.SENSE_CODE is set to appropriate sense data. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 6.0-6 |

```
Select,in order, based on BIU fields:
   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 category is DFC and FI = ¬FMH
      Set LOCAL.SENSE_CODE to X'400F0000'.
   When RU category is FMD, RTI = POS, and FI = FMH
      Set LOCAL.SENSE_CODE to X'400F0000'.
   When RTI = NEG (negative response) and the sense data 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: | BIU containing DFC request |
| OUTPUT: | If error, LOCAL.SENSE_CODE is set to appropriate sense data. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FORMAT_ERROR_RQ_FMD | page 6.1-29 |
| LOCAL | page 6.0-6 |

```
Select, in 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 request code is SIGNAL but the SIGNAL Extension field is not
    set to "soft"
      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 request code is LUSTAT then
         If LUSTAT Status Value field is "no-op" (only valid LUSTAT type) then
            Call FORMAT_ERROR_RQ_FMD(BIU), since LUSTAT is like FM data (page 6.1-29).
             (LOCAL.SENSE_CODE set by called procedure if error.)
         Else
            Set LOCAL.SENSE_CODE to X'10050000'.

      Else (not LUSTAT DFC request)
         Select, in order, based on one of the following:
            When (request code is BIS and form of response requested is not RQE1 or RQE3) or
             (request code is not BIS and form of response requested is not RQD1)
               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: | BIU containing FMD request, indication that alternate code will or will not be used |
| OUTPUT: | If error, LOCAL.SENSE_CODE is set to appropriate sense data. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 6.0-6 |

```
Select, in order, based on one of the following conditions:
   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 and RU category is FMD and ¬(FI = FMH and FM header type = 5)
      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 used with FM profile 19.)
      Set LOCAL.SENSE_CODE to X'40040000'.
   When CDI = CD and ECI = ¬EC (CD allowed only on EC)
      Set LOCAL.SENSE_CODE to X'40090000'.

   When CDI = CD and form of response requested is RQD1 (CD may not be sent RQD1)
      Set LOCAL.SENSE_CODE to X'40090000'.
   When CEBI = CEB and ECI = ¬EC
      Set LOCAL.SENSE_CODE to X'40040000'.

   When (BB, ¬QR) request is received from the bidder or
    (BB, QR) request is received from the first speaker
      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 RU category is FMD, 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, RQE not allowed from bidder)
      Set LOCAL.SENSE_CODE to X'40040000'.

   When FI = FMH, RU category is FMD, and FM header type is not 5 or 7
      Set LOCAL.SENSE_CODE to X'10084001'.
```

## FORMAT_ERROR_SSCP_LU

| | |
|---|---|
| FUNCTION: | Perform format error checks on RUs received on the SSCP-LU secondary half-session (for FM profiles 0 and 6). These checks are optional; none, some, or all of the checks may be done. |
| INPUT: | BIU, indication of FM profile type |
| OUTPUT: | TRUE if error; otherwise, FALSE. If TRUE, LOCAL.SENSE_CODE is set to appropriate sense data. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 6.0-6 |

```
If expedited-flow then
   Set LOCAL.SENSE_CODE to X'40110000'.
Else (normal-flow)
   If request (RRI = RQ) then
      Select, in order, based on one of the following conditions:
         When the RU length is < 3
            Set LOCAL.SENSE_CODE to X'10020000'.
         When RU category is not FMD
            Set LOCAL.SENSE_CODE to X'40110000'.
         When FI ≠ FMH
            Set LOCAL.SENSE_CODE to X'400F0000'.
         When SDI = SD
            Set LOCAL.SENSE_CODE to the first 4 bytes of the RU data.
         When BCI = ¬BC or ECI = ¬EC
            Set LOCAL.SENSE_CODE to X'400B0000'.
         When FM profile is 0 and the form of response requested is not RQD
            Set LOCAL.SENSE_CODE to X'40140000'.
         When FM profile is 6 and the form of response requested is not RQE, RQD, or RQN
            Set LOCAL.SENSE_CODE to X'40140000'.
         When QRI = QR
            Set LOCAL.SENSE_CODE to X'40150000'.
         When PI = PAC
            Set LOCAL.SENSE_CODE to X'40080000'.
         When BBI = BB, EBI = EB, or CEBI = CEB
            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'.
    Else (response)
       Select, in order, based on one of the following conditions:
          When (RTI = POS and RU length < 3) or (RTI = NEG and RU length < 7)
             Set LOCAL.SENSE_CODE to X'10020000'.
          When RU category is not FMD
             Set LOCAL.SENSE_CODE to X'40110000'.
          When FI ≠ FMH
             Set LOCAL.SENSE_CODE to X'400F0000'.
          When BCI = ¬BC or ECI = ¬EC
             Set LOCAL.SENSE_CODE to X'400B0000'.
          When (RTI = POS and SDI = SD) or (RTI = NEG and SDI = ¬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'.

If LOCAL.SENSE_CODE = 0 then
   Return with a value of FALSE (no format error found).
Else
   Return with a value of TRUE (format error found).
```

## GENERATE_RM_PS_INPUTS

| | |
|---|---|
| FUNCTION: | Generate the appropriate records for RM and PS based on the passed BIU's content. |
| INPUT: | BIU containing normal-flow request, information about the last request sent<br><br>In addition, a BID_RSP or an RTR_RSP record may be received 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:

| | |
|---|---|
| PROCESS_RU_DATA | page 6.1-34 |
| OK_TO_REPLY | page 6.1-33 |
| SEND_RSP_BIU | page 6.1-38 |
| UPDATE_FSMS | page 6.1-42 |
| FSM_BSM_FMP19 | page 6.1-43 |
| FSM_RCV_PURGE_FMP19 | page 6.1-50 |
| HS_TO_RM_RECORD | page A-13 |
| ATTACH_HEADER | page A-13 |
| BID | page A-14 |
| BID_RSP | page A-14 |
| FREE_SESSION | page A-15 |
| BIS_RQ | page A-14 |
| BIS_REPLY | page A-14 |
| RTR_RQ | page A-15 |
| RTR_RSP | page A-15 |
| BID_RSP | page A-28 |
| RTR_RSP | page A-30 |
| LOCAL | page 6.0-6 |

```
Select, in order, based on one of the following conditions:
   When BB request
      Create and send a BID record to RM.
      Receive the BID_RSP from RM.
      If a positive Bid response is received (BID_RSP.RTI = POS) then
          Call UPDATE_FSMS(BIU) (page 6.1-42).
          If RU category is FMD then
             Call PROCESS_RU_DATA(BIU) (page 6.1-34).
          Else
             Call SEND_RSP_BIU(BIU, NORMAL, POS, X'00000000') (page 6.1-38) to send
              a positive response to the BB request.

      Else (negative response to bid)
          Call FSM_RCV_PURGE_FMP19 SIGNAL(PURGE) (page 6.1-50) to cause the
           remainder of this BB chain to be purged.
          Call UPDATE_FSMS(BIU) (page 6.1-42).
          Call SEND_RSP_BIU(BIU, NORMAL, NEG, BID_RSP.SENSE_CODE) (page 6.1-38) to send
           a negative response to the BB request.

   When BIS request
      Call UPDATE_FSMS(BIU) (page 6.1-42).
      If the form of response requested = RQE1 then
         Create and send a BIS_RQ record to RM.
      Else (RQE3)
         Create and send a BIS_REPLY record to RM.
```

```
     When RTR request
        Create and send an RTR_RQ record to RM.
        Call UPDATE_FSMS(BIU) (page 6.1-42).
        Receive RTR_RSP from RM.
        If RTR_RSP.RTI = POS then
           Call SEND_RSP_BIU(BIU, NORMAL, POS, BID_RSP.SENSE_CODE) (page 6.1-38) to send
             a positive response to the RTR request.
        Else (negative response to RTR)
           Call SEND_RSP_BIU(BIU, NORMAL, NEG, RTR_RSP.SENSE_CODE) (page 6.1-38) to send
             a negative response to the RTR request.

    Otherwise
       Call OK_TO_REPLY(BIU) (page 6.1-33) to determine if BIU is a reply.
       If BIU is a reply then
          If FSM_BSM_FMP19 is in BETB state, and the last sent chain carried BB
           (this BIU is a reply to BB) then
             Create and send a BID_RSP (positive) record to RM.
             Receive the HS_PS_CONNECTED record from RM (this is a reply to the BID_RSP record).
             Record the ID of the PS connected to this HS.
             Set LOCAL.CURRENT_BRACKET_SQN to the sequence number of the sent BB request.
             Call FSM_BSM_FMP19 (page 6.1-43) with an INB signal to indicate
              that this HS is now connected to a PS.

       If the last chain sent was RQE2 or RQE3 then
          Create and send a CONFIRMED record to PS.

       Call PROCESS_RU_DATA(BIU) (page 6.1-34).
       Call UPDATE_FSMS(BIU) (page 6.1-42).
```

## INVALID_SENSE_CODE

| | |
|---|---|
| FUNCTION: | Determine if sense data on negative response is valid. |
| INPUT: | BIU containing negative response, information about the last chain sent, first speaker indicator |
| OUTPUT: | TRUE if invalid sense data; otherwise, FALSE. |

```
If this is a response to a BB chain then
   If this half-session is first speaker then
      If the response sense data is not X'08460000' or X'088B0000' then
         Return with a value of TRUE (invalid sense data).
   Else (bidder)
      If response to LUSTAT then (i.e., RSP(LUSTAT,BB))
         If the response sense data is not X'08130000', X'08140000',
          or X'088B0000' then
            Return with a value of TRUE (invalid sense data).
      Else (response to BB not LUSTAT)
         If the response sense data is not X'08130000', X'08140000',
          X'088B0000', X'08460000' then
            Return with a value of TRUE (invalid sense data).

Else (response to ¬BB chain)
   If response to RTR then
      If the response sense data 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 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: | BIU containing a normal-flow request, LOCAL.CURRENT_BRACKET_SEQUENCE_NUMBER, information about the last chain sent |
| OUTPUT: | TRUE if valid reply; otherwise, FALSE. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_BSM_FMP19 | page 6.1-43 |
| FSM_CHAIN_RCV_FMP19 | page 6.1-44 |
| FSM_CHAIN_SEND_FMP19 | page 6.1-46 |
| LOCAL | page 6.0-6 |

```
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 BB or ¬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 6.1-44 and page 6.1-46)
      Return with a value of FALSE (not valid reply).

   When receiving and state of FSM_BSM_FMP19 (page 6.1-43) 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).

Return with a value of TRUE (a 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: | BIU containing a normal-flow request, LOCAL.SHS_BB_REGISTER, LOCAL.PHS_BB_REGISTER, HS_ID, DCF from TH of request, indication that half-session is primary or secondary<br><br>In addition, an HS_PS_CONNECTED record may be received from RM. |
| OUTPUT: | Appropriate records sent to RM and PS; if an FMH-5 is present, LOCAL.CURRENT_BRACKET_SQN is set and ID of PS that is connected to this HS is saved. |
| NOTE: | PS and RM require that any FMH data is sent in a separate record from other RU data. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_BSM_FMP19 | page 6.1-43 |
| RECEIVE_DATA | page A-12 |
| HS_TO_RM_RECORD | page A-13 |
| ATTACH_HEADER | page A-13 |
| HS_PS_CONNECTED | page A-29 |
| LOCAL | page 6.0-6 |

Determine if a complete FM header is present. FM headers may fit into a single RU (along with other data) or span several RUs within a chain. FM header data is treated separately from other data. When FM headers span RUs (this can be determined by examining the data count field and the FMH length fields) the data is accumulated until the entire header has been received. The RH of the first (or only) RU containing an FM header indicates FMH and an RU category of FMD. FMH is indicated only in the first RU; successive RUs containing FM header data indicate ¬FMH.

```
If a complete FM header is present then
   Select based on the FM header type:
      When type 5 (Attach)
         Create and send an ATTACH_HEADER record (contains the FM header 5 data) to RM.
         Receive the HS_PS_CONNECTED record from RM (this is a reply to ATTACH_HEADER).
          Save the ID of the PS that is connected to this HS.
         Call FSM_BSM_FMP19 (page 6.1-43) with an INB signal to
          indicate that this HS is now connected to a PS.

         Update the current bracket sequence number using the sequence number of the
          last received BB request as follows:
         If this half-session is primary then
            Set LOCAL.CURRENT_BRACKET_SQN to LOCAL.SHS_BB_REGISTER.
         Else
            Set LOCAL.CURRENT_BRACKET_SQN to LOCAL.PHS_BB_REGISTER.

      When type 7 (Error data)
         Create a RECEIVE_DATA record and send it to PS (FMH=YES, TYPE=NOT_END_OF_DATA,
          DATA=FM header 7 data from BIU).

If EC or data other than FM header data is present then
   Create and send a RECEIVE_DATA record to PS (FMH=NO, DATA=non-FMH data from BIU,
    TYPE = see Figure 6.1-8 on page 6.1-11).
```

PROCESS_SEND_PARM

| | |
|---|---|
| FUNCTION: | Create and send FMD requests. The appropriate RH indicators are set for each RU of a chain based on the input parameters. Each RU size will not exceed the maximum RU size specified at session activation. |
| INPUT: | SEND_PARM record |
| OUTPUT: | One or more BIU records sent representing each RU, LOCAL.SEND_BUFFER |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| SEND_BIU | page 6.1-37 |
| SEND_PARM | page A-35 |
| LOCAL | page 6.0-6 |

```
If SEND_PARM.FMH = YES and LOCAL.SEND_BUFFER contains data then
   Call SEND_BIU(data in LOCAL.SEND_BUFFER, FLUSH) (page 6.1-37) to flush
    out data so this FM header may begin in a new RU.

Concatenate data to be sent (SEND_PARM.DATA) with the data in LOCAL.SEND_BUFFER.

Divide LOCAL.SEND_BUFFER into pieces of the maximum size and send all but the
last one by calling SEND_BIU and passing it the data from the send buffer and
indicating that it needs to be flushed. The last piece is saved to
minimize sending null RUs (RUs that contain no data). Otherwise, if the last
piece is sent and the next SEND from PS indicates EC but no data, a null EC
would have to be sent.

If SEND_PARM.TYPE = NOT_END_OF_DATA or
 (SEND_PARM.TYPE = FLUSH and the LOCAL.SEND_BUFFER is empty) then
   Don't send out a BIU now.

Else (send out a BIU)
   Call SEND_BIU(LOCAL.SEND_BUFFER, SEND_PARM.TYPE) (page 6.1-37).
```

## 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: | BIU containing request or response, indicator that a response to a SIGNAL is expected |
| OUTPUT: | TRUE if a state error was encountered; otherwise, FALSE. If TRUE, LOCAL.SENSE_CODE is set to appropriate sense code. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| INVALID_SENSE_CODE | page 6.1-32 |
| FSM_BSM_FMP19 | page 6.1-43 |
| FSM_QRI_CHAIN_RCV_FMP19 | page 6.1-49 |
| FSM_CHAIN_RCV_FMP19 | page 6.1-44 |
| FSM_CHAIN_SEND_FMP19 | page 6.1-46 |
| LOCAL | page 6.0-6 |

```
Select based on EFI and RRI:
   When normal-flow request
      Select 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).

         When executing FSM_BSM_FMP19(BIU) (page 6.1-43),
          FSM_CHAIN_RCV_FMP19(BIU) (page 6.1-44), or
          FSM_QRI_CHAIN_RCV_FMP19(BIU) (page 6.1-49) 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).

   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'.
         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'.
         When the QRI field of the response ≠ the QRI of the request
            Set LOCAL.SENSE_CODE to X'40210000'.
         When response is negative and contains an invalid sense data
          (call INVALID_SENSE_CODE(BIU) [page 6.1-32])
            Set LOCAL.SENSE_CODE to X'20120000'.
         When executing FSM_CHAIN_SEND_FMP19(BIU) (page 6.1-46).
          would cause a state-check (>) condition
            Execute the corresponding output code (to set LOCAL.SENSE_CODE).

   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).
```

## SEND_BIU

| | |
|---|---|
| FUNCTION: | Create and send a BIU according to passed instructions. |
| INPUT: | DATA, SEND_PARM.TYPE information from SEND_DATA_RECORD |
| OUTPUT: | Appropriate BIU sent |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| DFC_SEND_FSMS | page 6.1-25 |

```
Create a BIU and initialize it to all 0's.
If starting a new chain (the last RU sent was EC) then
   Set BCI to BC.
Set other RH indicators as described in Figure 6.1-7 on page 6.1-11.
If sending a BB chain and this half-session is the bidder then
   Set QRI to QR for every RU in this chain.
Set the RU to the passed input data.
If this BIU indicates (BC, EC) and there is no data in the RU then
   Convert the RU to an LUSTAT (RH indicates FMH and DFC; RU contains an LUSTAT
    (see Appendix E).

Call DFC_SEND_FSMS(BIU) (page 6.1-25).
```

## SEND_NEG_RSP_OR_LOG

| | |
|---|---|
| FUNCTION: | Convert the BIU to a negative response or log the error. |
| INPUT: | BIU and LOCAL.SENSE_CODE |
| OUTPUT: | Response BIU sent if possible; otherwise, error logged. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| TC.SEND | page 6.2-13 |
| LOCAL | page 6.0-6 |

```
If BIU is a response or a request with a form of response requested of RQN then
   Unable to send a negative response; optionally log the error.

Else (this is a request to which a negative response may be sent)
   Build and send a negative response.  This is done by copying the RH, EFI,
    and SNF from the request to the response and setting the following RH
    fields: RSP, SD, BC, EC, NEG, ¬PAC, ¬BB, ¬CD, CODE0, ¬ED, ¬PD, ¬CEB

   Set the response BIU RU to LOCAL.SENSE_CODE followed by the
    request code of the request BIU.  For CP-LU sessions the BIU
    indicates FMH, the RU category is FMD, and the request code is 3 bytes long.
    For request with an RU category of DFC the request code is 1 byte long.

   Call TC.SEND(response BIU along with the SNF and EFI) (page 6.2-13).
```

## SEND_RSP_BIU

| | |
|---|---|
| FUNCTION: | Create and send a response. The response is based on the request BIU (if passed by the caller) or on information about the last received chain (when a null BIU is passed). |
| INPUT: | Request BIU (may be null value), flow (expedited or normal), response type (positive or negative), sense data (Information about the last received chain will be used when the input request BIU has a null value.) |
| OUTPUT: | BIU containing response sent if possible |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| DFC_SEND_FSMS | page 6.1-25 |
| FSM_CHAIN_RCV_FMP19 | page 6.1-44 |

```
Create a response BIU and initialize it to all 0's.
Set the RH fields of the response BIU to (RSP, BC, EC).
If the input response type is negative (NEG) then
   Set the RH to (SD, NEG) and copy the input sense data into the response BIU data.

If input flow is normal then
   If input request BIU has null value then
      Copy the RU category; FI; DR1I; DR2I; QRI; and if the RU category = DFC, the request
       code; from the last received normal-flow request into the response BIU.
   Else (a request BIU was passed as input)
      Copy the RU category; FI; DR1I; DR2I; QRI; and if the RU category = DFC, the request
       code; from the input request BIU to the response BIU.

Else (expedited, the only expedited-flow response is for SIGNAL)
   Set EFI to expedited, RU category to DFC, DR1, and request code to SIGNAL
    in the response BIU.

(Note: the DFC request code always immediately follows the sense data in the RU)

If executing FSM_CHAIN_RCV_FMP19(response BIU) (page 6.1-44)
 would cause a state-check (>) condition then
   Execute the corresponding output code in the FSM.
Else
   Call DFC_SEND_FSMS(response BIU) (page 6.1-25) to send the response.
```

## SEND_RSP_TO_RM_OR_PS

| | |
|---|---|
| FUNCTION: | Build and send records to RM or PS based on the passed response BIU. |
| INPUT: | BIU containing a response, indicator that session is first speaker, information about the last sent request.<br><br>In addition, an HS_PS_CONNECTED record may be received from RM. |
| OUTPUT: | Appropriate "response" record sent to RM or PS. LOCAL.CURRENT_BRACKET_SQN 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:

| | |
|---|---|
| FSM_BSM_FMP19 | page 6.1-43 |
| CONFIRMED | page A-12 |
| RECEIVE_ERROR | page A-12 |
| BID_RSP | page A-14 |
| RTR_RSP | page A-15 |
| HS_PS_CONNECTED | page A-29 |
| LOCAL | page 6.0-6 |

```
If the response is RTR then
   Create and send an RTR_RSP record to RM.
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 6.1-43) is BETB then
         Create and send a BID_RSP (positive) record to RM.
         Receive the HS_PS_CONNECTED record from RM (this is a reply to BID_RSP record).
         Save the ID of the PS connected to this HS.
         Set LOCAL.CURRENT_BRACKET_SQN to the sequence number of the sent BB request.
         Call FSM_BSM_FMP19 (page 6.1-43) with an INB signal to
          indicate that this HS is now connected to a PS.

      If the form of response requested of the last chain sent was RQD2 or RQD3 then
         Create and send a CONFIRMED record to PS.

   Else (response is negative)
      If the response sense data is X'08460000' then
         If this half-session is not the first speaker and the last chain sent
          carried BB (this is a response to a bidder's BB with data) then
            Create and send a BID_RSP (positive) record to RM.
            Receive the HS_PS_CONNECTED record from RM (this is a reply to BID_RSP record).
            Save the ID of the PS connected to this HS.
            Set LOCAL.CURRENT_BRACKET_SQN to the sequence number of the sent BB request.
            Call FSM_BSM_FMP19 (page 6.1-43) with an INB signal to
             indicate that this HS is now connected to a PS.

         Create and send a RECEIVE_ERROR record to PS.

      Else (bracket reject, i.e., X'08130000', X'08140000', or X'088B0000')
         Create and send a BID_RSP record (indicates negative response and
          contains the sense data from the response) to RM.
```

## STATE_ERROR_SSCP_LU

| | |
|---|---|
| FUNCTION: | Perform state error checks on RUs received on the CP-LU secondary half-session (FM profiles 0 and 6). These checks are optional; none, some, or all of the checks may be done. |
| INPUT: | BIU, FM profile type, sequence number of the last sent request |
| OUTPUT: | TRUE if error; otherwise, FALSE. If TRUE, LOCAL.SENSE_CODE is set to appropriate sense data. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_IMMEDIATE_RQ_MODE_SEND | page 6.1-48 |
| FSM_IMMEDIATE_RQ_MODE_RCV | page 6.1-48 |
| LOCAL | page 6.0-6 |

```
If this session is using FM profile 0 (using immediate request mode) then
   If RRI = RQ then
      If executing FSM_IMMEDIATE_RQ_MODE_RCV(BIU) (page 6.1-48)
       would cause a state-check (>) condition then
         Execute the corresponding output code in that FSM to set LOCAL.SENSE_CODE.

   Else (response)
      If the state of FSM_IMMEDIATE_RQ_MODE_SEND (page 6.1-48) is PEND_RSP
       (Half-session is awaiting a response to a sent RQD request.) then
         If the response SNF ≠ the SNF of the last sent request then
            Set LOCAL.SENSE_CODE to X'200E0000' (response correlation error).
      Else (not waiting for a response)
         Set LOCAL.SENSE_CODE to X'200E0000' (response correlation error).

If LOCAL.SENSE_CODE = 0 then
   Return with a value of FALSE (no state error).
Else
   Return with a value of TRUE (state error).
```

## STRAY_RSP

| | |
|---|---|
| FUNCTION: | Determine if a response is stray. A stray response is one that was sent in a bracket (conversation) but received in a different (later) bracket. |
| INPUT: | BIU containing a response, information about the last request sent, LOCAL.CURRENT_BRACKET_SQN |
| OUTPUT: | TRUE if stray response; otherwise, FALSE. If stray response represents a response correlation error, LOCAL.SENSE_CODE is set. |
| NOTE: | An outstanding request is a request that has not been responded to nor replied to. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_BSM_FMP19 | page 6.1-43 |
| LOCAL | page 6.0-6 |

```
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 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.

   Return with a value of TRUE (stray response).
Else
   Return with a value of FALSE (not stray response).
```

## UPDATE_FSMS

| | |
|---|---|
| FUNCTION: | Update the appropriate FSMs for received requests. |
| INPUT: | BIU containing request |
| OUTPUT: | FSMs updated |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_RCV_PURGE_FMP19 | page 6.1-50 |
| FSM_QRI_CHAIN_RCV_FMP19 | page 6.1-49 |
| FSM_CHAIN_RCV_FMP19 | page 6.1-44 |
| FSM_CHAIN_SEND_FMP19 | page 6.1-46 |

```
Call FSM_RCV_PURGE_FMP19(BIU) (page 6.1-50).
If the state of FSM_CHAIN_SEND_FMP19 = PEND_RCV_REPLY then
   Call FSM_CHAIN_SEND_FMP19(BIU) (page 6.1-46).

If BCI = BC then
   Call FSM_CHAIN_RCV_FMP19(BIU, BEGIN_CHAIN) (page 6.1-44).

If ECI = EC then
   Call FSM_CHAIN_RCV_FMP19(BIU, END_CHAIN) (page 6.1-44).

Call FSM_QRI_CHAIN_RCV_FMP19(BIU) (page 6.1-49).
```

These are the FSM input definitions used for all the FSMs in this chapter:

- R or S: BIU that is being processed is being received or sent, respectively.
- RQ, RSP, BC, EC, CD, CEB, FMD, QR: Refer to the RH of the BIU.
- 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: BIU = RQD1, RQD2, or RQD3.
- RQE: BIU = RQE1, RQE2, or RQE3.
- REPLY: A call to OK_TO_REPLY(BIU) (page 6.1-33) returns TRUE.
- BIS: BIU is a BIS RU.
- RTR: BIU is an RTR RU.
- FMH5: BIU contains an FMH5.
- LUSTAT: BIU is a LUSTAT request or response.
- NOT_BID_REPLY: BIU = BC, ¬BB and either the last sent chain did not carry BB or a call to OK_TO_REPLY (page 6.1-33) returns a value of FALSE.
- CEB_UNCOND: BIU = CEB and response category = (RQD1|RQE1).

NOTE: FSM_IMMEDIATE_RQ_MODE_SEND and FSM_IMMEDIATE_RQ_MODE_RCV are used for CP-LU sessions. All others are used for LU-LU sessions.

FSM_BSM_FMP19

| | |
|---|---|
| FUNCTION: | Enforce the bracket protocol. State transitions are made via the signals INB (go in brackets) and BETB (go between brackets). 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: | BIU 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:<br>• BETB: between brackets<br>• INB: in bracket |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 6.0-6 |

| INPUTS STATE NAMES----> STATE NUMBERS--> | BETB 01 | INB 02 |
|---|---|---|
| SIGNAL(INB) | 2 | - |
| SIGNAL(BETB) | - | 1 |
| R,RQ,(FMD\|LUSTAT),NOT_BID_REPLY,¬FMH5,¬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: | BIU, 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 6.1-33 |
| FREE_SESSION | page A-15 |
| LOCAL | page 6.0-6 |

| STATE NAMES----> | BETC | INC | NEG RSP SENT | PEND RSP | PEND SEND REPLY |
|---|---|---|---|---|---|
| INPUTS STATE NUMBERS--> | 01 | 02 | 03 | 04 | 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(A) | / | / |
| 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<br>(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<br>  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<br>    Stop communication between PS and HS. This involves purging records currently in transit from PS and disabling PS's capability to send to this HS.<br>    Create and send a FREE_SESSION record to RM.<br>    Call FSM_BSM_FMP19 with a BETB signal (page 6.1-43). |
| 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: BIU, 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 SEND 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 6.1-33 |
| FSM_BSM_FMP19 | page 6.1-43 |
| FREE_SESSION | page A-15 |
| LOCAL | page 6.0-6 |

| STATE NAMES----> | BETC | INC | NEG RSP RCVD | PEND RSP | PEND RCV REPLY |
|---|---|---|---|---|---|
| INPUTS STATE NUMBERS--> | 01 | 02 | 03 | 04 | 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(A) | / | / |
| S,RQ,END_CHAIN,BIS | / | 1 | / | / | / |
| 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 | >(R) |
| R,RQ,REPLY | / | / | / | / | 1 |
| 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<br>  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<br>    Stop communication between PS and HS. This involves purging records currently in transit from PS and disabling PS's capability to send to this HS.<br>    Create and send a FREE_SESSION record to RM.<br>    Call FSM_BSM_FMP19 with a BETB signal (page 6.1-43). |
| R | Set LOCAL.SENSE_CODE to X'200F0000' (response protocol error). |

## FSM_IMMEDIATE_RQ_MODE_SEND

| | |
|---|---|
| FUNCTION: | Enforce the immediate request mode send protocol. It is used only on CP-LU half-sessions using FM profile 0. |
| INPUT: | BIU |
| OUTPUT: | LOCAL.SENT_RQD_SNF may be set. |
| NOTE: | The state names mean the following:<br>• RESET: no request is awaiting a response.<br>• PEND_RSP: a response is expected to the last sent request. |

| STATE NAMES----><br>INPUTS STATE NUMBERS--> | RESET<br>01 | PEND RSP<br>02 |
|---|---|---|
| S,RQ,RQD | 2 | / |
| R,±RSP | - | 1 |
| SIGNAL(RESET) | - | 1 |

## FSM_IMMEDIATE_RQ_MODE_RCV

| | |
|---|---|
| FUNCTION: | Enforce the immediate request mode receive protocol. It is used only on CP-LU half-sessions using FM profile 0. |
| INPUT: | BIU |
| OUTPUT: | LOCAL.SENSE_CODE is set if an error is found. |
| NOTE: | The state names mean the following:<br>• RESET: a response is not owed to a chain.<br>• PEND RSP: a chain was received to which a response is owed. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 6.0-6 |

| STATE NAMES----><br>INPUTS STATE NUMBERS--> | RESET<br>01 | PEND RSP<br>02 |
|---|---|---|
| R,RQ,RQD | 2 | >(E) |
| S,±RSP | - | 1 |
| SIGNAL(RESET) | - | 1 |

| OUTPUT CODE | FUNCTION |
|---|---|
| E | Set LOCAL.SENSE_CODE to X'200A0000' (immediate request mode violation). |

FSM_QRI_CHAIN_RCV_FMP19

| | |
|---|---|
| FUNCTION: | Enforce the setting of the QRI indicator in the RH. This indicator is set the same for all BIUs in a chain; i.e., all BIUs in a chain have QRI=QR or have QRI=¬QR. |
| INPUT: | BIU |
| OUTPUT: | If a QRI state error is detected, LOCAL.SENSE_CODE is set. |
| NOTE: | 1) The state names mean the following:<br>• RESET: no chain is currently being received<br>• INC QR: the chain that is being received is a QR chain<br>• INC NOT QR: the chain that is being received is not a QR chain<br>2) The implementation of this FSM is optional because it is used only to detect receive error conditions. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 6.0-6 |

| STATE NAMES----> | RESET | INC QR | INC NOT QR |
|---|---|---|---|
| INPUTS STATE NUMBERS--> | 01 | 02 | 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: | BIU |
| OUTPUT: | None |

| INPUTS / STATE NAMES----> / STATE NUMBERS--> | RESET 01 | PURGE 02 |
|---|---|---|
| R, EC | - | 1 |
| SIGNAL(PURGE) | 2 | - |
| SIGNAL(RESET) | - | 1 |

# CHAPTER 6.2. TRANSMISSION CONTROL

## INTRODUCTION

A distinct transmission control (TC) element is provided for each half-session supported in a node.

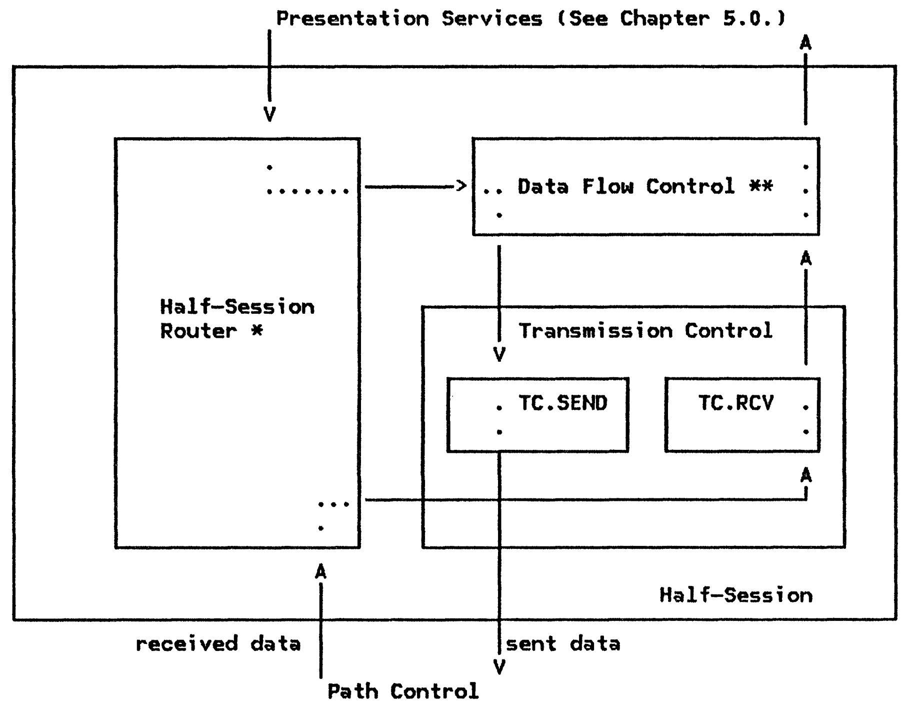
Each TC element participates in two activities:

- Initialization:
  - Variable initialization
  - Cryptography initialization
- Normal operation:
  - Sending data from data flow control (DFC) to path control (PC)
  - Receiving data from PC and giving it to DFC

The protocol machine for session initialization, TC.INITIALIZE (page 6.2-8), is invoked after LU network services (LNS) processes a BIND or ACTLU. 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 TC.SEND and TC.RCV components control sequence number checking, pacing, enciphering and deciphering, and manage expedited and normal flows.

The relationship of transmission control to the other elements of the half-session, after initialization, is shown in Figure 6.2-1.



* See "Chapter 6.0. Half-Session" for details.
** See "Chapter 6.1. Data Flow Control" for details.

Figure 6.2-1. Structure of TC and Flow of Data within the Half-Session

TC.INITIALIZE (page 6.2-8) is called by half-session initialization ("Chapter 6.0. Half-Session") during initialization when a half-session is being activated. The initialization procedure sets up pacing, CRYPTOGRAPHY VERIFICATION (CRV), and sequence number usage variables according to the TS profile in use.

For sessions that support cryptography, the initialization procedure calls TC.EXCHANGE_CRV (page 6.2-10) 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 "Appendix E. Request-Response Unit (RU) 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). Failure to compare resets the session cryptography key and the session seed. Failure to compare also causes the session to be deactivated.

Valid cryptography options are defined under the BIND format in "Appendix E. Request-Response Unit (RU) Formats"; "Appendix D. RH Formats" 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 "Appendix E. Request-Response Unit (RU) Formats" and Figure 6.2-2 on page 6.2-3). The flows involved in distributing the session seed to the LU are shown in Figure 6.2-2 on page 6.2-3.

The comments below correspond to the numbers in Figure 6.2-2 on page 6.2-3.

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

```
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

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 LNS (see "Chapter 4. LU Network Services") without passing through TC. Session control requests and responses have the header bit-settings described below.

All SC requests are issued by TC or by LNS. The following fields of the TH and RH are set for session control RUs.

<u>TH:</u> All SC requests and responses are sent expedited (the EFI bit is <u>on</u> in the TH).

<u>RH:</u> The RH settings for SC requests are defined in TC.BUILD_CRV on page 6.2-11 .

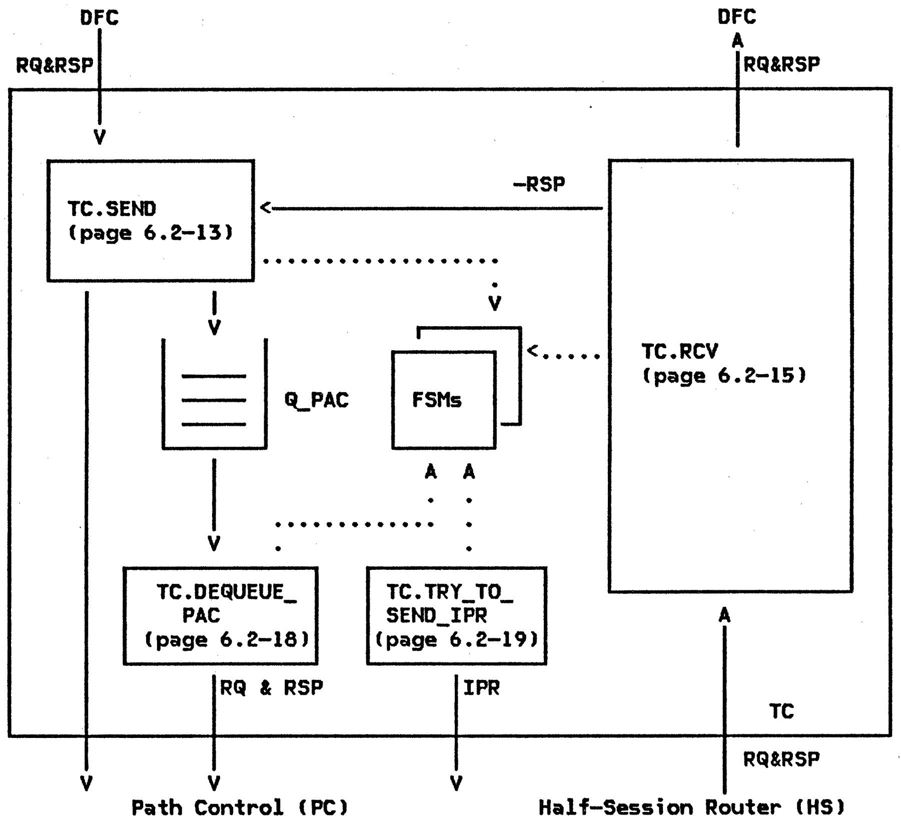


Figure 6.2-3. Interrelation of TC.SEND and TC.RCV

## NORMAL OPERATION

The TC.SEND and TC.RCV protocol machines are related as shown in Figure 6.2-3. Detailed definitions for TC.SEND and TC.RCV, the major TC procedures, are shown on page 6.2-13 and page 6.2-15, 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; this involves a queue (LOCAL.Q_PAC) for temporarily holding outgoing requests, and a set of coupled FSMs and procedures that manage the sending and receiving of pacing requests and responses (FSM_PAC_RQ_SEND [page 6.2-20] and FSM_PAC_RQ_RCV [page 6.2-21])

- 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.TRY_TO_ENCIPHER [page 6.2-14] and TC.RCV_NORM_RQ [page 6.2-17])

### TC PROCEDURES INVOKED FROM OTHER COMPONENTS OF THE HALF-SESSION

Procedures TC.RCV (page 6.2-15) and TC.TRY_TO_SEND_IPR (page 6.2-19) are invoked by the half-session router (see "Chapter 6.0. Half-Session" for details).

When the half-session router receives a message unit from path control, it calls TC.RCV to initiate TC processing of the message unit.

TC.TRY_TO_SEND_IPR, which is called periodically from the half-session router, is responsible for generating an ISOLATED PACING RESPONSE (IPR, see "Pacing") when both the architectural and resource requirements are satisfied.

TC.SEND (page 6.2-13) 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 contains more than the maximum RU size as specified in BIND.

## SEQUENCE NUMBERING OF REQUESTS AND RESPONSES

For TS profile 7 (used in LU-LU sessions, see "Appendix F. Profiles") 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 RU SIGNAL is assigned an identifier by DFC; the expedited-flow request CRV is assigned an identifier by TC; expedited-flow session-activation (BIND) and session-deactivation (UNBIND) requests are assigned identifiers by LNS (see "Chapter 4. LU Network Services").

For TS profile 1 (used in CP-LU and CP-PU sessions), identifiers are used on the normal flows as well as on the expedited flows.

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).

For further information on sequence numbering, see "Sequence Numbering of Requests and Responses" in "Chapter 6.1. Data Flow Control".

## 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.

- 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.

## 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. A TC.SEND performing send pacing has a session partner TC.RCV that is doing receive pacing. 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.

The pacing environment assumes that the receiving TC is able to accept no more than a certain number of requests (N) at a time. This number, called the _window size_, is defined when the session is being activated. Pacing operates according to the following cycle. The sending TC initially may send up to N requests. On the first request, it turns on the Pacing Request indicator. 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 number of requests that it can send before receiving a pacing response; this number is kept in the pacing count field (SEND_PACING_COUNT). This field and all others related to session-level pacing or the maximum RU size are maintained in the transmission control control block (TCCB). The TCCB is a substructure of the control block named LOCAL. When a pacing response is received, the sending TC can send N more requests and therefore increases the pacing count by N. This makes the pacing count equal to the window size (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.

Only one pacing response is generated for each pacing request. There are two methods by which the pacing response may be returned: on a normal-flow response header or on an <u>ISOLATED</u> <u>PACING</u> <u>RESPONSE</u> <u>(IPR)</u>. The IPR may be used at any time; 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.

TC.TRY_TO_SEND_IPR, which includes all the checks to determine if a pacing response should be sent, is invoked by the half-session router (see "Chapter 6.0. Half-Session"). 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, 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 with QRI=QR as it is dequeued from Q_PAC or to a response with QRI=¬QR. If FSM_PAC_RQ_SEND (page 6.2-20) is preventing the only available responses from flowing from the queue, an IPR can be generated and sent directly to PC; this prevents session deadlock, which could occur when both TCs' 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.

In the BIND RU, four fields exist that define values for the send and receive window sizes of each stage of pacing. BIND also contains the staging indicators that specify one or two stages of pacing in the PLU-to-SLU direction and in the SLU-to-PLU direction.

If pacing on a session stage in a particular direction is not to be performed, the values for the window size on that stage are set to 0. For example, if there is to be no pacing in the SLU-to-PLU direction, the PLU-receive and the SLU-send window sizes are both set to 0.

When a T2.1 node is sending a BIND to activate a session with an LU in an adjacent T2.1 node, the PLU sets the staging indicators to specify one-stage in both directions, and sets the pacing window sizes to implementation-dependent values.

## ISOLATED PACING RESPONSE (IPR)

An IPR is sent by TC.TRY_TO_SEND_IPR (page 6.2-19) to return a Pacing indication as discussed in the preceding section.

The following fields of the TH and RH are set for an IPR:

<u>TH:</u> The normal or expedited flow is indicated. The sequence number is undefined (it may be set to any value, and it is not checked by the receiver).

<u>RH:</u> IPRs are coded all 0's except for the Response indication, the Pacing Response indication, and the chaining bits; thus, the IPR RH is coded X'830100', and the test for an IPR is: RRI=RSP, ¬DR1, ¬DR2, and PI=PAC. IPR is the only response that indicates both ¬DR1 and ¬DR2.

There is no RU accompanying the TH and RH.

## 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 <u>primary</u> <u>TC</u> does during initialization is to send a CRV request and receive the CRV response. The last thing that the <u>secondary</u> <u>TC</u> 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 except CRV, during this time.

TC is not involved in enforcing immediate request mode at any other time.

TC is not involved in inforcing immediate response mode at any time.

## TRANSMISSION CONTROL CALLING TREES

Figure 6.2-4 through Figure 6.2-6 show the calling trees for transmission control initialization and CRV exchange (Figure 6.2-4), sending data (Figure 6.2-5), and receiving data (Figure 6.2-6). In addition to the procedures in these calling trees, TC also contains TC.TRY_TO_SEND_IPR, a procedure that is called only by the half-session router.

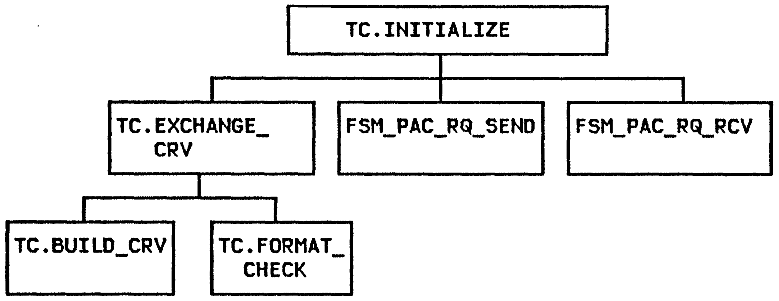


Figure 6.2-4. TC Initialization Calling Tree

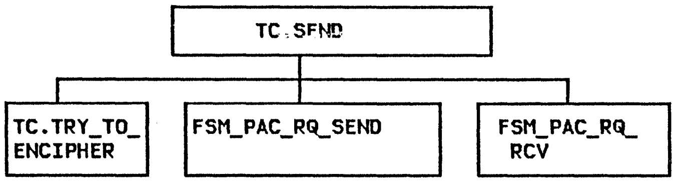
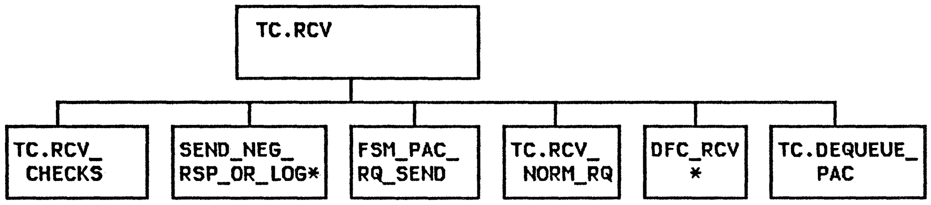


Figure 6.2-5. SEND Calling Tree



* See "Chapter 6.1. Data Flow Control" for details.

Figure 6.2-6. RCV Calling Tree

## SESSION INITIALIZATION PROCEDURES

### TC.INITIALIZE

| | |
|---|---|
| FUNCTION: | Sets up session parameters needed by TC. This procedure is called by half-session initialization (see Chapter 6.0 ) when the session is being activated. The TCCB (a substructure of LOCAL) is initialized according to whether this is a primary or secondary LU-LU half-session or a CP-LU half-session. The maximum receive RU size is initialized. |
| INPUT: | INIT_HS is a structure that indicates whether the type of session to initialize is an LU-LU session or a CP-LU session. For LU-LU sessions, the INIT_HS record contains BIND information. For CP-LU sessions, the INIT_HS record contains ACTLU information. The BIND or ACTLU information contains the values to which the fields of the TCCB will be initialized. The TS and FM profiles, 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. Caller checks that the TS profile is 1 or 7. |
| OUTPUT: | The correct initialization procedure is executed. A variable indicating that initialization was SUCCESSFUL or UNSUCCESSFUL is set. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| TC.EXCHANGE_CRV | page 6.2-10 |
| FSM_PAC_RQ_SEND | page 6.2-20 |
| FSM_PAC_RQ_RCV | page 6.2-21 |
| LOCAL | page 6.0-6 |
| INIT_HS | page A-16 |

```
Initialize the half-session according to the TS profile (see Appendix F) and the session
 activation RU (see Appendix E). The procedure has access to the INIT_HS record
 and the LOCAL control block.
 If CP-LU half-session then
    see TS profile 1 and the ACTLU RU.  Record the following information:
    •  Maximum RU size that can be received (obtained from the INIT_HS.ACTLU_IMAGE),
        converted from exponent/mantissa form to binary form (see BIND in Appendix E
        for the conversion table).  The maximum RU size parameter is not part of
        the ACTLU RU, but is initialized by LNS and passed in the ACTLU_IMAGE
    •  That identifiers are used
    •  That neither send nor receive pacing is active
    •  That cryptography is not active

 Else (LU-LU sessions--see TS profile 7 and the BIND RU)
    Record the following information:
    •  Maximum RU size sent by the partner half-session on the normal flow,
        (obtained from the INIT_HS.BIND_IMAGE), converted from exponent/mantissa
        form to binary form
    •  That sequence numbers are used
    •  For a primary half-session,
       •  Whether send pacing is active, and, if so, the primary send window size
       •  Whether receive pacing is active, and , if so, the primary receive window size
    •  For a secondary half-session,
       •  Whether send pacing is active, and, if so, the secondary send window size
       •  Whether receive pacing is active, and, if so, the secondary receive window size
    •  Whether cryptography is active

    If the BIND_IMAGE indicates that cryptography is active then
       Call TC.EXCHANGE_CRV(INIT_HS.BIND_IMAGE) (page 6.2-10) to participate in
        cryptography verification.  (The BIND image and the CRV format are found in
        Appendix E.)
       If cryptography verification is unsuccessful (LOCAL.SENSE_CODE ≠ 0) then
           Return with a value of UNSUCCESSFUL.
    Call FSM_PAC_RQ_SEND (page 6.2-20) and FSM_PAC_RQ_RCV (page 6.2-21), passing
     them RESET signals.
    Purge the pacing queue (LOCAL.Q_PAC), set the send and receive pacing counts
     (LOCAL.SEND_PACING_COUNT and LOCAL.RCV_PACING_COUNT) to the values of the
     corresponding window sizes, and initialize the receive sequence number
     (LOCAL.SQN_RCV_CNT) to 0.
Return with a value of SUCCESSFUL.
```

## TC.EXCHANGE_CRV

| | |
|---|---|
| FUNCTION: | Called from a primary half-session to initiate the exchange of CRV with a secondary and to receive RSP(CRV). Called from a secondary half-session to receive CRV and return RSP(CRV) to the primary. |
| INPUT: | BIND image (received in RSP[BIND]) contains 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.<br><br>The initialization of a secondary TC instance involves receiving a PC_TO_HS_RECORD containing a CRV request, and the initialization of a primary TC instance involves receiving a PC_TO_HS_RECORD containing a CRV response. |
| OUTPUT: | CRV exchange completed. If successful, LOCAL.SENSE_CODE = 0.<br><br>The initialization of a secondary TC instance involves sending an HS_TO_PC_RECORD containing a CRV response, and the initialization of a primary TC instance involves sending an HS_TO_PC_RECORD containing a CRV request. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| TC.FORMAT_CHECK | page 6.2-11 |
| TC.BUILD_CRV | page 6.2-11 |
| LOCAL | page 6.0-6 |
| PC_TO_HS_RECORD | page A-23 |
| HS_TO_PC_RECORD | page A-11 |

```
If primary half-session then
   Call TC.BUILD_CRV(BIND_IMAGE,BIU) (page 6.2-11) to build a CRV request BIU,
    including the appropriate PIU fields.
   Incorporate the BIU into the PIU field of the HS_TO_PC_RECORD (see page 6.2-13).
   Send the HS_TO_PC_RECORD to the path control that the half-session uses.
   Receive a PC_TO_HS_RECORD from path control.  This implies a possible wait.
   Extract the BIU from the PIU field of the PC_TO_HS_RECORD.
   If not a CRV response then
      Set LOCAL.SENSE_CODE to X'20090000' (SC protocol error).
   Else (CRV response)
      Optionally, call TC.FORMAT_CHECK(BIU) (page 6.2-11) to verify the RH.
      If the format checked out OK then
         If RTI = NEG then
            Set LOCAL.SENSE_CODE to the sense data value in the BIU.
Else (secondary half-session)
   Receive a PC_TO_HS_RECORD from path control.  This implies a possible wait.
   Extract the BIU from the PIU field of the PC_TO_HS_RECORD.
   If that received record is a CRV request then
      Optionally call TC.FORMAT_CHECK(BIU) (page 6.2-11) to verify the RH.
      If the format checked out OK then
         Check that the CRV test value was correctly encoded by the session partner by:
          deciphering the test value (bytes 2-9 of the RU data) using the session key
          and 0 as the initial chaining value, inverting the bits
          in the first 4 bytes, and comparing the results with the value that was
          generated by LNS for the +RSP(BIND).
         If the values are equal then
            Incorporate a positive response into the PIU field of the HS_TO_PC_RECORD.
            See page 6.2-13 for details on constructing the HS_TO_PC_RECORD.
            Send the HS_TO_PC_RECORD to the path control that the half-session uses.
         Else (values not equal)
            Set LOCAL.SENSE_CODE to X'08350001' (Invalid Parameter).
             (LOCAL.SENSE_CODE now has a value of nonzero, so the half-session router
             will cause UNBIND to be sent.)
      Else (RH not valid)
          (LOCAL.SENSE_CODE already has a value of nonzero, so the half-session
          router will cause UNBIND to be sent.)
   Else (not CRV request)
      Set LOCAL.SENSE_CODE to X'20090000' (SC protocol error).  (LOCAL.SENSE_CODE
       now has a value of nonzero, so the half-session router will cause UNBIND
       to be sent.)
```

## TC.BUILD_CRV

| | |
|---|---|
| FUNCTION: | This procedure builds a CRV BIU by appropriately assigning the RH and RU fields. |
| INPUT: | BIND information and BIU to be initialized. The test value sent in CRV is derived from the BIND image. |
| OUTPUT: | The CRV PIU, including the TH settings for EFI and SNF. The session cryptography seed is retained. |
| NOTE: | For the actual TH and RH bit settings see Appendix D . |

```
Set EFI to EXP.
Set SNF to some value (implementation-dependent). CRV is on the TC-TC flow,
 not the half-session to half-session flow, so is not related to the half-session
 send sequence number (LOCAL.SQN_SEND_CNT).

Set the RH to the following values:
   (RQ, SC, FMH, ¬SD, BC, EC, RQD1, ¬QR, ¬PAC, ¬BB, ¬CD, CODE0, ¬ED, ¬PD, ¬CEB).

Set the RU data to CRV request code (see page A-33).
Prepare the cryptography test value:
 Decipher the test value in the BIND image, which is in the INIT_HS record.
  Use the session cryptography key that was received from the CP in the CINIT
  request as the cryptography key, and 0 as the initial chaining value.
  See the Data Encryption Standard for details. Retain the resulting value
  for use as the session seed. Transform the result by inverting each bit
  of the first four bytes and enciphering the transformed value (use the session
  key as the cryptography key and 0 as the initial chaining value).
 Append this transformed test value to the RU data.
```

## TC.FORMAT_CHECK

| | |
|---|---|
| FUNCTION: | Checks the RH bits of the request or response. All of these checks are optional. An implementation may choose to do all, some, or none of them. |
| INPUT: | A request or response BIU. |
| OUTPUT: | OK if all bits are properly set; otherwise, NG. If NG, LOCAL.SENSE_CODE is set to a nonzero value. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 6.0-6 |

```
If EFI ≠ EXP then
   Set LOCAL.SENSE_CODE to X'40110000'.
Else (expedited BIU)
   If RRI = RQ then
      Select in the following order, based on the RH bits:
         When (SDI ≠ SD and RU_LENGTH < 1) or
          (SDI = SD and RU_LENGTH < 5)
            Set LOCAL.SENSE_CODE to X'10020000'.
         When FI ≠ FMH
            Set LOCAL.SENSE_CODE to X'400F0000'.
         When SDI = SD
            Set LOCAL.SENSE_CODE to the sense data in the BIU.
         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 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 (response)
      Select in the following order, based on RH bits:
         When (RTI = POS and RU_LENGTH < 1) or
              (RTI = NEG and RU_LENGTH < 5)
              Set LOCAL.SENSE_CODE to X'10020000'.
         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 DR1I ≠ DR1
            Set LOCAL.SENSE_CODE to X'40140000'.
         When 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'.
If LOCAL.SENSE_CODE = 0 (no error) then
   Return with a value of OK.
Else (format error)
   Return with a value of NG.
```

TC.SEND

| | |
|---|---|
| FUNCTION: | Send the input BIU to path control. If required, the message unit is enciphered. If pacing is supported, the message unit may be placed on Q_PAC rather than sent directly to path control. |
| INPUT: | BIU along with the EFI and SNF. (The RH, and the RU were set up by the procedure that passed this record to TC); whether send pacing is active; whether receive pacing is active; and the send pacing count (LOCAL.SEND_PACING_COUNT) if send pacing is active. |
| OUTPUT: | If no errors, BIU is sent to PC or placed on Q_PAC. The Pacing indicator is set to specify whether or not the BIU is a pacing request or response. If send pacing is active, LOCAL.SEND_PACING_COUNT is decremented. LOCAL.SENSE_CODE = 0. If any errors are detected, a nonzero sense code is returned to the caller in LOCAL.SENSE_CODE. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| TC.TRY_TO_ENCIPHER | page 6.2-14 |
| FSM_PAC_RQ_SEND | page 6.2-20 |
| FSM_PAC_RQ_RCV | page 6.2-21 |
| LOCAL | page 6.0-6 |
| HS_TO_PC_RECORD | page A-11 |

```
Initialize PI to ¬PAC.
Select in the following order, based on RRI and EFI RH bits:

   When EFI = EXP
      Indicate that the pacing queue does not need to be checked.

   When RRI = RQ
      If send pacing is active then
         Indicate that the pacing queue needs to be checked.
      Else (send pacing not active)
         Indicate that the pacing queue does not need to be checked.
      Call TC.TRY_TO_ENCIPHER(BIU) (page 6.2-14) to encipher the BIU.
       LOCAL.SENSE_CODE will be nonzero if enciphering failed.

   When RRI = RSP
      If send pacing is active and QRI = QR and the pacing queue (LOCAL.Q_PAC)
       is not empty then
         Indicate that the pacing queue needs to be checked.
      Else
         Indicate that the pacing queue does not need to be checked.
         If receive pacing is active and FSM_PAC_RQ_RCV (page 6.2-21)
          is in the PEND state and there are sufficient (implementation-dependent)
          resources then
            Call FSM_PAC_RQ_RCV(BIU) (page 6.2-21) to manipulate the Pacing
             indicator in this response.
```

```
If LOCAL.SENSE_CODE = 0 then
   Select, based on whether the pacing queue should be checked (as indicated
    above):

     When YES
        If RRI = RQ and LOCAL.SEND_PACING_COUNT > 0 then
           Call FSM_PAC_RQ_SEND(BIU) (page 6.2-20) to record the ability
            to send a session-level pacing request for send pacing.
           Decrement LOCAL.SEND_PACING_COUNT by 1.
           Incorporate the BIU into the PIU field of the HS_TO_PC_RECORD:
            Set the HS_ID to the identifier of the half-session that
             is sending this record.
            (EFI, SNF, the RH, and the RU were set up by the procedure that
             passed this record to TC.)
            Set DCF to the length of the RH plus the length of the RU.
           Send the HS_TO_PC_RECORD to the path control that the half-session
            uses.
        Else (not normal flow or send pacing count ≤ 0)
           Enqueue the BIU to the end of the pacing send queue (LOCAL.Q_PAC).

     When NO
        Incorporate the BIU into the PIU field of the HS_TO_PC_RECORD
         (see above):
        Send the HS_TO_PC_RECORD to the path control that the half-session uses.

Else (cryptography error)
   (The half-session router will cause the session to be terminated because
    LOCAL.SENSE_CODE is nonzero.)
```

## TC.TRY_TO_ENCIPHER

| | |
|---|---|
| FUNCTION: | Encipher a normal-flow request if necessary. |
| INPUT: | A BIU that includes a normal-flow request from TC.SEND; indicator of whether cryptography is active for the session; the cryptography session key and session seed (the technique for providing the cryptography session key and session seed is defined by the implementation). |
| OUTPUT: | If necessary, BIU is enciphered and padded, and EDI and PDI are set accordingly. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 6.0-6 |

```
If the RU category is FMD and the RU data length > 0
 and cryptography is active 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
       added 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 PDI to PD.
   Else
      Set 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.  Details of the DES algorithm
     are not formally specified in this book.
   If enciphering fails then
      Set LOCAL.SENSE_CODE to X'08480000' (cryptography function inoperative).
   Else
      Set EDI to ED.
```

## TC.RCV

| | |
|---|---|
| FUNCTION: | Receive message units sent to the half-session by PC. The usage and state checks are made. If the message unit contains a pacing response, it is processed. Requests and responses are routed and pacing requests are processed. Sequence numbers are processed. |
| INPUT: | A request or response BIU from the half-session router (see Chapter 6.0). |
| OUTPUT: | If no errors, DFC.RCV is called to process the BIU. If an error is encountered for CP-LU sessions, a negative response is generated. If an error is encountered for LU-LU sessions, LOCAL.SENSE_CODE is set to a nonzero value and the half-session router causes an UNBIND to be generated.<br><br>If the BIU is an Isolated Pacing Response (IPR) it is discarded. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| TC.RCV_CHECKS | page 6.2-16 |
| TC.RCV_NORM_RQ | page 6.2-17 |
| DFC_RCV | page 6.1-23 |
| SEND_NEG_RSP_OR_LOG | page 6.1-37 |
| LOCAL | page 6.0-6 |
| PC_TO_HS_RECORD | page A-23 |

```
This procedure has access to the PC_TO_HS_RECORD and to the LOCAL control block.
Extract the BIU from the PIU field of the PC_TO_HS_RECORD.

Call TC.RCV_CHECKS(BIU) (page 6.2-16) to check for
 errors in the received BIU.
If there is a receive check error (LOCAL.SENSE_CODE ≠ 0) then
   If this is an LU-LU session then
      (The nonzero setting of LOCAL.SENSE_CODE causes an UNBIND to terminate
      the session.)
   Else (CP_LU session)
      Call SEND_NEG_RSP_OR_LOG(BIU) (page 6.1-37) to send a negative
       response or to log the error.

Else (no receive-check errors)
   If send pacing is active then
      If RRI=RSP and PI=PAC then
         Call FSM_PAC_RQ_SEND(BIU) (page 6.2-20) to record the ability to send
          a pacing request for send pacing.
         Call TC.DEQUEUE_PAC (page 6.2-18) to remove BIUs from the
          send pacing queue (LOCAL.Q_PAC).

   If RRI=RSP and PI=PAC and DR1I≠DR1 and DR2I≠DR2 (it is an IPR) then
      Discard the IPR.
   Else (not IPR)
      If  EFI = NORMAL and RRI = RQ then
         Call TC.RCV_NORM_RQ(BIU) (page 6.2-17) to decipher the RU data (if necessary),
          update the receive pacing FSM, and increment the last received sequence
          number (LOCAL.SQN_RCV_CNT).
      If LOCAL.SENSE_CODE = 0 then
         Call DFC_RCV(BIU) (page 6.1-23) to pass the record to DFC.
      Else
         (The nonzero setting of LOCAL.SENSE_CODE causes an UNBIND to terminate
         an LU-LU session.)
```

## TC.RCV_CHECKS

| | |
|---|---|
| FUNCTION: | Usage checks are made for valid RU length and valid sequence number on a normal-flow request. If cryptography is to be used, an optional check is made that EDI is set when enciphering is mandatory, and the length of the RU is checked for being a multiple of 8. An optional check is made that the pacing protocol was not violated by the sender. The procedure verifies that all FSMs are in the proper state. |
| INPUT: | A request or response BIU from TC.RCV; indication whether maximum receive RU size is being enforced, and the maximum receive RU size (LOCAL.MAX_RCV_RU_SIZE) if so; indication whether sequence numbers are being used, and the last receive sequence number if so; indication whether receive pacing is active, and the receive pacing count (LOCAL.RCV_PACING_COUNT) if so; indication whether cryptography is active. |
| OUTPUT: | If a problem is found, LOCAL.SENSE_CODE is set to nonzero. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 6.0-6 |

```
If RRI = RQ and SDI = SD then
   Return with LOCAL.SENSE_CODE set to the sense code of the BIU.

If EFI = NORMAL then
   If a maximum receive RU size was specified at session activation and
    the length of the received RU > maximum receive RU size then
      Return with LOCAL.SENSE_CODE set to X'10020000' (RU length error).
   If RRI = RQ then
      If sequence numbers are being used then
         If SNF ≠ next expected sequence number (LOCAL.SQN_RCV_CNT + 1)
          (including consideration of the wrap case) then
            Return with LOCAL.SENSE_CODE set to X'20010000' (sequence number error).
      If PI = PAC but receive pacing is not active then
         Return with LOCAL.SENSE_CODE set to X'40080000' (pacing not supported).

      If cryptography is active and the RU category is
       FMD and the length of the RU > 0 then
         If EDI = ¬ED then
            Return with LOCAL.SENSE_CODE set to X'08090000' (mode inconsistency).
         Else (enciphered)
            If the RU data length is not an even multiple of 8 bytes then
               Return with  LOCAL.SENSE_CODE set to X'10010000' (RU data error).

      (The following is an optional check for pacing protocol violation)
      If receive pacing is active and the receive pacing count (LOCAL.RCV_PACING_COUNT)
       = 0 then
         Return with LOCAL.SENSE_CODE set to X'20110000' (pacing error).

If the RU category is network control or session control then
   Return with LOCAL.SENSE_CODE set to X'10070000' (category not supported).

Return with LOCAL.SENSE_CODE set to 0 (no errors).
```

## TC.RCV_NORM_RQ

| | |
|---|---|
| FUNCTION: | Decipher a normal-flow request if necessary, update receive pacing FSM, and increment sequence number. |
| INPUT: | Normal-flow request BIU; indication whether cryptography is active; indication whether sequence numbers are used, and, if so, the last received sequence number (LOCAL.SQN_RCV_CNT); indication whether receive pacing is active, and, if so, the receive pacing count (LOCAL.RCV_PACING_COUNT); the session cryptography key and seed. |
| OUTPUT: | Normal-flow request deciphered if input was enciphered. If sequence numbers are used, the sequence number is updated. If receive pacing is active, the receive pacing count is decremented. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_PAC_RQ_RCV | page 6.2-21 |
| LOCAL | page 6.0-6 |

```
If cryptography is active and the RU category is FMD and the RU data
 length > 0 then
   Execute the DES decipher algorithm.  Use the session key as the cryptography key.
    Use 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.  Details of the DES algorithm are not formally
    specified in this book.
   If deciphering is not successful then
      Set LOCAL.SENSE_CODE to X'08480000' (cryptography function inoperative).
      Log the error.
   Else (deciphering was successful)
      If PI = PAD then
         If the pad count is less than 1 or greater than 7 then
            Set LOCAL.SENSE_CODE to X'10010000', RU data error.
            Log the error.
         Else
            Eliminate the padding.  Set PI to ¬PAD.

If sequence numbers are being used then
   Increment the last received sequence number (LOCAL.SQN_RCV_CNT) by 1, including
    handling the wrap condition.

If receive pacing is active then
   Call FSM_PAC_RQ_RCV(BIU) (page 6.2-21) to record the ability to send a session
    pacing response for receive pacing.
   Decrement the receive pacing count (LOCAL.RCV_PACING_COUNT) by 1.
```

## TC.DEQUEUE_PAC

| | |
|---|---|
| FUNCTION: | A pacing response has been received so Q_PAC is now unlocked. Determine if it is valid to remove a BIU from Q_PAC. It is valid to remove a BIU from Q_PAC if it is a response or if it is a request and the pacing count is nonzero.<br><br>If valid, removes BIU from Q_PAC and sends it to path control. This procedure may cause the Pacing indicator in the BIU to be set to PAC. |
| INPUT: | LOCAL.Q_PAC has BIUs that were not sent earlier because a pacing response was outstanding; state of FSM_PAC_RQ_RCV; the send pacing count (LOCAL.SEND_PACING_COUNT). |
| OUTPUT: | HS_TO_PC_RECORD is sent to path control. |
| NOTE: | Procedure called only if LOCAL.SEND_PACING = YES. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_PAC_RQ_SEND | page 6.2-20 |
| FSM_PAC_RQ_RCV | page 6.2-21 |
| LOCAL | page 6.0-6 |
| HS_TO_PC_RECORD | page A-11 |

```
Do while LOCAL.Q_PAC is not empty and (send pacing count [LOCAL.SEND_PACING_COUNT] > 0
 or the top entry on  Q_PAC is a response)
   Remove the first enqueued BIU from Q_PAC.
   Select, based on the RRI of the removed BIU:

      When RRI = RQ
         CALL FSM_PAC_RQ_SEND(removed BIU) (page 6.2-20) to manipulate the PI
           in the BIU being sent and to manage send pacing states.
         Decrement the send pacing count (LOCAL.SEND_PACING_COUNT) by 1.

      When RRI = RSP
         If sufficient (implementation-dependent) resources exist and FSM_PAC_RQ_RCV
           (page 6.2-21) is in the PEND state and receive pacing is active then
             CALL FSM_PAC_RQ_RCV(removed BIU) (page 6.2-21) to manipulate
               the PI in the BIU being sent and to manage receive pacing states.

   Incorporate the BIU into the PIU field of the HS_TO_PC_RECORD (see page 6.2-13
    for details).
   (EFI, SNF, the RH, and the RU data were set up by the process that originally
    passed this record to TC.)
   Send the HS_TO_PC_RECORD to the path control that the half-session uses.
```

## TC.TRY_TO_SEND_IPR

| | |
|---|---|
| FUNCTION: | Determines if an ISOLATED PACING RESPONSE (IPR) may be sent, based on the state of FSM_PAC_RQ_RCV (page 6.2-21) and the availability of resources. If an IPR may be sent, the procedure generates an ISOLATED PACING RESPONSE (RH=X'830100') and sends it to path control. |
| INPUT: | An indication whether receive pacing is active for the session; state of the FSM PAC_RQ_RCV. |
| OUTPUT: | If an IPR is allowed, an ISOLATED PACING RESPONSE is sent to path control. |
| NOTE: | 1) An IPR is unique because it is the only response that can be sent with DR1I and DR2I off (response category = RQN). <br><br> 2) When an implementation sets the return code to NG, a method must be provided to insure that the half-session router will execute again when resources become available. Otherwise, the session could deadlock. <br><br> IPRs are needed to prevent deadlocks when no responses are being sent that can carry the pacing response or when the only available responses are blocked on the pacing queue. IPRs cannot be blocked on the pacing queue. <br><br> This routine is called periodically by the half-session router (see Chapter 6.0). |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| FSM_PAC_RQ_RCV | page 6.2-21 |
| LOCAL | page 6.0-6 |

```
If receive pacing is active and the state of FSM_PAC_RQ_RCV(BIU) (page 6.2-21)
 is PEND then
   If sufficient (implementation-dependent) resources exist then
      Create a BIU to contain the IPR.

      Set EFI to NORMAL or EXP (either normal or expedited flow is valid).

      Set SNF to some value (implementation dependent).  IPR is on the TC-TC flow,
       not the half-session to half-session flow, so is not related to the half-
       session send sequence number (LOCAL.SQN_SEND_CNT).

      Set DCF to the length of the RH plus the length of the RU.

      Set the RH to X'830100':
       (RSP, FMD, ¬FMH, ¬SD, BC, EC, RQN, POS, ¬DR1, ¬DR2, ¬QR, PAC,
       ¬BB, ¬CD, CODE0, ¬ED, ¬PD, ¬CEB).
      Set the RU to the null value.
      Call FSM_PAC_RQ_RCV(BIU) (page 6.2-21) to manage receive pacing.
      Incorporate the BIU into the PIU field of the HS_TO_PC_RECORD.
      Send the HS_TO_PC_RECORD to the path control that the half-session uses.
   Else
      NOTE: When the implementation does not have sufficient resources at this
      time, a method must be provided to insure that the half-session router
      will execute again when resources become available.  Otherwise, the session
      could deadlock.
```

FSM_PAC_RQ_SEND

| | |
|---|---|
| FUNCTION: | Records the ability to send a session-level pacing request for send pacing. RESET state indicates that a pacing request can be sent. AWAITING indicates that a pacing request has been sent but no pacing response has been received. |
| INPUT: | BIU; send pacing count (LOCAL.SEND_PACING_COUNT).<br><br>S, RQ, FIRST_IN_WINDOW means sending a BIU with RRI=RQ when the send pacing count equals the send window size.<br><br>S, RQ, ¬FIRST_IN_WINDOW means sending a BIU with RRI=RQ, when the send pacing count does not equal the send window size.<br><br>R, RSP, PAC means receiving a BIU with RRI=RSP and PI=PAC. |
| OUTPUT: | PI, LOCAL.SEND_PACING_COUNT. |
| NOTE: | FIRST_IN_WINDOW is TRUE when the pacing count equals the window size. This is never true when the FSM is in the AWAITING state because when the FSM enters the AWAITING state, the pacing count is set to 1 less than the window size. The pacing count is increased only when a pacing response is received, at which time the FSM returns to the RESET state. |

Referenced procedures, FSMs, and data structures:

LOCAL page 6.0-6

| INPUTS — STATE NAMES----> / STATE NUMBERS--> | RESET<br>01 | AWAITING<br>02 |
|---|---|---|
| S, RQ, FIRST_IN_WINDOW | 2(PACRQ) | /NOTE |
| S, RQ, ¬FIRST_IN_WINDOW | -(NOPAC) | -(NOPAC) |
| R, RSP, PAC | -(PACERR) | 1(PACRSP) |
| SIGNAL(RESET) | - | 1 |

| OUTPUT CODE | FUNCTION |
|---|---|
| PACRQ | Set PI to PAC. |
| NOPAC | Set PI to ¬PAC. |
| PACERR | Set PI to ¬PAC.<br>Log the unexpected pacing response that was received. |
| PACRSP | Increase the send pacing count (LOCAL.SEND_PACING_COUNT) by the value of one send window (specified at session activation). |

## FSM_PAC_RQ_RCV

| | |
|---|---|
| FUNCTION: | Records the ability to send a session pacing response for receive pacing. In RESET state, no pacing response is sent; in PEND state, it is.<br><br>In PEND state this half-session has resources to receive another window of BIUs but there has not been a response flowing on which to indicate the condition. The half-session is looking for a response to be sent.<br><br>When a pacing response is sent, the receive-pacing count is incremented by the receive-pacing window size. The receive-pacing count field is required only for an optional receive check. |
| INPUT: | Receive pacing count (LOCAL.RCV_PACING_COUNT).<br><br>R, RQ, PAC means receiving a BIU with RRI=RQ and PI=PAC.<br><br>S, RSP means sending a BIU with RRI=RSP. PI must be ¬PAC. |
| OUTPUT: | PI may be set to PAC or ¬PAC; LOCAL.RCV_PACING_COUNT may have been incremented by window size. |

Referenced procedures, FSMs, and data structures:

| | |
|---|---|
| LOCAL | page 6.0-6 |

| INPUTS | STATE NAMES----><br>STATE NUMBERS--> | RESET<br>01 | PEND<br>02 |
|---|---|---|---|
| R, RQ, PAC | | 2 | -(PACERR) |
| S, RSP | | - | 1(PAC) |
| SIGNAL(RESET) | | - | 1 |

| OUTPUT CODE | FUNCTION |
|---|---|
| PAC | Set PI to PAC.<br>Increase the receive pacing count (LOCAL.RCV_PACING_COUNT) by the value of one receive window (specified at session activation). |
| PACERR | Set PI to ¬PAC.<br>Log the unexpected pacing request that was received. |

# APPENDIX A. NODE DATA STRUCTURES

This appendix contains the shared data structures for LU 6.2.

CPLU_CB

The CP-LU control block represents an active session between this LU and a control point (SSCP or PNCP).

```
CPLU_CB
   CP_ID:  control point identifier (see page A-2)
   PC_ID:  identifier of path control being used by this CP-LU session
   HS_ID:  identifier of the CP-LU half-session
```

LUCB

The LUCB_LIST contains information about LUs. There is one LUCB_LIST per node and one LUCB per 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. Fully-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 "Appendix E. Request-Response Unit (RU) 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 have to know their own names.
3. The FULLY_QUALIFIED_LU_NAME contains no trailing blanks.

LUCB

Shared Data

```
LU_ID:  identifier of the local LU
FULLY_QUALIFIED_LU_NAME (see Notes)
PARTNER_LU_LIST (see page A-2)
```

Data Unique to PS.COPR

```
LU_SESSION_LIMIT:  maximum number of LU-LU sessions the local LU can have
```

CP_ID

The CP_ID structure is the unique control point (e.g., SSCP, PNCP) identifier.

CP_ID
  Subarea node contents:
    CP_NETWORK_ADDRESS: full network address of control point
  Peripheral node contents:
    ALS: adjacent link station that control point is using for CP-LU and CP-PU sessions

PARTNER_LU

The PARTNER_LU_LIST is a list contained within each LUCB entry. There is one PARTNER_LU_LIST per LU and one PARTNER_LU entry for each LU name known by a given 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 LU knows another LU and is not sent outside the 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 LU. This allows for cases when the remote transaction program is located in the same LU as the local program.

2. Local LU names consist of type-G symbol strings. Fully-qualified LU names consist of type-A symbol strings. See "Appendix E. Request-Response Unit (RU) 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, FULLY_QUALIFIED_LU_NAME, and UNINTERPRETED_LU_NAME fields contain no trailing blanks.

PARTNER_LU

Shared Data

LOCAL_LU_NAME (see Notes 1, 2, and 4)
FULLY_QUALIFIED_LU_NAME (see Notes 2, 3, and 4)
UNINTERPRETED_LU_NAME (see Note 4)
MODE_LIST (see page A-3)
SESSION_LIMIT: maximum number of sessions that the local LU can have with the partner LU

## MODE

The MODE_LIST is a list contained within each PARTNER_LU entry. There is one MODE entry 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 "Appendix E. Request-Response Unit (RU) Formats" for symbol-string definitions.

4. TERMINATION_COUNT is the count of the number of sessions that this 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 the LU resources manager 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 the resources manager receives SESSION_ACTIVATED or +ACTIVATE_SESSION_RSP to the time that the resources manager 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 the resources manager sends ACTIVATE_SESSION to the time that the resources manager receives ACTIVATE_SESSION_RSP. PENDING_SESSION_COUNT is the sum of PENDING_CONWINNERS_COUNT and PENDING_CONLOSERS_COUNT.

## 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

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 LU Resources Manager

TERMINATION_COUNT (see Note 4)
PENDING_TERMINATION_CONWINNERS
PENDING_TERMINATION_CONLOSERS
SINGLE_SESSION_POLARITY: possible values: FIRST_SPEAKER, BIDDER

TRANSACTION_PROGRAM

Each LUCB contains a TRANSACTION_PROGRAM_LIST. This list contains one entry for each transaction program known at the 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: Transaction program names consist of type-AE up through type-GR symbol strings, depending upon the implementation. See "Appendix E. Request-Response Unit (RU) Formats" for symbol-string definitions.

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
RESOURCES_SUPPORTED_LIST: possible values: BASIC_CONVERSATION, MAPPED_CONVERSATIN

Data Unique to PS.INITIALIZE

NUMBER_OF_PIP_SUBFIELDS

Data Unique to RM

SYNC_LEVELS_SUPPORTED_LIST: possible values: NONE, CONFIRM, SYNCPT

Data Unique to PS.MC

MC_FUNCTIONS_SUPPORTED_LIST: possible values: MAPPING, FMH_DATA

## LULU_CB

The LU-LU session control block is used by LU network services (LNS) to keep information about an LU-LU session. There is one LULU_CB 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).

```
CP_LU:  contains information pertaining to CP_LU session
   CP_ID:  control point identifier (see page A-2)
   HS_ID:  identifier for CP-LU half-session
LUNAME:  contains local and fully qualified target LU names
MODENAME:  mode name for this LU-LU session
SESSION_ID:  session instance identifier
SESSION_INFORMATION
   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
LU_LU:  information pertaining to the LU-LU session
```

PC_ID—path control identifier representing the path control process being used by this LU-LU session. This field is set when a BIND request or PC_CONNECT_RSP is received.

```
   PC_ID
```

ALS—adjacent link station identifier representing the link this LU-LU session is using. This field is set when a BIND request is received or a PC_CONNECT is sent. It is used only in peripheral nodes.

```
   ALS
```

ADDRESS—the addresses of the LUs for this LU-LU session. For subarea nodes this field is set when a CINIT or BIND request is received. For peripheral nodes it is set when a BIND request or PC_CONNECT_RSP is received.

```
   ADDRESS (see page A-33)
```

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

SENT_INITIATE_RQ fields are set when an INIT-SELF request is sent.

SENT_INITIATE_RQ
URC: used to correlate future CINIT or BIND request.
SNF: TH sequence number of sent INIT-SELF request (used to correlate INIT-SELF response).

SENT_BIND_RQ fields are set when a BIND request is sent. A copy of the sent BIND request RU is saved because it is needed to perform error checking on the received BIND response.

SENT_BIND_RQ
SNF: TH sequence number of sent BIND request (used to correlate BIND resonse)
BIND_RQ_RU: saved BIND request RU

SENT_UNBIND_RQ fields are set when an UNBIND request is sent.

SENT_UNBIND_RQ
SNF: TH sequence number of sent UNBIND request (used to correlate UNBIND response)

RCB

The RCB_LIST contains information about resources. There is one RCB_LIST per 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 LU knows another LU and is not sent outside the LU. The maximum length of the LU_NAME is implementation-defined, but is shown here as having a maximum length of 17 characters.

2. LU names consist of type-G symbol strings. Mode names consist of type A symbol strings. See "Appendix E. Request-Response Unit (RU) Formats" for symbol string definitions.

3. When the resources manager receives a GET_SESSION (page A-26) 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.

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
LU_NAME: Partner LU name (see Notes 1 and 2)
MODE_NAME (see Note 2)
CONVERSATION_TYPE: possible values: BASIC_CONVERSATION, MAPPED_CONVERSATION
FSM_CONVERSATION page 5.1-59

Data Unique to RM

SESSION_PARMS_PTR (see Note 3)

Data Unique to PS.CONV

PS_TO_HS_RECORD, see SEND_DATA_RECORD page A-24
SEND_LL_REMAINDER: number of bytes remaining to be sent in the outgoing logical record
RECEIVE_LL_REMAINDER: number of bytes remaining to be received in the incoming logical record
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_LL_BYTE: possible values: PRESENT, NOT_PRESENT
SAVED_BYTE: retains SEND_LL_BYTE (reserved when SEND_LL_BYTE=NOT_PRESENT)
MAX_BUFFER_LENGTH: maximum number of bytes in outgoing logical record or buffer
SYNC_LEVEL: possible values: NONE, CONFIRM, SYNCPT
RQ_TO_SEND_RCVD: possible values: YES, NO
FSM_ERROR_OR_FAILURE page 5.1-61
FSM_POST page 5.1-62
HS_TO_PS_BUFFER_LIST: List of BUFFER_ELEMENT (see page A-8)

Data Unique to PS.MC

MC_RECEIVE_BUFFER: Contains RECEIVED_INFO (see page A-8)
MAPPER_SAVE_AREA
MC_MAX_SEND_SIZE: maximum number of bytes in a mapped-conversation logical record

BUFFER_ELEMENT

BUFFER_ELEMENT is the structure that is inserted into the HS_TO_PS_BUFFER_LIST. The HS_TO_PS_BUFFER_LIST is contained within an RCB and consists of information received by PS.CONV ("Chapter 5.1. Presentation Services--Conversation Verbs") from the half-session but not yet passed to the transaction program.

BUFFER_ELEMENT:
TYPE: possible values: DATA, FMH7, CONFIRM, PREPARE_TO_RCV_FLUSH, PREPARE_TO_RCV_CONFIRM, DEALLOCATE_FLUSH, DEALLOCATE_CONFIRM
DATA (reserved when TYPE≠DATA, FMH7)

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 LU knows another LU and is not sent outside the LU. The maximum length of the LU_NAME is implementation-defined, but is shown here as having a maximum length of 17 characters.

2. LU names consist of type-G symbol strings. Fully-qualified LU names and mode names consist of type-A symbol strings. See "Appendix E. Request-Response Unit (RU) Formats" for symbol-string definitions.

SCB

Shared Data

HS_ID: unique SCB identifier
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
FULLY_QUALIFIED_LU_NAME: Partner LU name (see Note 2)

## TCB

The TCB_LIST contains information about active transaction program instances. There is one TCB_LIST per 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. Transaction program names and access security information subfield consist of type-AE up through type-GR symbol strings, depending upon the implementation. See "Appendix E. Request-Response Unit (RU) Formats" for symbol-string definitions.

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**

```
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
ACCESS_SECURITY_INFORMATION (see Note 1)
   TYPE_0:  contains profile
   TYPE_1:  contains password
   TYPE_2:  contains user ID
```

## HS_TO_LNS_RECORD

HS_TO_LNS_RECORD is a record sent by the half-session (HS) to LU network services (LNS).

HS_TO_LNS_RECORD: contains INIT_HS_RSP, HS_RCV_RECORD, or ABORT_HS record (see below)

ABORT_HS

ABORT_HS indicates to LU network services that the half-session has found a severe error and cannot continue processing. This will cause an UNBIND request to be sent for the aborted half-session.

NOTE: This record is sent only by LU-LU half-sessions.

ABORT_HS
  HS_ID: identifies the half-session sending this record
  SENSE_CODE: indicates the reason the half-session aborted

HS_RCV_RECORD

This record contains PIU information pertaining to FMD NS RUs that flow from the control point to the LU on the LU-CP session (e.g., CINIT, NOTIFY).

NOTE: This record is sent only by LU-CP half-sessions.

HS_RCV_RECORD
  HS_ID: identifies the half-session sending this record
  PIU (see page A-35)

INIT_HS_RSP

This record is a response to the INIT_HS record that was sent from LU network services (LNS) to the half-session (HS) 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

HS_TO_PC_RECORD

HS_TO_PC_RECORD is a record sent by the half-session (HS) to path control (PC). It contains the sending half-session's process identifier (many half-sessions may send to the same path control) and PIU information from which path control will build and send a PIU.

HS_TO_PC_RECORD
  HS_ID: identifier of the half-session sending this record.
  PIU: contains path information unit (see page A-35)

HS_TO_PS_RECORD

The HS_TO_PS_RECORD is the record that HS ("Chapter 6.0. Half-Session") sends to PS_CONV ("Chapter 5.1. Presentation Services--Conversation Verbs").

HS_TO_PS_RECORD: contains a RECEIVE_DATA, REQUEST_TO_SEND, RSP_TO_REQUEST_TO_SEND, RECEIVE_ERROR, or CONFIRMED record (see below)

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. Confirmation is requested when SEND_PARM.TYPE (page A-35) = CONFIRM, DEALLOCATE_CONFIRM, PREPARE_TO_RCV_CONFIRM_SHORT, or PREPARE_TO_RCV_CONFIRM_LONG.

CONFIRMED
    HS_ID: identifies the half-session sending this record

RECEIVE_DATA

RECEIVE_DATA is sent by the half-session to PS_CONV to inform PS_CONV of receipt of conversation data. The data is passed to PS_CONV in the DATA field. If FMH = YES, the DATA contains an FMH-7.

RECEIVE_DATA
    HS_ID: identifies the half-session sending this record
    FMH: possible values: YES, NO (If FMH=YES, DATA contains an FMH-7.)
    TYPE: possible values: NOT_END_OF_DATA, CONFIRM, PREPARE_TO_RCV_CONFIRM, PREPARE_TO_RCV_FLUSH, DEALLOCATE_CONFIRM, DEALLOCATE_FLUSH
    DATA: data received from partner transaction program

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
    HS_ID: identifies the half-session sending this record

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
    HS_ID:  identifies the half-session sending this record
```

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 REQUEST_TO_SEND record (page A-26) has been received.

```
RSP_TO_REQUEST_TO_SEND
    HS_ID:  identifies the half-session sending this record
```

HS_TO_RM_RECORD

The HS_TO_RM_RECORD is the record that HS ("Chapter 6.0. Half-Session") sends to RM ("Chapter 3. LU Resources Manager").

```
HS_TO_RM_RECORD:  contains ATTACH_HEADER, BID, BID_RSP, FREE_SESSION,
 BIS_RQ, BIS_REPLY, RTR_RQ, or RTR_RSP record (see below)
```

ATTACH_HEADER

ATTACH_HEADER is sent by the half-session to the resources manager to inform the resources manager of the receipt of an FMH-5 on the half-session. The HEADER field contains the FMH-5.

```
ATTACH_HEADER
   HS_ID:  identifies the half-session sending this record
   HEADER:  contains the received FMH-5
```

## 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 will reply with a BID_RSP record (page A-28). The half-session will send 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-29] and BID_WITH_ATTACH [page A-28]). 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
  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
  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
  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
   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
   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)
```

LNS_TO_HS_RECORD

LNS_TO_HS_RECORD is a record sent by LU network services (LNS) to the half-session (HS).

```
LNS_TO_HS_RECORD:  contains HS_SEND_RECORD or INIT_HS record (see below)
```

HS_SEND_RECORD

This record contains PIU information pertaining to FMD NS RUs that flow from the LU to the control point on the LU-CP session (e.g., INIT-SELF, SESSST).

NOTE: This record is sent only to LU-CP half-sessions.

```
HS_SEND_RECORD
   PIU (see page A-35)
```

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 activation RUs (e.g., BIND, ACTLU). This is the first record received by the half-session after its creation.

```
INIT_HS
   PC_ID:  identifies the path control the half-session communicates with
   TYPE of half-session:  possible values:  PRI, SEC
   DATA_TYPE:  specifies whether the DATA contains ACTLU image or BIND image
   DATA:  contains either ACTLU image or BIND image
      ACTLU_IMAGE:  contains fields associated with activating an LU-CP half-session
         FM_PROFILE (see ACTLU in Appendix E)
         TS_PROFILE (see ACTLU in Appendix E)
         MAX_RU_SIZE:  maximum RU size to be used on the LU-CP session
      BIND_IMAGE:  fields associated with activating an LU-LU half-session
       (see BIND request in Appendix E)
```

LNS_TO_NNM_RECORD

LNS_TO_NNM_RECORD is a record sent by LU network services (LNS) to the nodal NAU manager (NNM).

```
LNS_TO_NNM_RECORD:  contains BIND_RQ_SEND_RECORD, BIND_RSP_SEND_RECORD,
 UNBIND_RQ_SEND_RECORD, UNBIND_RSP_SEND_RECORD, ACTLU_RSP_SEND_RECORD,
 DACTLU_RSP_SEND_RECORD, PC_CONNECT, HIERARCHICAL_RESET_RSP,
 or PC_HS_CONNECT record (see below)
```

## ACTLU_RSP_SEND_RECORD

This record contains information for an ACTLU response PIU that is to be sent.

```
ACTLU_RSP_SEND_RECORD
   LU_ID:  process identifier of the sending LU
   PC_ID:  process identifier of path control to be sent to
   ADDRESS:  contains TH address fields (see page A-33)
   PIU:  contains ACTLU response (see page A-35)
```

## BIND_RQ_SEND_RECORD

This record contains information for a BIND request PIU that is to be sent.

```
BIND_RQ_SEND_RECORD
   LU_ID:  process identifier of the sending LU
   PC_ID:  process identifier of path control to be sent to
   ADDRESS:  contains TH address fields (see page A-33)
   PIU:  contains BIND request (see page A-35)
```

## BIND_RSP_SEND_RECORD

This record contains information for a BIND response PIU that is to be sent.

```
BIND_RSP_SEND_RECORD
   LU_ID:  process identifier of the sending LU
   PC_ID:  process identifier of path control to be sent to
   ADDRESS:  contains TH address fields (see page A-33)
   PIU:  contains BIND response (see page A-35)
```

## DACTLU_RSP_SEND_RECORD

This record contains information for a DACTLU response PIU that is to be sent.

```
DACTLU_RSP_SEND_RECORD
   LU_ID:  process identifier of the sending LU
   PC_ID:  process identifier of path control to be sent to
   ADDRESS:  contains TH address fields (see page A-33)
   PIU:  contains DACTLU response (see page A-35)
```

HIERARCHICAL_RESET_RSP

This record is a response to a HIERARCHICAL_RESET record. It indicates that hierarchical reset processing is complete.

```
HIERARCHICAL_RESET_RSP
   LU_ID:  process identifier of the sending LU
   PC_ID:  process identifier of path control to be sent to
   CP_ID:  control point identifier (see page A-2)
```

PC_CONNECT

This record is used, by primary LUs, to request information about a path control that will be used to activate an LU-LU session. For peripheral nodes, the adjacent link station (ALS) is used to specify the path to be used. For subarea nodes, the subarea address for the secondary LU (SUBAREA_ADDRESS) and path information (PATH_INFORMATION) are used to specify the path. Path information includes the class-of-service name and the virtual route identifier list.

```
PC_CONNECT
   LU_ID:  process identifier of the sending LU
   HS_ID:  half-session process identifier used to correlate the PC_CONNECT_RSP
   TYPE of node this PLU resides in:  possible values:  PERIPHERAL, SUBAREA
   ALS:  adjacent link station (reserved when TYPE=SUBAREA)
   SUBAREA_ADDRESS:  for SLU needed to assign route (reserved when TYPE=PERIPHERAL)
   PATH_INFORMATION:  contains class-of-service and virtual-route-identifier list
    (reserved when TYPE=PERIPHERAL)
```

PC_HS_CONNECT

This record is used to notify a path control that it can now send to, and receive from, a newly activated half-session.

```
PC_HS_CONNECT
   LU_ID:  process identifier of the sending LU
   PC_ID:  process identifier of path control to be sent to
   HS_ID:  half-session process identifier
   ADDRESS:  contains TH address fields (see page A-33)
```

PC_HS_DISCONNECT

This record is used to notify a path control that a half-session has been deactivated. Path control will stop sending to, and receiving from, the half-session.

```
PC_HS_DISCONNECT
   LU_ID:  process identifier of the sending LU
   HS_ID:  process identifier of half-session being deactivated
```

UNBIND_RQ_SEND_RECORD

This record contains information for an UNBIND request PIU that is to be sent.

```
UNBIND_RQ_SEND_RECORD
   LU_ID:  process identifier of the sending LU
   PC_ID:  process identifier of path control to be sent to
   ADDRESS:  contains TH address fields (see page A-33)
   PIU:  contains UNBIND request (see page A-35)
```

UNBIND_RSP_SEND_RECORD

This record contains information for an UNBIND response PIU that is to be sent.

```
UNBIND_RSP_SEND_RECORD
   LU_ID:  process identifier of the sending LU
   PC_ID:  process identifier of path control to be sent to
   ADDRESS:  contains TH address fields (see page A-33)
   PIU:  contains UNBIND response (see page A-35)
```

LNS_TO_RM_RECORD

The LNS_TO_RM_RECORD is the record that LNS ("Chapter 4. LU Network Services") sends to RM ("Chapter 3. LU Resources Manager").

```
LNS_TO_RM_RECORD:  contains ACTIVATE_SESSION_RSP, SESSION_ACTIVATED,
 SESSION_DEACTIVATED, or CTERM_DEACTIVATE_SESSION record (see below)
```

## ACTIVATE_SESSION_RSP

ACTIVATE_SESSION_RSP is sent by LU network services to the resources manager in reply to an ACTIVATE_SESSION record (page A-31). ACTIVATE_SESSION_RSP records need not be sent in the same order as the 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 contains 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-31)
   TYPE of response:  possible values:  POS, NEG
   SESSION_INFORMATION (reserved when TYPE=NEG—see page A-35)
   ERROR_TYPE:  possible values:  RETRY, NO_RETRY (reserved when TYPE=POS)
```

## CTERM_DEACTIVATE_SESSION

CTERM_DEACTIVATE_SESSION is sent by LU network services to the resources manager to request normal shutdown (i.e., BIS exchange followed by DEACTIVATE_SESSION) of the session identified by HS_ID.

```
CTERM_DEACTIVATE_SESSION
   HS_ID:  identifier of the half-session to be shut down
```

## SESSION_ACTIVATED

SESSION_ACTIVATED is sent by LU network services 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 given in SESSION_INFORMATION.

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 LU knows another LU and is not sent outside the 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 "Appendix E. Request-Response Unit (RU) Formats" for symbol-string definitions.

```
SESSION_ACTIVATED
   SESSION_INFORMATION (see page A-35)
   LU_NAME (see Notes 1 and 2)
   MODE_NAME (see Note 2)
```

## SESSION_DEACTIVATED

SESSION_DEACTIVATED is sent by LU network services to the resources manager to notify the resources manager that the session identified by HS_ID has been deactivated by the partner LU.

```
SESSION_DEACTIVATED
   HS_ID:  identifies half-session that was deactivated
   REASON for deactivation:  possible values:  NORMAL, ABNORMAL_RETRY, ABNORMAL_NO_RETRY
```

## NNM_TO_LNS_RECORD

NNM_TO_LNS_RECORD is a record sent by the nodal NAU manager (NNM) to LU network services (LNS).

```
NNM_TO_LNS_RECORD:  contains BIND_RQ_RCV_RECORD, BIND_RSP_RCV_RECORD,
 UNBIND_RQ_RCV_RECORD, UNBIND_RSP_RCV_RECORD, ACTLU_RQ_RCV_RECORD,
 DACTLU_RQ_RCV_RECORD, PC_CONNECT_RSP, SESSION_ROUTE_INOP, or
 HIERARCHICAL_RESET record (see below)
```

## ACTLU_RQ_RCV_RECORD

This record contains information about a received ACTLU request PIU.

```
ACTLU_RQ_RCV_RECORD
   PC_ID:  process identifier of path control that received this PIU
   ADDRESS:  contains TH address fields (see page A-33)
   CP_ID:  control point identifier (see page A-2)
   PIU:  contains ACTLU request (see page A-35)
```

## BIND_RQ_RCV_RECORD

This record contains information about a received BIND request PIU and information about the path control that received it.

```
BIND_RQ_RCV_RECORD
   PC_ID:  process identifier of path control that received this PIU
   ADDRESS:  contains TH address fields (see page A-33)
   PC_CHARACTERISTICS:  path control characteristics (see page A-34)
   PIU:  contains BIND request (see page A-35)
```

BIND_RSP_RCV_RECORD

This record contains information about a received BIND response PIU.

```
BIND_RSP_RCV_RECORD
   PC_ID:  process identifier of path control that received this PIU
   ADDRESS:  contains TH address fields (see page A-33)
   PIU:  contains BIND response (see page A-35)
```

DACTLU_RQ_RCV_RECORD

This record contains information about a received DACTLU request PIU.

```
DACTLU_RQ_RCV_RECORD
   PC_ID:  process identifier of path control that received this PIU
   ADDRESS:  contains TH address fields (see page A-33)
   CP_ID:  control point identifier (see page A-2)
   PIU:  contains DACTLU request (see page A-35)
```

HIERARCHICAL_RESET

This record is used to reset all sessions with respect to a specific control point (e.g., SSCP) session. It contains the identifier of the control point affected (CP_ID) and path control process identifier associated with that control point (PC_ID).

```
HIERARCHICAL_RESET
   PC_ID:  path control process identifier associated with the CP-LU session
   CP_ID:  control point identifier (see page A-2)
```

PC_CONNECT_RSP

This record is a response to a PC_CONNECT record sent by LU network services (LNS). If positive, it contains information about the path control (PC_ID and PC_CHARACTERISTICS) that will be used for the LU-LU session being activated. For peripheral nodes, it also contains an assigned address (ADDRESS) for the LU-LU session.

```
PC_CONNECT_RSP
   HS_ID:  half-session process identifier used to correlate to PC_CONNECT record
   TYPE of response:  possible values:  POS, NEG
   PC_ID:  process identifier of path control that received this PIU
    (reserved when TYPE=NEG)
   ADDRESS:  contains TH address fields (for peripheral node—see page A-33)
   PC_CHARACTERISTICS:  path control characteristics (reserved when TYPE=NEG—see page A-34)
   SENSE_CODE:  indicating the type of error (reserved when TYPE=POS)
```

**SESSION_ROUTE_INOP**

This record indicates that a route, represented by a path control process, has become inoperative.

```
SESSION_ROUTE_INOP
   PC_ID:  process identifier of path control process that has become inoperative
```

**UNBIND_RQ_RCV_RECORD**

This record contains information about a received UNBIND request PIU.

```
UNBIND_RQ_RCV_RECORD
   PC_ID:  process identifier of path control that received this PIU
   ADDRESS:  contains TH address fields (see page A-33)
   PIU:  contains UNBIND request (see page A-35)
```

**UNBIND_RSP_RCV_RECORD**

This record contains information about a received UNBIND response PIU.

```
UNBIND_RSP_RCV_RECORD
   PC_ID:  process identifier of path control that received this PIU
   ADDRESS:  contains TH address fields (see page A-33)
   PIU:  contains UNBIND response (see page A-35)
```

**PC_TO_HS_RECORD**

PC_TO_HS_RECORD is a record sent by path control (PC) to the half-session (HS). It contains PIU information that path control obtained from a received PIU.

```
PC_TO_HS_RECORD
   PIU (see page A-35)
```

PS_TO_HS_RECORD

The PS_TO_HS_RECORD is the record that PS_CONV ("Chapter 5.1. Presentation Services--Conversation Verbs") sends to HS ("Chapter 6.0. Half-Session").

```
PS_TO_HS_RECORD:  contains SEND_DATA_RECORD, SEND_ERROR, REQUEST_TO_SEND,
 or CONFIRMED record (see below)
```

CONFIRMED

CONFIRMED is sent by PS_CONV to the half-session to request the half-session to send a positive response to a previous request for confirmation by the partner transaction program.

```
CONFIRMED
```

REQUEST_TO_SEND

REQUEST_TO_SEND is sent by PS_CONV to the half-session to request the half-session to send a SIGNAL(SOFT). SIGNAL(SOFT) is used to request permission to enter the send state for the conversation.

```
REQUEST_TO_SEND
```

SEND_DATA_RECORD

SEND_DATA is sent by PS_CONV to the half-session to request the half-session to send conversation data.

```
SEND_DATA_RECORD
   SEND_PARM (see page A-35)
```

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
```

PS_TO_RM_RECORD

The PS_TO_RM_RECORD is the record that presentation services (i.e., PS.CONV ["Chapter 5.1. Presentation Services--Conversation Verbs"], PS.INITIALIZE ["Chapter 5.0. Overview of Presentation Services"], or PS.COPR ["Chapter 5.4. Presentation Services--Control-Operator Verbs"]) sends to RM ("Chapter 3. LU Resources Manager") to request that a certain function be performed.

```
PS_TO_RM_RECORD:  contains ALLOCATE_RCB, GET_SESSION, DEALLOCATE_RCB, TERMINATE_PS,
 CHANGE_SESSIONS, UNBIND_PROTOCOL_ERROR, RM_ACTIVATE_SESSION, or
 RM_DEACTIVATE_SESSION record (see below)
```

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 will also attempt to reserve a first-speaker session if IMMEDIATE_SESSION = YES. The resources manager will reply to the ALLOCATE_RCB with an RCB_ALLOCATED record (page A-32).

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 LU knows another LU and is not sent outside the LU. The maximum length of the LU_NAME is implementation-defined, but is shown here as having a maximum length of 17 characters.

2. LU names consist of type-G symbol strings. Mode names consist of type-A symbol strings. See "Appendix E. Request-Response Unit (RU) 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
```

## 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 LU knows another LU and is not sent outside the 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 "Appendix E. Request-Response Unit (RU) 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 will reply to the DEALLOCATE_RCB with an RCB_DEALLOCATED record (page A-32).

```
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 will reply to the GET_SESSION with a SESSION_ALLOCATED record (page A-33).

```
GET_SESSION
   TCB_ID:  ID of the PS process that sent GET_SESSION
   RCB_ID:  ID of the conversation
   BID_INDICATOR:  possible values:  ATTACH, NO_ATTACH
```

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 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 LU knows another LU and is not sent outside the LU. The maximum length of the LU_NAME is implementation-defined, but is shown here as having a maximum length of 17 characters.

2. LU names consist of type-G symbol strings. Mode names consist of type-A symbol strings. See "Appendix E. Request-Response Unit (RU) Formats" 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
```

RM_TO_HS_RECORD

The RM_TO_HS_RECORD is the record that RM ("Chapter 3. LU Resources Manager") sends to HS ("Chapter 6.0. Half-Session").

```
RM_TO_HS_RECORD: contains BID_WITHOUT_ATTACH, BID_RSP, BID_WITH_ATTACH, BIS_REPLY,
 HS_PS_CONNECTED, BIS_RQ, YIELD_SESSION, RTR_RQ, or RTR_RSP record (see below).
```

BID_RSP

BID_RSP is sent by the resources manager to the half-session in response to a previous BID record (page A-14) 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
   RTI:  possible values:  POS, NEG
   SENSE_CODE (reserved when RTI=POS)
```

BID_WITH_ATTACH

BID_WITH_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 accompanied by conversation data (including the FMH-5 that will attach the remote transaction program) in the SEND_PARM structure (page A-35). The resources manager will send BID_WITH_ATTACH if this LU is the first speaker or the bidder. When bidding for a session, the resources manager chooses between BID_WITHOUT_ATTACH and BID_WITH_ATTACH on the basis of the BID_INDICATOR field in the GET_SESSION (page A-26) from PS_CONV. If this LU is the bidder, the half-session will inform the resources manager of the partner LU's response with a BID_RSP record (page A-14).

```
BID_WITH_ATTACH
   SEND_PARM (see page A-35)
```

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 will send 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 will inform the resources manager of the partner LU's response with a BID_RSP record (page A-14).

BID_WITHOUT_ATTACH

BIS_REPLY

BIS_REPLY is sent by the resources manager to the half-session to request the half-session to send a BIS(RQE3) request unit.

BIS_REPLY

BIS_RQ

BIS_RQ is sent by the resources manager to the half-session to request the half session to send a BIS(RQE1) request unit.

BIS_RQ

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 allocation of a session to a conversation.

```
HS_PS_CONNECTED
   PS_ID:  ID of presentation services process
```

RTR_RQ

RTR_RQ is sent by the resources manager to the half-session to request the half-session to send an RTR request unit.

RTR_RQ

RTR_RSP

RTR_RSP is 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
   RTI:  possible values:  POS, NEG
   SENSE_CODE (reserved when RTI=POS)
```

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-brackets" 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-26), it will send YIELD_SESSION to the half-session; the half-session then reverts to contention state.

YIELD_SESSION

RM_TO_LNS_RECORD

The RM_TO_LNS_RECORD is the record that RM ("Chapter 3. LU Resources Manager") sends to LNS ("Chapter 4. LU Network Services").

RM_TO_LNS_RECORD:  contains ACTIVATE_SESSION or DEACTIVATE_SESSION record (see below)

## ACTIVATE_SESSION

ACTIVATE_SESSION is sent by the resources manager to LU network services 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. LU network services will reply to ACTIVATE_SESSION with an ACTIVATE_SESSION_RSP record (page A-20) 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 LU knows another LU and is not sent outside the 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 "Appendix E. Request-Response Unit (RU) Formats" 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 LU network services 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 (reserved when TYPE≠ABNORMAL)
```

## RM_TO_PS_RECORD

The RM_TO_PS_RECORD is the record that RM ("Chapter 3. LU Resources Manager") sends to PS_INITIALIZE ("Chapter 5.0. Overview of Presentation Services") or PS_CONV ("Chapter 5.1. Presentation Services--Conversation Verbs").

```
RM_TO_PS_RECORD: contains ATTACH_RECEIVED, RCB_DEALLOCATED, RM_SESSION_ACTIVATED,
 or CONVERSATION_FAILURE record (see below).
```

ATTACH_RECEIVED

ATTACH_RECEIVED is sent by the resources manager to PS_INITIALIZE in a newly created PS process (created as the result of an Attach FMH-5). TCB_ID is the ID of the transaction control block, RCB_ID is the ID of the initial resource control block, and FMH_5 is the FMH-5 that initiated the new presentation services process. The resources manager performs some validity checks on the FMH-5 before passing it to presentation services. SENSE_CODE indicates the result of these checks.

```
ATTACH_RECEIVED
   TCB_ID:  ID of transaction control block
   RCB_ID:  ID of resource control block
   SENSE_CODE
   FMH_5:  Attach FMH-5 header (see Appendix H)
```

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. The REASON field assumes only the values SON or PROTOCOL_VIOLATION.

```
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-25). 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)
```

RCB_DEALLOCATED

RCB_DEALLOCATED is sent by the resources manager to PS_CONV in reply to a DEALLOCATE_RCB record (page A-26).

```
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-27). 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-26). RETURN_CODE indicates the success or failure of the session allocation.

```
SESSION_ALLOCATED
   RETURN_CODE:  possible values:  OK, UNSUCCESSFUL_RETRY, UNSUCCESSFUL_NO_RETRY,
```

## CRV_RQ_RU

```
CRV_RQ_RU
    RQ_CODE:  possible values:  X'C0' (signifying CRV)
    CRYPTO_SEED
```

## ADDRESS

ADDRESS contains TH addresses. For subarea nodes they are 6-byte network addresses. For peripheral nodes, they are the local 1-byte representations for network addresses plus the ODAI field.

```
ADDRESS
   Subarea address structure:
      THIS_NAU:  address of the local NAU (contains 32-bit subarea and 16-bit
       element address)
      OTHER_NAU:  address of the partner NAU (contains 32-bit subarea and 16-bit
       element address)
   Peripheral node structure:
      ODAI:  origin/destination assignment indicator
      THIS_NAU:  8-bit address representing the local NAU
      OTHER_NAU:  8-bit address representing the partner NAU
```

BIU

This record is used only by the half-session (HS) process. It contains information about TH, RH, and RU fields.

BIU: same as PIU (see page A-35)

PC_CHARACTERISTICS

PC_CHARACTERISTICS: path control characteristics

PATH_CONTROL_TYPE—PEER means that this path control is being used for a PNCP-mediated session; BACKBONE means that this path control is being used for an SSCP-mediated session.

PATH_CONTROL_TYPE: possible values: PEER, BACKBONE

ALS—adjacent link station address associated with this path control. This field is used only by peripheral nodes.

ALS

SEGMENTING—path control segmenting capability. This applies to both send and receive segmenting. Either both are supported or both are not supported.

SEGMENTING: possible values: SUPPORTED, NOT_SUPPORTED

MAX_RU_SEGMENT_SIZE—the maximum number of RU bytes that may be sent or received by this path control. This value is independent of path control's segmenting capability.

MAX_RU_SEGMENT_SIZE

## PIU

This record contains selected TH fields, an RH, and an RU. This is the information used by components in layers above path control dealing with PIUs (e.g., half-session, LU network services). The PIU data structure does not contain a complete TH. It contains only the TH fields that are needed by the layers above PC. Other TH fields are not visible above PC.

```
PIU
   TH:  fields from the transmission header needed above the PC layer
      EFI expedited-flow indicator:  possible values:  EXP, NORMAL
      SNF:  contains a 16-bit sequence number field
      DCF:  data count field—contains length of BIU
   BIU:  basic information unit
      RH:  request/response header (see Appendix D)
      RU:  request unit (see Appendix E)
```

## SEND_PARM

SEND_PARM is a substructure that is embedded in SEND_DATA_RECORD (page A-24) and BID_WITH_ATTACH (page A-28). 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 half-session
```

## SESSION_INFORMATION

SESSION_INFORMATION is a substructure that is embedded in SESSION_ACTIVATED (page A-20) and ACTIVATE_SESSION_RSP (page A-20). Sent from LU Network Services to Resources Manager, SESSION_INFORMATION contains data about the session that has just been established.

```
SESSION_INFORMATION
   HS_ID:  half-session identifier
   HALF_SESSION_TYPE:  possible values:  PRI, SEC
   BRACKET_TYPE:  possible values:  FIRST_SPEAKER, BIDDER
```

# APPENDIX D. RH FORMATS

The request/response header (RH) is a 3-byte field; it may be a request header or a response header. Figure D-1 on page D-2 shows the RH formats and summarizes the allowed values.

The control fields in the request header include:

- Request indicator
- RU Category
- Format indicator
- Sense Data Included indicator
- Chaining Control
- Form of Response Requested
- Queued Response indicator
- Pacing indicator
- Bracket Control
- Change Direction indicator
- Code Selection indicator
- Enciphered Data indicator
- Padded Data indicator

The control fields in the response header include:

- Response indicator
- RU Category
- Format indicator
- Sense Data Included indicator
- Chaining Control
- Response Type indicator
- Queued Response indicator
- Pacing indicator

The above RH control fields are described below.

Request/Response Indicator (RRI): Denotes whether this is a request or a response.

RU Category: Denotes that the BIU belongs to one of four categories: session control (SC), network control (NC), data flow control (DFC), or function management data (FMD). (The NC category is not supported by T2.1 nodes.)

Format Indicator: Indicates which of two formats (denoted Format 1 and Format 0) is used within the associated RU (but not including the sense data field, if any; see Sense Data Included indicator, below).

For SC, NC, and DFC RUs, this indicator is always set to Format 1.

For (SSCP,PU) and (SSCP,LU) sessions, Format 1 indicates on FMD requests that the request RU includes a network services (NS) header and is field-formatted (with various encodings, such as binary data or bit-significant data, in the individual fields). Format 0 indicates that no NS header is contained in the request RU and the RU is character-coded. The Format indicator value on a response is the same as on the corresponding request.

For LU-LU sessions that support FM headers on FMD requests, Format 1 indicates that an FM header is present. The Format indicator is always set to 0 on positive responses.

Sense Data Included Indicator (SDI): Indicates that a 4-byte sense data field is included in the associated RU. The sense data field (when present) always immediately follows the RH and has the format and meaning described in Appendix G. Any other data contained in the RU follows the sense data field. Sense data is included on negative responses and on EXRs, where it indicates the type of condition causing the exception.

(The Format indicator does not describe or affect the sense data, which is always in the 4-byte format shown in Appendix G.)

Chaining Control: Indicates that a sequence of contiguous transmitted requests is being grouped in a chain. Two indicators, Begin Chain indicator (BCI) and End Chain indicator (ECI), together denote the relative position of the associated RU within a chain. The 1 values of these indicators (BCI = 1 and ECI = 1) are referred to as BC and EC, respectively.

| | |
|---|---|
| (BC, ¬EC) | = first RU in chain |
| (¬BC, ¬EC) | = middle RU in chain |
| (¬BC, EC) | = last RU in chain |
| (BC, EC) | = only RU in chain |

Responses are always marked "only RU in chain."

Form of Response Requested: In a request header, defines the response protocol to be executed by the request receiver.

```
Request Header
|              Byte 0                 |              Byte 1              |              Byte 2              |
 ________________________________________________________________________________________________________
|      RU                             |                                  |                                  |
|RRI  Category    FI  SDI BCI ECI|DR1I     DR2I ERI       QRI PI |BBI EBI CDI    CSI EDI PDI CEBI|
|= 0|     |    | r |   |   |   |   |   | r |   |   | r | r |   |   |   |   |   | r |   |   |   |    |
 --------------------------------------------------------------------------------------------------------
:                                     :                                  :                                  :
Response Header                       :                                  :                                  :
:                                     :                                  :                                  :
 ________________________________________________________________________________________________________
|      RU                             |                                  |                                  |
|RRI  Category    FI  SDI  1   1 |DR1I     DR2I RTI       QRI PI |                                  |
|= 1|     |    | r |   |   |   |   |   | r |   |   | r | r |   |   | r | r | r | r | r | r | r | r |
 --------------------------------------------------------------------------------------------------------
```

| Field | Description | Explanation/Usage |
|---|---|---|
| RRI | Request/Response indicator | 0 = request (RQ); 1 = response (RSP) |
| RU Category | Request/Response Unit Category | 00 = FM data (FMD)<br>01 = network control (NC)<br>10 = data flow control (DFC)<br>11 = session control (SC) |
| FI | Format indicator | 0 = no FM header (¬FMH), for LU-LU sessions; or character-coded without an NS header (¬NSH), for network services (NS)<br>1 = FM header (FMH) follows, for LU-LU sessions; or field-formatted with an NS header (NSH), for NS |
| SDI | Sense Data Included indicator | 0 = not included (¬SD); 1 = included (SD) |
| BCI | Begin Chain indicator | 0 = not first in chain (¬BC); 1 = first in chain (BC) |
| ECI | End Chain indicator | 0 = not last in chain (¬EC); 1 = last in chain (EC) |
| DR1I | Definite Response 1 indicator | 0 = ¬DR1; 1 = DR1 |
| DR2I | Definite Response 2 indicator | 0 = ¬DR2; 1 = DR2 |
| ERI | Exception Response indicator | Used in conjunction with DR1I and DR2I to indicate, in a request, the form of response requested:<br>(DR1I, DR2I, ERI) = 000 means no-response requested<br>= 100\|010\|110 means definite-response requested<br>= 101\|011\|111 means exception-response requested |
| RTI | Response Type indicator | 0 = positive (+); 1 = negative (-) |
| QRI | Queued Response indicator | 0 = response bypasses TC queues (¬QR);<br>1 = enqueue response in TC queues (QR) |
| PI | Pacing indicator | 0 = ¬PAC; 1 = PAC |
| BBI | Begin Bracket indicator | 0 = ¬BB; 1 = BB |
| EBI | End Bracket indicator | 0 = ¬EB; 1 = EB (reserved for LU type 6.2) |
| CDI | Change Direction indicator | 0 = do not change direction (¬CD);<br>1 = change direction (CD) |
| CSI | Code Selection indicator | 0 = code 0; 1 = code 1 |
| EDI | Enciphered Data indicator | 0 = RU is not enciphered (¬ED); 1 = RU is enciphered (ED) |
| PDI | Padded Data indicator | 0 = RU is not padded (¬PD); 1 = RU is padded (PD) |
| CEBI | Conditional End Bracket indicator | 0 = not conditional end bracket (¬CEB); 1 = conditional end bracket (CEB) (used for LU type 6.2; else, reserved) |

|r| = Reserved

Figure D-1. RH Formats

Three bits in a request header specify the form of response that is desired. They are: Definite Response 1 indicator (DR1I), Definite Response 2 indicator (DR2I), and the Exception Response indicator (ERI). They can be coded to request:

1. No-response, which means that a response will not be issued by the half-session receiving the request. (DR1I,DR2I) = (0,0) = (¬DR1,¬DR2) and ERI=0 is the only coding possible; the abbreviation RQN refers to a request with this coding. (A special response, ISOLATED PACING RESPONSE [IPR], does set [DR1I,DR2I,ERI]=[0,0,0], but it is used independently of the other responses listed. IPR is sent in connection with session-level pacing; the sequence number in its associated TH does not correlate it to any given request.)

2. Exception response, which means that a negative response will be issued by the half-session receiving the request only in the event of a detected exception (a positive response will not be issued). (DR1I, DR2I) = (1,0)|(0,1)|(1,1) and ERI=1 are the possible codings; RQE1, RQE2, and RQE3 are the abbreviations, respectively; the abbreviation RQE or RQE* refers to a request with any of these codings.

3. Definite response, which means that a response will always be issued by the half-session receiving the request, whether the response is positive or negative. (DR1I, DR2I) = (1,0)|(0,1)|(1,1) and ERI=0 are the possible codings; RQD1, RQD2, and RQD3 are the abbreviations, respectively; the abbreviation RQD or RQD* refers to a request with any of these codings.

A request that asks for an exception response or a definite response has one or both of the DR1I and DR2I bits set to 1 (three combinations); a response to a request returns the same (DR1I, DR2I) bit combination (see Figure D-2 on page D-4).

The setting of the DR1I, DR2I, and ERI bits varies by RU category. Chapter 4 and Chapter 6.2 define the settings for SC; Chapter 6.1 defines them for DFC; Chapter 4 defines them for network services FMD.

In the case of LU-LU sessions, BIND parameters (see Appendix E) specify the form of response to be requested during the session; see Chapter 2 and Chapter 5.3, as well as Figure D-2 on page D-4.

The (DR1I, DR2I, ERI) = (0, 0, 1) combination is reserved.

<u>Queued Response Indicator (QRI)</u>: In a response header for a normal-flow RU, the Queued Response indicator denotes whether the response is to be enqueued in TC queues: QRI=QR, or whether it is to bypass these queues: QRI=¬QR. In a request header for a normal-flow RU, it indicates what the setting of the QRI should be on the response, if any, to this request (i.e., the values on the request and response are the same).

For expedited-flow RUs, this bit is reserved.

The setting of the QRI bit is the same for all RUs in a chain.

<u>Response Type</u>: In a response header, two basic response types can be indicated: positive response or negative response. For negative responses, the RH is always immediately followed by four bytes of sense data in the RU. Thus, RTI=NEG and RTI=POS occur jointly with SDI=SD and SDI=¬SD, respectively.

Three kinds of positive and negative responses correspond to the three valid (DR1I, DR2I) combinations allowed on requests. The settings of the DR1I and DR2I bits in a response always equal the settings of the DR1I and DR2I bits of the form-of-response-requested field of the corresponding request header, except as shown in Figure D-2 on page D-4.

<u>Pacing</u>: In a request header, the Pacing Request indicator denotes that the sending TC element can accept a Pacing Response indicator.

The Pacing Response indicator in a response header is used to indicate to the receiving TC element that additional requests may be sent on the normal flow. The Pacing Response indicator may be <u>on</u> in an RH that is attached to a response RU on the normal flow; or, if desired, a separate, or isolated, response header may be used, to which no RU is attached. This latter RH signals only the pacing response; it is called an ISOLATED PACING RESPONSE (see Chapter 6.2). Isolated and nonisolated pacing responses are functionally equivalent.

| REQUEST | VALID RESPONSE | MEANING OF RESPONSE |
|---|---|---|
| RQD1=(1,0,0)<br><br>(Used by DFC) | +RSP1=(1,0,0)<br>-RSP1=(1,0,1) | positive response<br>negative response |
| RQE1=(1,0,1)<br><br>(Used by DFC and PS) | -RSP1=(1,0,1) | negative response |
| RQD2\|3=(*,1,0)<br><br>RQE2\|3=(*,1,1)<br><br>(Used by PS) | +RSP2\|3=(*,1,0)<br>-RSP2\|3=(*,1,1)<br><br>implied +RSP2\|3<br><br>-RSP2\|3=(*,1,1) | confirmed<br>not confirmed<br><br>reply received with no intervening response<br>not confirmed |
| RQN =(0,0,0)<br><br>(Not used) | | |

NOTES:

1. Values displayed in this table are in the order (DR1I,DR2I,ERI) for requests and (DR1I,DR2I,RTI) for responses.

2. All ¬EC requests are sent as RQE1.

Figure D-2. FMD Request/Response Combinations for Sessions between Two LU 6.2s

Bracket Control: Used to indicate the beginning or end of a group of exchanged requests and responses called a bracket. Bracket protocols are used only on LU-LU sessions. When used, BB appears only on the first request in the first chain of a bracket; CEB appears only on the last request of the last chain of a bracket. (When bracket usage is specified in BIND, the BIND request carries an implied BB.) The bracket indicators are set only on LUSTAT and FMD requests, and are thus sent normal-flow. See Chapter 6.1 for detailed discussion of bracket protocols.

Change Direction Indicator (CDI): Used when there is half-duplex (HDX) control of the normal flows within a session (not to be confused with link-level HDX protocols). It permits a sending half-session to direct the receiving half-session to send. The HDX protocol is useful to half-sessions with limited input/output capabilities that cannot simultaneously send and receive user data. When used, CD appears only on the last request in a chain; it is set only on LUSTAT and FMD requests. See Chapter 6.1 for detailed discussion of this protocol.

Code Selection Indicator (CSI): Specifies the encoding used for the associated FMD RU. When a session is activated, the half-sessions can choose to allow use of two codes in their FMD RUs (e.g., EBCDIC and ASCII), which they designate as Code 0 and Code 1. FM headers and request and response codes are not affected by the Code Selection indicator.

For SC, NC, and DFC RUs, this bit is reserved.

Enciphered Data Indicator (EDI): Indicates that information in the associated RU is enciphered under session-level cryptography protocols.

Padded Data Indicator (PDI): Indicates that the RU was padded at the end, before encipherment, to the next integral multiple of 8 bytes in length; the last byte of such padding is the count of pad bytes added, the count being a number (1-7 inclusive) in unsigned 8-bit binary representation.

# APPENDIX E. REQUEST-RESPONSE UNIT (RU) FORMATS

This appendix defines detailed RU formats. A categorized list of RU abbreviations is presented first, followed by an alphabetic list of request RU format descriptions, a summary of response RUs, and a list of response format descriptions for those positive response RUs that return data in addition to the request code. Two final sections describe control vectors and session keys.

The initial line for each RU in the two RU format description lists is in one of the following formats:

Requests

"RU ABBREVIATION; Origin NAU-->Destination NAU, Normal (Norm) or Expedited (Exp) Flow; RU Category (RU NAME)"

Responses

"RSP(RU ABBREVIATION); Origin NAU-->Destination NAU, Norm or Exp Flow; RU Category"

Notes:

1. "RU Category" is abbreviated as follows:

| | |
|---|---|
| DFC | data flow control |
| SC | session control |
| FMD NS(ma) | function management data, network services, maintenance services |
| FMD NS(s) | function management data, network services, session services |

2. The formats of character-coded FMD NS RUs are implementation dependent; LU-->LU FMD RUs (e.g., FM headers) are described in "Appendix H. FM Header and LU Services Commands" .

3. All values for field-formatted RUs that are not defined in this section are reserved.

4. The request code value X'FF' and the NS header values X'(3|7|B|F)F****' and X'**(3|7|B|F)F**' are set aside for implementation internal use, and will not be otherwise defined in SNA.

5. Throughout this appendix the following symbol-string types are used:

   - Type-A (Assembler oriented): a byte string consisting of one or more EBCDIC uppercase letters (A through Z), numerics (0 through 9), $, #, and @, the first character of which is nonnumeric.

   - Type-USS ("unformatted system services" or character-coded subset of the SNA character set): a byte string consisting of one or more EBCDIC uppercase letters (A through Z), numerics (0 through 9), $, #, @, line feed (X'15'), space (X'40') and the following 11 special characters: '=(),+-*./& with no restriction on the first character.

   - Type-AE (A extended): a byte string consisting of one or more EBCDIC lowercase letters (a through z), uppercase letters (A through Z), numerics (0 through 9), $, #, @, and period (.), with no restriction on the first character.

   - Type-GR (EBCDIC graphics): a byte string consisting of one or more EBCDIC characters in the range X'41' through X'FE', with no restriction on the first character.

   - Type-G (general): a byte string consisting of one or more bytes of binary values 0 through 255.

   The RU field to which a type-A, type-AE, or type-GR symbol string is assigned may be longer than the symbol string; in this case, the symbol string is left-justified within the field, which is filled out to the right with space (X'40') characters. Space characters, if present, are not part of the symbol string. If the symbol string is formed from the concatenation of two or more individual symbol strings, such as the fully-qualified LU name, the concatenated symbol string as a whole is left-justified within the field ,which is filled out to the right with space characters. Space characters, if present, are not part of the concatenated symbol string.

6. Throughout this appendix, reserved is used as follows: reserved bits, or fields, are currently set to 0's (unless explicitly stated otherwise); reserved values are those that currently are invalid. Correct usage of reserved fields is enforced by the sender; no receive checks are made on these fields.

7. Throughout this appendix, retired fields and values are those that were once defined by SNA but are no longer defined. To accommodate implementations of back-level SNA, current implementations of SNA treat retired fields as follows: send checks enforce the setting of retired fields to all 0's except where other unique values are required (described individually in this appendix); no receive checks are made on these fields, thereby accepting back-level settings of these fields. Special handling of retired fields, such as echoing or passing on retired fields as received, is discussed where appropriate.

## SUMMARY OF REQUEST RU'S BY CATEGORY

**SC**

|  |  |  |  |
|---|---|---|---|
| *ACTLU | CRV | DACTLU | UNBIND |
| *BIND |  |  |  |

**DFC**

|  |  |  |  |
|---|---|---|---|
| BIS | LUSTAT | RTR | SIG |

**FMD NS(ma)**

|  |  |
|---|---|
| ECHOTEST | REQECHO |

**FMD NS(s)**

|  |  |  |  |
|---|---|---|---|
| BINDF | CTERM | SESSEND | TERM-SELF |
| *CINIT | INIT-SELF | SESSST | UNBINDF |
| CLEANUP | NOTIFY |  |  |

* These request RUs require response RUs that, if positive, may contain data in addition to the NS header or request code. See "Summary of Response RU's" on page E-18 and "Positive Response RU's with Extended Formats" on page E-18 .

## INDEX OF RU'S BY NS HEADERS AND REQUEST CODES

Within DFC, SC, or any specific FMD NS category, the request code is unique. However, while a request code has only one meaning in a specific category, a given code can represent different requests in separate categories.

FMD NS Headers (Third byte is the request code)

| | |
|---|---|
| X'810387' | REQECHO |
| X'810389' | ECHOTEST |
| X'810601' | CINIT |
| X'810602' | CTERM |
| X'810620' | NOTIFY (SSCP<-->LU) |
| X'810629' | CLEANUP |
| X'810681' | INIT-SELF (Format 1) |
| X'810683' | TERM-SELF (Format 1) |
| X'810685' | BINDF |
| X'810686' | SESSST |
| X'810687' | UNBINDF |
| X'810688' | SESSEND |

DFC Request Codes

| | |
|---|---|
| X'04' | LUSTAT |
| X'05' | RTR |
| X'70' | BIS |
| X'C9' | SIG |

SC Request Codes

| | |
|---|---|
| X'0D' | ACTLU |
| X'0E' | DACTLU |
| X'31' | BIND |
| X'32' | UNBIND |
| X'C0' | CRV |

## REQUEST RU FORMATS

ACTLU; SSCP|PNCP-->LU, Exp; SC (ACTIVATE LOGICAL UNIT)

> ACTLU is sent from an SSCP|PNCP to an LU to activate a session between the SSCP|PNCP and the LU and to establish common session parameters.

```
0        X'0D' request code
1        Type activation requested:
         X'01' cold
         X'02' ERP
2        bits 0-3, FM profile:
                   X'0' FM profile 0
                   X'6' FM profile 6
         bits 4-7, TS profile:
                   X'1' TS profile 1 (only value defined)
```

BIND; PLU-->SLU, Exp; SC (BIND SESSION)

> BIND is sent from a primary LU to a secondary LU to activate a session between the LUs. The secondary LU uses the BIND parameters to help determine whether it will respond positively or negatively to BIND.

```
0        X'31' request code
1        bits 0-3, format: 0000
         bits 4-7, type:
                   0000 negotiable
2        FM profile:
         X'13' FM profile 19
3        TS profile:
         X'07' TS profile 7
         FM Usage—Primary LU Protocols for FM Data
4        bit  0, chaining use selection:
                 1 multiple-RU chains allowed from primary LU half-session
         bit  1, request control mode selection:
                 0 immediate request mode
         bits 2-3, chain response protocol used by primary LU half-session for FMD requests;
                   chains from primary will ask for:
                   11 definite or exception response
         bits 4-6, reserved
         bit  7, send End Bracket indicator:
                 0 primary will not send EB
         FM Usage—Secondary LU Protocols for FM Data
5        bit  0, chaining use selection:
                 1 multiple-RU chains allowed from secondary LU half-session
         bit  1, request control mode selection:
                 0 immediate request mode
         bits 2-3, chain response protocol used by secondary LU half-session for FMD requests;
                   chains from secondary will ask for:
                   11 definite or exception response
         bits 4-6, reserved
         bit  7, send End Bracket indicator
                 0 secondary will not send EB
         FM Usage—Common LU Protocols
6        bit  0, session segmenting support:
                 0 this LU supports reception of segments on this session
                 1 this LU does not support reception of segments on this session; the BIND
                   sender and receiver set the maximum RU sizes, in bytes 10-11 of BIND and
                   RSP(BIND), so that segmenting will not occur on the link for this session
         bit  1, FM header usage:
                 1 FM headers allowed
         bit  2, brackets usage and reset state:
                 0 brackets are used and bracket state managers' reset states are INB
         bit  3, bracket termination rule selection
                 1 Rule 1 (conditional termination) will be used during this session
         bit  4, alternate code set allowed indicator:
                 0 alternate code set will not be used
                 1 alternate code set may be used
         bits 5-6, reserved
         bit  7, BIND response queue capability:
                 0 BIND response cannot be held/queued
```

1 BIND sender allows bind receiver to queue BIND and withhold BIND response for an indefinite period
<u>Note:</u> BIND sender may provide a timer or operator interface to send UNBIND if session activation time exceeds BIND sender's limits. BIND queuing is terminated by sending UNBIND to the BIND receiver.

7 bits 0-1, normal-flow send/receive mode selection:
10 half-duplex flip-flop
bit 2, recovery responsibility:
1 symmetric responsibility for recovery
bit 3, contention winner/loser:
0 secondary is contention winner and primary is contention loser
1 primary is contention winner and secondary is contention loser
<u>Note:</u> Contention winner is also brackets first speaker.
bits 4-5, alternate code processing identifier (reserved unless Alternate Code Set Allowed indicator (byte 6, bit 4) is 1):
01 process alternate code FMD RUs as ASCII-8
<u>Note:</u> When the Alternate Code Processing Identifier indicator is set to the value 01, the entire FMD request RU is to be translated using the transforms defined by the ANSI X3.26 Hollerith Card Code.
bit 6, reserved
bit 7, half-duplex flip-flop reset states:
1 HDX-FF reset state is SEND for the primary and RECEIVE for the secondary (e.g., the primary sends normal-flow requests first after session activation)

<u>TS Usage</u>

8 bit 0, staging indicator for secondary TC to primary TC normal flow:
0 pacing in this direction occurs in one stage (only value used for PNCP-mediated sessions)
1 pacing in this direction occurs in more than one stage
<u>Note:</u> The meanings of 0 and 1 are reversed from the staging indicator for primary TC to secondary TC.
bit 1, reserved
bits 2-7, secondary TC's send window size: 0 means no pacing of requests flowing from the secondary

9 bits 0-1, reserved
bits 2-7, secondary TC's receive window size: a value of 0 causes the boundary function to substitute the value set by a system definition pacing parameter (if the system definition includes such a parameter) before it sends the BIND RU on to the secondary half-session; a value of 0 received at the secondary is interpreted to mean no pacing of requests flowing to the secondary

10 Maximum RU size sent on the normal flow by the secondary half-session: bit 0 is set to 1, and the byte is interpreted as X'ab' = a•2**b. (By definition, a≥8 and therefore X'ab' is a normalized floating point representation.) See Figure E-1 on page E-8 for all possible values.

11 Maximum RU size sent on the normal flow by the primary half-session: identical encoding as described for byte 10

12 bit 0, staging indicator for primary TC to secondary TC normal flow:
1 pacing in this direction occurs in one stage (only value used for PNCP-mediated sessions)
0 pacing in this direction occurs in two stages
<u>Note:</u> The meanings of 0 and 1 are reversed from the staging indicator for secondary to primary TC.
bit 1, reserved
bits 2-7, primary TC's send window size: a value of 0 causes the value set by a system definition pacing parameter (if the system definition includes such a parameter) to be assumed for the session; if this is also 0, it means no pacing of requests flowing from the primary. (For single-stage pacing in the primary-to-secondary direction, this field is redundant with, and indicates the same value as, the secondary TC's receive window size—see byte 9, bits 2-7, above.)

13 bits 0-1, reserved
bits 2-7, primary TC's receive window size: a value of 0 means no pacing of requests flowing to the primary. (For single-stage pacing in the secondary-to-primary direction, this field is redundant with, and indicates the same value as, the secondary TC's send window size—see byte 8, bits 2-7, above.)

<u>PS Profile</u>

14 bit 0, PS Usage field format:
0 basic format
bits 1-7, LU type:
0000110 LU type 6

<u>PS Usage</u> characteristics

15 LU-6 level:
X'02' Level 2 (i.e., LU 6.2)

16-22 Reserved

23 bits 0-2, retired
bits 3-7, reserved

24 bit 0, reserved
bits 1-2, synchronization level:
- 01 confirm is supported
- 10 confirm, sync point, and backout are supported

bit 3, reserved
bits 4-5, responsibility for session reinitiation:
- 00 operator controlled
- 01 primary half-session will reinitiate
- 10 secondary half-session will reinitiate
- 11 either may reinitiate

bit 6, parallel session support for LU-LU pair:
- 0 not supported
- 1 supported

bit 7, Change Number of Sessions GDS variable flow support (set to 1 if byte 24, bit 6 = 1):
- 0 not supported
- 1 supported

Note 1: fields defined by bits 0-5 are consistent with the corresponding fields in other BINDs used for the same (partner LU, mode name) pair.
Note 2: fields defined by bits 6-7 are consistent with the corresponding fields in other BINDs used for the same partner LU.

25 Reserved
End of PS Usage Field

26-k Cryptography Options
Note: Cryptography usage is consistent for all parallel sessions with the same (partner LU, mode name) pair.

26 bits 0-1, reserved
bits 2-3, session-level cryptography options:
- 00 no session-level cryptography supported
- 11 session-level mandatory cryptography supported; all cryptography key management is supported by the SSCP and LU; exchange (via +RSP(BIND)) and verification (via CRV) of the cryptography session-seed value is supported by the LUs for the session; all FMD requests are enciphered/deciphered by TC

bits 4-7, session-level cryptography options field length:
- X'0' no session-level cryptography specified; additional cryptography options fields (bytes 27-k) omitted
- X'9' session-level cryptography specified; additional options follow in next nine bytes

27 bits 0-1, session cryptography key encipherment method:
- 00 session cryptography key enciphered under SLU master cryptography key using a seed value of 0

bits 2-4, reserved
bits 5-7, cryptography cipher method:
- 000 block chaining with seed and cipher text feedback, using the Data Encryption Standard (DES) algorithm

28-k Session cryptography key enciphered under secondary LU master cryptography key; an eight-byte value that, when deciphered, yields the session cryptography key used for enciphering and deciphering FMD requests

k+1 Length of primary LU name. (For a session involving two T2.1 nodes, values 0 to 8 are valid; otherwise, values 1 to 8 are valid.)
Note: X'00' = no primary LU name present.

k+2-m Primary LU name or, if the secondary LU issued the INIT-SELF, the uninterpreted name as carried in that RU

m+1 Length of user data

m+2-n User data

m+2 User data key:
X'00' structured subfields follow
Note: Individual structured subfields may be omitted entirely. When present, they appear in ascending field number order.

m+3-n Structured subfields. (For detailed definitions, see "User Data Structured Subfield Formats" on page E-16 .)

n+1 Length of user request correlation (URC) field (values 0 to 12 are valid)
Note: X'00' = no URC present.

n+2-p URC: LU-defined identifier (present only if carried in INIT from SLU)

p+1 Length of secondary LU name. (For a session involving two T2.1 nodes, values 0 to 8 are valid; otherwise, values 1 to 8 are valid.)
Note: X'00' = no secondary LU name present.

p+2-r Secondary LU name

Note 1: The length of the BIND RU cannot exceed 256 bytes, lest a negative response be returned.

Note 2: If the last byte of a request is a length field and that field is 0, that byte may be omitted from the BIND request.

| | Mantissa (a) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Exponent (b) | 8 | 9 | A (10) | B (11) | C (12) | D (13) | E (14) | F (15) |
| 0 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 |
| 2 | 32 | 36 | 40 | 44 | 48 | 52 | 56 | 60 |
| 3 | 64 | 72 | 80 | 88 | 96 | 104 | 112 | 120 |
| 4 | 128 | 144 | 160 | 176 | 192 | 208 | 224 | 240 |
| 5 | 256 | 288 | 320 | 352 | 384 | 416 | 448 | 480 |
| 6 | 512 | 576 | 640 | 704 | 768 | 832 | 896 | 960 |
| 7 | 1024 | 1152 | 1280 | 1408 | 1536 | 1664 | 1792 | 1920 |
| 8 | 2048 | 2304 | 2560 | 2816 | 3072 | 3328 | 3584 | 3840 |
| 9 | 4096 | 4608 | 5120 | 5632 | 6144 | 6656 | 7168 | 7680 |
| A (10) | 8192 | 9216 | 10240 | 11264 | 12288 | 13312 | 14336 | 15360 |
| B (11) | 16384 | 18432 | 20480 | 22528 | 24576 | 26624 | 28672 | 30720 |
| C (12) | 32768 | 36864 | 40960 | 45056 | 49152 | 53248 | 57344 | 61440 |
| D (13) | 65536 | 73728 | 81920 | 90112 | 98304 | 106496 | 114688 | 122880 |
| E (14) | 131072 | 147456 | 163840 | 180224 | 196608 | 212992 | 229376 | 245760 |
| F (15) | 262144 | 294912 | 327680 | 360448 | 393216 | 425984 | 458752 | 491520 |

Note: A value of X'ab' in byte 10 or byte 11 of BIND represents a•2**b. For example, X'C5' represents (in decimal) 12•2**5 = 384.

Figure E-1. RU Sizes Corresponding to Values X'ab' in BIND

BINDF; PLU-->SSCP, Norm; FMD NS(s) (BIND FAILURE)

BINDF is sent, with no-response requested, by the PLU to notify the SSCP that the attempt to activate the session between the specified LUs has failed.

| | |
|---|---|
| 0-2 | X'810685' NS header |
| 3-6 | Sense data |
| 7 | Reason: |
| | bit 0, reserved |
| | bit 1, 1 BIND error in reaching SLU |
| | bit 2, 1 setup reject at PLU |
| | bit 3, 1 setup reject at SLU |
| | bits 4-7, reserved |
| 8-m | Session key, as described in the section "Session Keys" on page E-23 |
| | Note: One of the following session keys is used: |
| | X'07' network address pair: PLU and SLU, respectively |
| | X'15' network-qualified address pair: PLU and SLU, respectively |

BIS; LU-->LU, Norm; DFC (BRACKET INITIATION STOPPED)

BIS is sent by a half-session to indicate that it will not attempt to begin any more brackets.

| | |
|---|---|
| 0 | X'70' request code |

CINIT; SSCP-->PLU, Norm; FMD NS(s) (CONTROL INITIATE)

CINIT requests the PLU to attempt to activate, via a BIND request, a session with the specified SLU.

| | |
|---|---|
| 0-2 | X'810601' NS header |
| 3 | Format |
| | bits 0-3, 0000 Format 0 (only value defined) |
| | Note: CINIT format 0 may carry control vectors at the end of the basic RU. |
| | bits 4-7, reserved |
| 4 | bit 0, INITIATE origin: |
| | 0 ILU is OLU |
| | 1 ILU is not OLU |
| | bit 1, reserved |
| | bit 2, origin LU: |
| | 0 SLU is OLU |
| | 1 PLU is OLU |
| | bit 3, initiator: |
| | 0 network user is the initiator |
| | 1 network manager is the initiator |
| | bits 4-7, reserved |
| 5-9 | Session key, as described in the section "Session Keys" on page E-23 |
| | Note: The following session key is used: |
| | X'07' network address pair: PLU and SLU, respectively |
| | Note: If control vector X'15' is supported by the LU, then bytes 5-9 are reserved; otherwise, these bytes contain session key X'07' when sent from the SSCP to a subarea LU. |
| 10-11 | Length, in binary, of BIND Image field |
| 12-m | BIND image: bytes 1-p of the BIND RU, i.e., through the URC field (see BIND format description) |
| | Note: The URC Length field is included, even if it is set to 0. |
| m+1-n | Name of SLU |
| m+1 | Type: X'F3' logical unit |
| m+2 | Length, in binary, of symbolic name (1-8 characters) |
| m+3-n | Symbolic name, in EBCDIC characters |
| n+1-n+2 | Retired—set to X'0000' |
| n+3-r | User Field |
| n+3 | Length, in binary, of user data |
| | Note: X'00' = no user data is present. |
| n+4-r | User data (retired for LU 6.2—not sent by current-level implementation) |
| r+1-s | LU or Non-SNA Device Specifications |
| r+1-r+2 | Length, in binary, of Characteristics field |
| | Note: X'0000' = no Characteristics field is present. |
| r+3-s | Characteristics field (retired for LU 6.2—not sent by current-level implementation) |
| s+1 | Length of Session Cryptography Key field |
| | Note: X'00' = no Session Cryptography Key field present. |

| | |
|---|---|
| s+2-t | Session Cryptography Key field: session cryptography key enciphered under PLU master cryptography key |

<u>Note:</u> End of base RU

| | |
|---|---|
| t+1-u | Control vector, as described in the section "Control Vectors" on page E-20 |
| | <u>Note:</u> The following vector keys are used in CINIT: |
| | X'0D' Mode/Class of Service/Virtual Route List (this control vector is always present) |
| | X'15' network-qualified address pair: PLU and SLU, respectively (This control vector is always present when using extended network addressing; otherwise, it is optional.) |

CLEANUP; SSCP-->SLU, Norm; FMD NS(s) (CLEAN UP SESSION)

CLEANUP is sent by the SSCP to an LU (in a subarea node or BF for peripheral LU) requesting that the LU or BF attempt to deactivate the session for the specified (PLU,SLU) network address pair.

| | |
|---|---|
| 0-2 | X'810629' NS header |
| 3 | bits 0-3, 0000 Format 0 |
| | bits 4-7, reserved |
| 4 | Reserved |
| 5 | Reason: |
| | bit 0, 0 network user |
| | 1 network manager |
| | bit 1, 0 normal |
| | 1 abnormal |
| | bits 2-7, reserved |
| 6-n | Session key, as described in the section "Session Keys" on page E-23 |
| | <u>Note:</u> One of the following session keys is used: |
| | X'07' network address pair: PLU and SLU, respectively |
| | X'15' network-qualified address pair: PLU and SLU, respectively |

CTERM; SSCP-->PLU, Norm; FMD NS(s) (CONTROL TERMINATE)

CTERM requests that the PLU attempt to deactivate a session identified by the specified (PLU,SLU) network address pair.

| | |
|---|---|
| 0-2 | X'810602' NS header |
| 3 | bits 0-3, 0000 Format 0 |
| | bits 4-7, reserved |
| 4 | Type: |
| | bits 0-1, reserved |
| | bits 2-3, 00 reserved |
| | 01 orderly |
| | 10 forced |
| | 11 reserved |
| | bits 4-7, reserved |
| 5 | Reason: |
| | bit 0, 0 network user |
| | 1 network manager |
| | bit 1, 0 normal |
| | 1 abnormal |
| | bits 2-7, reserved |
| 6-7 | Reserved |
| 8-m | Session key, as described in the section "Session Keys" on page E-23 |
| | <u>Note:</u> One of the following session keys is used: |
| | X'07' network address pair: PLU and SLU, respectively |
| | X'15' network-qualified address pair: PLU and SLU, respectively |
| m+1-m+2 | Retired: set to X'0000' |

CRV; PLU-->SLU, Exp; SC (CRYPTOGRAPHY VERIFICATION)

CRV, a valid request only when session-level cryptography was selected in BIND, is sent by the primary LU session control to verify cryptography security and thereby enable sending and receiving of FMD requests by both half-sessions.

| | |
|---|---|
| 0 | X'C0' request code |
| 1-8 | A transform of the (deciphered) cryptography session-seed value received (enciphered) in bytes 28-k of +RSP(BIND), re-enciphered under the session cryptography key using a seed value of 0; the transform is the cryptography session-seed value with the first four bytes inverted |
| | <u>Note:</u> The cryptography session-seed is used as the seed for all session-level cryptography encipherment and decipherment provided for FMD RUs. |

**DACTLU; SSCP-->LU, Exp; SC (DEACTIVATE LOGICAL UNIT)**

DACTLU is sent to deactivate the session between the SSCP and the LU.

| | |
|---|---|
| 0 | X'0E' request code |

Note: End of short (one-byte) request.

| | |
|---|---|
| 1 | Type of deactivation requested: |
| | X'01' normal deactivation |
| | X'03' session outage notification (SON) |
| 2 | Cause (reserved if byte 1 ≠ X'03'): |
| | X'07' virtual route inoperative: the virtual route serving the SSCP-LU session has become inoperative, thus forcing the deactivation of the session |
| | X'08' route extension inoperative: the route extension serving the SSCP-LU session has become inoperative, thus forcing the deactivation of the session |
| | X'09' hierarchical reset: the SSCP-LU session is being deactivated because of a +RSP(ACTPU, Cold) |
| | X'0B' virtual route deactivated: the SSCP-LU session is being deactivated because of a forced deactivation of the virtual route being used by the session |
| | X'0C' SSCP or LU failure—unrecoverable: the SSCP-LU session had to be reset because of an abnormal termination; recovery from the failure was not possible |
| | X'0D' session override: the SSCP-LU session has to be deactivated because of a more recent session activation request for the SSCP to subarea PU session over a different virtual route |
| | X'0E' SSCP or LU failure—recoverable: the SSCP-LU session had to be deactivated because of an abnormal termination of the SSCP or LU of the session; recovery from the failure may be possible |
| | X'0F' cleanup: the SSCP is resetting its half-session before receiving the response from the LU being deactivated |

**ECHOTEST; SSCP-->LU, Norm; FMD NS(ma) (ECHO TEST)**

ECHOTEST carries test data to the target LU; the test data is the same as that carried in the corresponding REQECHO.

| | |
|---|---|
| 0-2 | X'810389' NS header |
| 3-n | Echo data field: same as bytes 4-m in the soliciting REQECHO |
| 3 | Number of data bytes |
| 4-n | Data |

**INIT-SELF Format 1; ILU-->SSCP, Norm; FMD NS(s) (INITIATE-SELF)**

INIT-SELF from the ILU requests that the SSCP authorize and assist in the initiation of a session between the LU sending the request (that is, the ILU, which also becomes the OLU) and the LU named in the request (the DLU).

| | |
|---|---|
| 0-2 | X'810681' NS header |
| 3 | bits 0-3, format: |
| | 0001 Format 1 |
| | bits 4-7, reserved |
| 4 | Type: |
| | bits 0-1, 01 initiate only (I): do not enqueue |
| | 11 initiate/enqueue (I/Q): enqueue the request if it cannot be satisfied immediately |
| | bits 2-4, reserved |
| | bits 5-6, PLU/SLU specification: |
| | 00 DLU is PLU |
| | 01 DLU is SLU |
| | bit 7, reserved |
| 5 | Queuing conditions for DLU: |
| | bit 0, 0 do not enqueue if session limit exceeded |
| | 1 enqueue if session limit exceeded |
| | bit 1, 0 do not enqueue if DLU is not currently able to comply with the PLU/SLU specification (as given in byte 4, bits 5-6) |
| | 1 enqueue if DLU is not currently able to comply with the PLU/SLU specification |
| | bits 2-4, reserved |
| | bits 5-6, queuing position/service: |
| | 01 enqueue this request FIFO |
| | bit 7, reserved |
| | Note: Since queuing conditions are specified for the DLU only, the following default values are used by SSCP(OLU) for the OLU: |
| | • Enqueue if session limit exceeded. |

| | |
|---|---|
| | • Enqueue this request at the back of the queue (FIFO). |
| 6-7 | Reserved |
| 8-15 | Mode name: an eight-character symbolic name (implementation and installation dependent) that identifies the set of rules and protocols to be used for the session; used by the SSCP(SLU) to select the BIND image that will be used by the SSCP(PLU) to build the CINIT request |
| 16-n | Uninterpreted Name of DLU |
| 16 | Type: X'F3' logical unit |
| 17 | Length, in binary, of DLU name |
| 18-n | DLU name EBCDIC character string |
| n+1-n+2 | Retired: set to X'0000' |
| n+3-r(=n+3) | Reserved |
| r+1-s | User Request Correlation (URC) Field |
| r+1 | Length, in binary, of URC<br>Note: X'00' = no URC. (The length field is always present.) |
| r+2-s | URC: end-user defined identifier; this value can be returned by the SSCP in a subsequent NOTIFY to correlate a given session to this initiating request |

LUSTAT; LU-->LU|SSCP, Norm; DFC (LOGICAL UNIT STATUS)

> LUSTAT is used by one half-session to send up to four bytes of status information to its paired half-session.

| | |
|---|---|
| 0 | X'04' request code |
| 1-4 | Status value + status extension field (two bytes each):<br>X'0006'+'rrrr' no-op (used to allow an RH to be sent when no other request is available or allowed) + reserved field |

NOTIFY; SSCP<-->LU, Norm; FMD NS(s) (NOTIFY)

> NOTIFY is used to send information between an SSCP and an LU. NOTIFY carries information in the form of a (vector key, vector data) pair.

| | |
|---|---|
| 0-2 | X'810620' NS header (for SSCP-->LU and LU-->SSCP) |
| 3-p | One NOTIFY vector as described in detail below<br>Note: One of the following vector keys is used:<br>X'03' ILU/TLU Notification: sent by the SSCP to inform the sender of an INIT or TERM request of the status of the session<br>X'0C' LU Session Services Capabilities: sent by the LU to inform the SSCP having an active session with the sending LU of the current LU-LU session services capability of that LU |

NOTIFY vectors (described zero-origin)

ILU/TLU Notification NOTIFY Vector

| | |
|---|---|
| 0 | Key: X'03' |
| 1 | Status:<br>X'03' procedure error |
| 2-9 | PCID: a unique value used as a session identifier |
| 10 | Reason (defined for Status field value of X'03' only)<br>Setup Procedure Error<br>bit 0, 1 CINIT error in reaching the PLU<br>bit 1, 1 BIND error in reaching the SLU<br>bit 2, 1 setup reject at the PLU<br>bit 3, 1 setup reject at the SLU<br>bit 4, 0 setup procedure error<br>bit 5, reserved<br>bit 6, 1 setup reject at SSCP<br>bit 7, reserved |
| 11-14 | Sense data |
| 15-m | Session key, as described in the section "Session Keys" on page E-23<br>Note: One of the following session keys is used:<br>X'06' network name pair: (PLU or OLU) and (SLU or DLU), respectively<br>X'15' network-qualified address pair: PLU and SLU, respectively |
| m+1-n | User Request Correlation (URC) Field |
| m+1 | Length, in binary, of the URC |
| m+2-n | URC: end user defined identifier, specified in an INIT request; used to correlate the NOTIFY to the initiating requests |

LU-LU Session Services Capabilities NOTIFY Vector

| | |
|---|---|
| 0 | Key: X'0C' |
| 1 | Length, in binary, of Vector Data field |
| 2-m | Vector Data |

| | |
|---|---|
| 2 | bits 0-3, primary LU capability:<br>0000 PLU capability is inhibited, sessions can neither be queued nor started<br>0001 PLU capability is disabled, sessions can be queued but not started<br>0010 reserved<br>0011 PLU capability is enabled, sessions can be queued or started<br>bits 4-7, secondary LU capability:<br>0000 SLU capability is inhibited, sessions can neither be queued nor started<br>0001 SLU capability is disabled, sessions can be queued but not started<br>0010 reserved<br>0011 SLU capability is enabled, sessions can be queued or started |
| 3-4 | LU-LU session limit (where a value of 0 means that no session limit is specified) |
| 5-6 | LU-LU session count: the number of LU-LU sessions that are not reset, for this LU, and for which SESSEND will be sent to the SSCP |
| 7 | bit 0, parallel session capability:<br>0 parallel sessions not supported<br>1 parallel sessions supported<br>bit 1, reserved<br>bit 2, SESSST capability in RSP(ACTLU) (reserved in NOTIFY):<br>0 SESSST RU is suppressed if SLU<br>1 SESSST RU is sent if SLU<br>bits 3-7, reserved |
| 8-15(=m) | Retired (set to X'4040404040404040') or omitted |

REQECHO; LU-->SSCP, Norm; FMD NS(ma) (REQUEST ECHO TEST)

REQECHO requests that the SSCP return to the LU in ECHOTEST the data included in REQECHO.

| | |
|---|---|
| 0-2 | X'810387' NS header |
| 3 | Repetition factor: number of times the test data is to be echoed to the target LU<br>Note: X'00' is not a valid repetition factor. |
| 4-m | Echoed Data Field |
| 4 | Number of data bytes to be echoed |
| 5-m | Echoed data |

RTR; LU-->LU, Norm; DFC (READY TO RECEIVE)

RTR indicates to the bidder that it is now allowed to initiate a bracket. RTR is sent only by the first speaker.

| | |
|---|---|
| 0 | X'05' request code |

SESSEND; LU-->SSCP, Norm; FMD NS(s) (SESSION ENDED)

SESSEND is sent, with no response requested, by the LU (or the boundary function on behalf of the LU in a peripheral node) to notify the SSCP that the session between the specified LUs has been successfully deactivated.

| | |
|---|---|
| 0-2 | X'810688' NS header |
| 3 | bits 0-3, format:<br>0010 format 2<br>bits 4-7, reserved |
| 4 | Cause: indicates the reason for the deactivation of the LU-LU session (see UNBIND for values) |
| 5 | Action indicating if any resultant action is to be taken and by whom:<br>X'01' normal, no resultant automatic action |
| 6-n | Session key, as described in the section "Session Keys" on page E-23<br>Note: One of the following session keys is used:<br>X'07' network address pair: PLU and SLU, respectively<br>X'15' network-qualified address pair: PLU and SLU, respectively |

SESSST; LU-->SSCP, Norm; FMD NS(s) (SESSION STARTED)

SESSST is sent, with no response requested, by the LU (or the boundary function on behalf of the LU in a peripheral node) to notify the SSCP that the session between the specified LUs has been successfully activated.

| | |
|---|---|
| 0-2 | X'810686' NS header |
| 3 | Format:<br>X'00' Format 0: no control vectors present<br>X'01' Format 1: control vectors present in bytes n+1-p |
| 4-n | Session key, as described in the section "Session Keys" on page E-23 |

Note: One of the following session keys is used:
X'07' network address pair: PLU and SLU, respectively
X'15' network-qualified address pair: PLU and SLU, respectively

Note: End of Format 0; Format 1 continues below.

n+1-p One or more control vectors, as described in the section "Control Vectors" on page E-20
Note: The following vector keys may be used in SESSST:
X'1E' VR-ER Mapping Data
X'23' Local Form Session Identifier

SIG; LU-->LU, Exp; DFC (SIGNAL)

SIG is an expedited request that can be sent between half-sessions, regardless of the status of the normal flows. 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.

```
0         X'C9' request code
1-2       Signal code:
          X'0001' request to send
3-4       Signal extension:
          X'0001' soft
```

TERM-SELF Format 1; TLU-->SSCP, Norm; FMD NS(s) (TERMINATE-SELF)

TERM-SELF from the TLU requests that the SSCP assist in the termination of one or more sessions between the sender of the request (TLU = OLU) and the DLU.

```
0-2       X'810683' NS header
3         bits 0-3, format:
                    0001 Format 1
          bits 4-6, reserved
          bit  7, 1 indicates that byte 3, bits 0-3, contain the format value
4         Type:
          bits 0-1, 01 the request applies to active, pending-active, and queued sessions
          bit  2, reserved if byte 4, bit 7 = 1; otherwise:
                  0 forced termination—session to be deactivated immediately and uncondi-
                    tionally
                  1 orderly termination—permitting an end-of-session procedure to be executed
                    at the PLU before the session is deactivated
          bit  3, 1 send DACTLU to OLU when appropriate; no further session initiation request
                    will be sent (from this sender) for OLU
          bit  4, reserved
          bits 5-6, 00 select session(s) for which DLU is PLU
                    01 select session(s) for which DLU is SLU
                    10 select session(s) regardless of whether DLU is SLU or PLU
                    11 reserved
          bit  7, 0 orderly or forced (see byte 4, bit 2)
                  1 clean up
5         Reason:
          bit  0, 0 network user
          bit  1, 0 normal termination
                  1 abnormal termination
          bits 2-7, reserved
6-7       Reserved
8-n       Session key, as described in the section "Session Keys" on page E-23
          Note: One of the following session keys is used:
          X'0A' URC
                Note: This URC is the one carried in the INIT issued previously by the same LU
                (i.e., ILU = TLU), and differs from the one in bytes n+4 through p.
n+1-n+2   Retired: set to X'0000'
n+3-p     User Request Correlation (URC) Field
n+3       Length, in binary, of URC field
          Note: X'00' = no URC.
n+4-p     URC: end-user defined identifier; this value can be returned by the SSCP in a subse-
          quent NOTIFY to correlate the NOTIFY to this terminating request
```

UNBIND; LU-->LU, Exp; SC (UNBIND SESSION)

UNBIND is sent to deactivate a session between the two LUs.

| | |
|---|---|
| 0 | X'32' request code |
| 1 | Type: |
| | X'01' normal end of session |
| | X'02' BIND forthcoming: retain the node resources allocated to this session, if possible |
| | X'06' invalid session parameters: the BIND negotiation has failed due to an inability of the primary half-session to support parameters specified by the secondary |
| | X'07' virtual route inoperative: the virtual route used by the LU-LU session has become inoperative, thus forcing the deactivation of the identified LU-LU session |
| | X'08' route extension inoperative: the route extension used by the LU-LU session has become inoperative, thus forcing the deactivation of the identified LU-LU session |
| | X'09' hierarchical reset: the identified LU-LU session is being deactivated because of a +RSP((ACTPU \| ACTLU), Cold) |
| | X'0A' SSCP gone: the identified LU-LU session had to be deactivated because of a forced deactivation of the SSCP-PU or SSCP-LU session (e.g., DACTPU, DACTLU, or DISCONTACT) |
| | X'0B' virtual route deactivated: the identified LU-LU session had to be deactivated because of a forced deactivation of the virtual route being used by the LU-LU session |
| | X'0C' LU failure—unrecoverable: the identified LU-LU session had to be deactivated because of an abnormal termination of the PLU or SLU; recovery from the failure was not possible |
| | X'0E' LU failure—recoverable: the identified LU-LU session had to be deactivated because of an abnormal termination of one of the LUs of the session; recovery from the failure may be possible |
| | X'0F' cleanup: the LU sending UNBIND is resetting its half-session before receiving the response from the partner LU |
| | X'11' gateway node cleanup: a gateway node is cleaning up the session because a gateway SSCP has directed the gateway node (via NOTIFY) to deactivate the session (e.g., a session setup error or session takedown failure has occurred) |
| | X'FE' format or protocol error: the LU sending UNBIND has detected a format or protocol error; the error is identified by the associated sense data |
| 2-5 | Sense data (included only when Type = X'FE'; otherwise, this field is omitted): same value as generated at the time the error was originally detected (e.g., for a negative response, receive check, or EXR) |

UNBINDF; PLU-->SSCP, Norm; FMD NS(s) (UNBIND FAILURE)

UNBINDF is sent, with no-response requested, by the PLU to notify the SSCP that the attempt to deactivate the session between the specified LUs has failed (for example, because of a path failure).

| | |
|---|---|
| 0-2 | X'810687' NS header |
| 3-6 | Sense data |
| 7 | Reason: |
| | bit 0, reserved |
| | bit 1, 1 UNBIND error in reaching SLU |
| | bit 2, 1 takedown reject at PLU |
| | bits 3-7, reserved |
| 8-n | Session key, as described in the section "Session Keys" on page E-23 |
| | Note: One of the following session keys is used: |
| | X'07' network address pair: PLU and SLU, respectively |
| | X'15' network-qualified address pair: PLU and SLU, respectively |

## USER DATA STRUCTURED SUBFIELD FORMATS

The structured subfields of the User Data field are defined as follows (shown with zero-origin indexing of the subfield bytes—see the individual RU description for the actual displacement within the RU). Each subfield starts with a one-byte binary Length field and is identified by a subfield number in the following byte. The length does not include the Length byte itself. When more than one subfield is included, they appear in ascending order by subfield number.

Any subfields received in the Structured User Data field of BIND that are not recognized by the SLU are discarded and not returned as part of the Structured User Data field of the RSP(BIND).

### Unformatted Data Structured Data Subfield

> The Unformatted Data subfield may optionally be sent in BIND or RSP(BIND). The content is implementation-defined.

| | |
|---|---|
| 0 | Length of the remainder of the Unformatted Data subfield: values 1 to 17 are valid |
| 1 | X'00' |
| 2-n | Unformatted data: a type-G symbol string |

### Mode Name Structured Data Subfield

> The Mode Name subfield is present in both BIND and RSP(BIND) if the PLU knows the mode name being used by the session.

| | |
|---|---|
| 0 | Length of the remainder of the Mode Name Structured User Data subfield: values 1 to 9 are valid |
| 1 | X'02' |
| 2-n | Mode name: 0 to 8 type-A symbol string characters with optional (but not significant) trailing blanks |

### Session Instance Identifier Structured Data Subfield

> The Session Instance Identifier subfield may be present in both BIND and RSP(BIND).

| | |
|---|---|
| 0 | Length of the remainder of the Session Instance Identifier subfield: values 3 to 9 are valid |
| 1 | X'03' |
| 2-n | Session instance identifier: a type-G symbol string<br>Note: In BIND, the PLU sets a unique session instance identifier of length 1 to 7 and appends it to X'00'. If known, the SLU compares its fully qualified name with that of the PLU; if the PLU name > SLU name then the SLU changes the first byte of the Session Instance Identifier subfield in the BIND response from X'00' to X'F0'; if the PLU name < SLU name then the subfield is simply echoed. The session instance identifier is alway present when using either parallel sessions or synchronization level "all." |

### Fully Qualified PLU Network Name Structured Data Subfield

> BIND contains the Fully Qualified PLU Network Name subfield (if the name is known by the PLU).

| | |
|---|---|
| 0 | Length of the remainder of the Fully Qualified PLU Network Name subfield: values 2 to 18 are valid |
| 1 | X'04' |
| 2-n | Fully qualified PLU network name<br>Note: The fully qualified PLU network name is 1 to 17 bytes in length, consisting of an optional 1- to 8-byte network ID and a 1- to 8-byte LU name, both of which are type-A symbol strings. When present, the network ID is concatenated to the left of the LU name, using a separating period and having the form "NWID.NAME"; when the network ID is omitted, the period is also omitted. |

### Fully Qualified SLU Network Name Structured Data Subfield

> The RSP(BIND) contains the Fully Qualified SLU Network Name subfield (if the name is known by the SLU).

| | |
|---|---|
| 0 | Length of the remainder of the Fully Qualified SLU Network Name subfield: values 2 to 18 are valid |
| 1 | X'05' |
| 2-n | Fully qualified SLU network name |

Note: The fully qualified SLU network name is 1 to 17 bytes in length, consisting of an optional 1- to 8-byte network ID and a 1- to 8-byte LU name, both of which are type-A symbol strings. When present, the network ID is concatenated to the left of the LU name, using a separating period and having the form "NWID.NAME"; when the network ID is omitted, the period is also omitted.

## SUMMARY OF RESPONSE RU'S

Apart from the exceptions cited below, response RUs return the number of bytes specified in the following table; only enough of the request RU is returned to include the field-formatted request code.

| RU Category of Response | Number of Bytes in RU |
|---|---|
| SC | 1 |
| DFC | 1 |
| FMD NS (FI=1) (field-formatted) | 3 |
| FMD NS (FI=0) (character-coded) | 0 |
| FMD (LU-LU) | 0 |

Various positive response RUs return additional data. See "Positive Response RU's with Extended Formats" for details.

All negative responses return four bytes of sense data in the RU, followed by either (1) the number of bytes specified in the table above or (2) three bytes (or the entire request RU, if shorter than three bytes). The second option applies where a sensitivity to SSCP-based sessions versus LU-LU sessions does not exist and can be chosen for implementation simplicity. Refer to "Appendix G. Sense Data" for sense data values and their corresponding meanings.

## POSITIVE RESPONSE RU'S WITH EXTENDED FORMATS

```
RSP(ACTLU); LU-->SSCP, Exp; SC
0         X'0D' request code
1         Type of activation selected:
          X'01' cold
          X'02' ERP
2         bits 0-3, FM profile:
                    X'0' FM Profile 0
                    X'6' FM Profile 6
                    Note: This field contains the same value as the FM profile field received
                    in the ACTLU request except in the following case.  If the request specified
                    FM profile 0, the LU may respond either FM profile 0 or FM profile 6.
          bits 4-7, TS profile:  same as the corresponding request
3-m       Control vectors, as described in the section "Control Vectors" on page E-20
```

Note: Two versions of this RU are defined.

- A full response can be sent in which all fields and control vectors are present. These control vectors always appear in the following order:

  X'00' SSCP-LU session capabilities
  X'0C' LU-LU session services capabilities

- A two-byte response can be sent; it means maximum RU size = 256 bytes, LU-LU session limit = 1, the LU can act as a secondary LU, and all other fields in control vectors X'00' and X'0C' are defaulted to 0's.

RSP(BIND); SLU-->PLU, Exp; SC

| | |
|---|---|
| 0 | X'31' request code |
| 1 | bits 0-3, format: 0000 |
| | bits 4-7, type: 0000 negotiable |
| 2-25 | Bytes as received on BIND request, or bytes having the same format, but possibly with values changed from those received on the BIND request |
| 26-k | <u>Cryptography Options</u> |
| 26 | bits 0-1, reserved |
| | bits 2-3, session-level cryptography options |
| | bits 4-7, session-level cryptography options field length: same value returned as received in the request. (Bytes 27-k are omitted if this length field is omitted or set to 0.) |
| 27 | bits 0-1, session cryptography key encipherment method: same value returned as received in the request, if present |
| | bits 2-4, reserved |
| | bits 5-7, cryptography cipher method: same value returned as received in the request, if present |
| 28-k | An eight-byte implementation-chosen, nonzero, pseudo random session-seed cryptography value enciphered under the session cryptography key, if session-level cryptography is specified |
| k+1-r | Bytes as received on BIND request, or bytes having the same format, but possibly with values changed from those received on the BIND request |

<u>Note 1:</u> The extended format is required for the BIND response.

<u>Note 2:</u> On a response, if the last byte of a response is a Length field and that field is 0, that byte may be omitted from the response. This applies also to byte 26 (where the count occupies only bits 4-7) if bits 0-3 are also 0—the entire byte may be omitted if no bytes follow.

<u>Note 3:</u> Reserved fields in the BIND are set by the SLU to binary 0's in the RSP(BIND); any fields at the end of the BIND that are not recognized by the SLU are discarded and not returned in the RSP(BIND).

RSP(CINIT); PLU-->SSCP, Norm; FMD NS(s)

| | |
|---|---|
| 0-2 | X'810601' NS header |
| 3-n | Control vectors as described in the section "Control Vectors" on page E-20<br><u>Note:</u> The following control vector key is used in RSP(CINIT):<br>X'FE' control vector keys not recognized |

## COMMON STRUCTURED SUBFIELDS

### CONTROL VECTORS

The following table shows, by key value, the control vector and the message-unit structures that can carry the control vector.

| Key | Control Vector | Applicable Message-Unit Structures |
|---|---|---|
| X'00' | SSCP-LU Session Capabilities | RSP(ACTLU) |
| X'0C' | LU Session Services Capabilities | RSP(ACTLU) |
| X'0D' | Mode / Class-of-Service / Virtual-Route-Identifier-List | CINIT |
| X'15' | Network-Qualified Address Pair | CINIT |
| X'1E' | VR-ER Mapping Data | SESSST |
| X'23' | Local-Form Session Identifier | SESSST |
| X'FE' | Control Vector Keys Not Recognized | RSP(CINIT) |

Note: Control vector X'FE' is used to report receipt of one or more unrecognized control vectors, provided that each unrecognized control vector has a key greater thatn X'08'. A negative response indicating sense code 0835—Invalid Parameter (with Pointer Only)—is returned if a request is received with an unrecognized control vector with a key less than or equal to X'08'. When all unrecognized control vectors have keys greater than 8, the receiver responds using a X'FE' control vector that identifies each unrecognized control vector by key; this allows the response sender to indicate that some control vectors have been processed, while others have not.

The control vectors are defined as follows (with zero-origin indexing of the vector bytes—see the individual RU description for the actual displacement within the RU):
Note: When more than one control vector may appear in an RU, the vectors may appear in any order, unless otherwise stated.

SSCP-LU Session Capabilities Control Vector

| | |
|---|---|
| 0 | Key: X'00' |
| 1 | Maximum RU size sent on the normal flow by either half-session: if bit 0 is set to 0, then no maximum is specified and the remaining bits 1-7 are ignored; if bit 0 is set to 1, then the byte is interpreted as X'ab' = a•2**b. (Notice that, by definition, a≥8 and therefore X'ab' is a normalized floating point representation.) See Figure E-1 on page E-8 for all possible values. |
| 2-3 | LU capabilities:<br>bit 0, character-coded capability:<br>0 the SSCP may not send unsolicited character-coded requests; a solicited request is a reply request or a request that carries additional error information to supplement a previously sent negative response or error information after a positive response has already been sent<br>1 the SSCP may send unsolicited character-coded requests<br>bit 1, field-formatted capability:<br>0 the SSCP may not send unsolicited field-formatted requests<br>1 the SSCP may send unsolicited field-formatted requests<br>bits 2-15, reserved |
| 4 | Reserved |

LU-LU Session Services Capabilities Control Vector

| | |
|---|---|
| 0 | Key: X'0C' |
| 1 | Length, in binary, of Vector Data field |
| 2-m | Vector Data |
| 2 | bits 0-3, primary LU capability:<br>0000 PLU capability is inhibited, sessions can neither be queued nor started<br>0001 PLU capability is disabled, sessions can be queued but not started<br>0010 reserved<br>0011 PLU capability is enabled, sessions can be queued or started |
| | bits 4-7, secondary LU capability:<br>0000 SLU capability is inhibited, sessions can neither be queued nor started<br>0001 SLU capability is disabled, sessions can be queued but not started<br>0010 reserved<br>0011 SLU capability is enabled, sessions can be queued or started |
| 3-4 | LU-LU session limit (where a value of 0 means that no session limit is specified) |
| 5-6 | LU-LU session count: the number of LU-LU sessions that are not reset, for this LU, and for which SESSEND will be sent to the SSCP |
| 7 | bit 0, parallel session capability:<br>0 parallel sessions not supported<br>1 parallel sessions supported |
| | bit 1, reserved |
| | bit 2, SESSST capability in RSP(ACTLU) (reserved in NOTIFY):<br>0 SESSST RU is suppressed if SLU<br>1 SESSST RU is sent if SLU |
| | bits 3-7, reserved |
| 8-15(=m) | Retired (set to X'4040404040404040') or omitted |

Mode/ Class-of-Service/ Virtual-Route-Identifier-List Control Vector

| | |
|---|---|
| 0 | Key: X'0D' |
| 1 | Length, in binary, of Vector Data field |
| 2-n | Vector Data |
| 2-9 | Mode name: an eight-character symbolic name (implementation and installation dependent) of type-A symbol string characters that identifies the set of rules and protocols to be used for the session; used by the SSCP(SLU) to select the BIND image that is to be used by the SSCP(PLU) to build the CINIT request |
| 10-17 | COS name: symbolic name of class of service in EBCDIC characters |
| 18-n | Virtual Route Information |
| 18 | Length (in bytes)—binary, not including this length field, of remainder of Virtual Route Information field |
| 19 | Format of virtual route identifier list:<br>X'00' format 0 |
| 20 | Type of virtual route required:<br>X'00' only virtual routes mapping to ER0 from the subarea of the SLU to the subarea of the PLU may be used<br>X'01' virtual routes mapping to any ERN may be used |
| 21 | Number of entries in the virtual route identifier list |
| 22-n | Virtual route identifier list: two-byte (VRN, TPF) entries where VRN is one byte and TPF is one byte |

Network-Qualified Address Pair Control Vector

| | |
|---|---|
| 0 | Key: X'15' |
| 1 | Length, in binary, of Vector Data field |
| 2-n | Vector Data |
| 2-7 | NAU 1 network address |
| 8-13 | NAU 2 network address<br>Note: See the RUs that carry this vector for NAU1/NAU2 definitions and order requirements. |
| 14-21(=n) | Network ID of the subnetwork in which the above addresses are valid<br>Note: If the Network ID contains all space (X'40...40') characters, the network addresses are in the sender's network. |

VR-ER Mapping Data Control Vector

| | |
|---|---|
| 0 | Key: X'1E' |
| 1 | Length, in binary, of Vector Data field |
| 2-n | <u>Vector</u> <u>Data</u> |
| 2 | VRN and TPF data: |
| | bits 0-3, virtual route number (VRN) used by the session indicated in the containing RU |
| | bits 4-5, reserved |
| | bits 6-7, Transmission Priority field (TPF) used by the session indicated in the containing RU |
| 3 | Explicit route data: |
| | bits 0-3, reserved |
| | bits 4-7, outbound ERN for the VRN specified in byte 2, bits 0-3 |
| 4(=n) | Reverse explicit route data: |
| | bits 0-3, reserved |
| | bits 4-7, RERN corresponding to the VRN specified in byte 2, bits 0-3 |

Local Form Session Identifier Control Vector

| | |
|---|---|
| 0 | Key: X'23' |
| 1 | Length, in binary, of Vector Data field |
| 2-p | <u>Vector</u> <u>Data</u> |
| 2 | Format: |
| | X'02' Format 2: FID 2 format session identifier |
| | X'03' Format 3: FID 3 format session identifier |
| | |
| | • For format 2—FID 2 |
| 3-p | Session identifier for Format 2—FID 2 |
| 3 | OAF' from the TH of the BIND request |
| 4 | DAF' from the TH of the BIND request |
| 5(=p) | Flags: |
| | bits 0-5, reserved |
| | bit 6, ODAI field from TH of the BIND request |
| | bit 7, reserved |
| | |
| | • For format 3—FID 3 |
| 3-p | Session identifier for Format 3—FID 3 |
| 3(=p) | LSID from TH of the BIND request |

Control Vector Keys Not Recognized Control Vector

| | |
|---|---|
| 0 | Key: X'FE' |
| 1 | Length, in binary, of Vector Data field |
| 2-n | Vector data: one or more one-byte control vector key values that were not recognized in the corresponding request |

## SESSION KEYS

The following table shows, by key value, the session key and the message-unit structures that can carry the session key.

| Key | Session Key | Applicable Message-Unit Structures |
|---|---|---|
| X'06' | Network name pair | NOTIFY |
| X'07' | Network address pair | BINDF, CINIT, CLEANUP, CTERM, NOTIFY, SESSEND, SESSST, UNBINDF |
| X'0A' | URC | NOTIFY, TERM-SELF |
| X'15' | Network-Qualified address pair | BINDF, CLEANUP, CTERM, NOTIFY, SESSEND, SESSST, UNBINDF |

The session keys are defined as follows (with zero-origin indexing of the key bytes—see the individual RU description for the actual displacement within the RU).

Network Name Pair Session Key

| | |
|---|---|
| 0 | Key: X'06' |
| 1 | Type: X'F3' logical unit |
| 2 | Length, in binary, of PLU (or OLU or LU1) name |
| 3-m | Name in EBCDIC characters (see Note below) |
| m+1 | Type: X'F3' logical unit |
| m+2 | Length, in binary, of SLU (or DLU or LU2) name |
| m+3-n | Name in EBCDIC characters (see Note below) |

Note: The names in this session key consist of type-A symbol string characters.

Network Address Pair Session Key

| | |
|---|---|
| 0 | Key: X'07' |
| 1-2 | Network address of NAU1 |
| 3-4 | Network address of NAU2 |
| | Note: See the RUs that carry this session key for NAU1/NAU2 definitions and order requirements. |

URC Session Key

| | |
|---|---|
| 0 | Key: X'0A' |
| 1 | Length, in binary, of the URC |
| 2-n | URC: LU-defined identifier |

Network-Qualified Address Pair Session Key

| | |
|---|---|
| 0 | Key: X'15' |
| 1 | Length, in binary, of Key Data field |
| 2-21 | KEY Data field |
| 2-7 | NAU1 network address |
| 8-13 | NAU2 network address |
| | Note: See the RUs that carry this session key for NAU1/NAU2 definitions and order requirements. |
| 14-21 | Network ID of the subnetwork in which the above addresses are valid |
| | Note: The Length byte is set to 12 when network ID is not included and to 20 when network ID is included. If the Network ID contains all space (X'40...40') characters, the network addresses are in the sender's network. |

## COMMON SUBVECTORS

The following table shows, by key value, the common subvectors and the message-unit structures that can carry the subvector.

| Key | Subvector | Applicable Message-Unit Structures |
|---|---|---|
| X'10' | Product Set ID | Error Log GDS variable |
| X'11' | Product ID | Error Log GDS variable |

The common subvectors are defined as follows (with zero-origin indexing of the vector bytes—see the specific major vector for the actual displacement within the RU):

### Product Set ID (X'10') Common Subvector

The Product Set ID subvector is a common subvector that identifies one or more products that implement a network component being referenced.

| | |
|---|---|
| 0 | Length (p+1), in binary, of the Product Set ID subvector |
| 1 | Key: X'10' |
| 2 | Retired |
| 3-p | Network product ID consisting of one or more Product ID (X'11') Common Subvectors, as described below, one for each product in the product set implementing the network component indicated in byte 2. Each Product ID (X'11') Common Subvector uniquely identifies a product instance and, optionally, gives its characteristics such as release level, or product being emulated. |

### Product ID (X'11') Common Subvector

The Product ID Common Subvector uniquely identifies a product instance and, optionally, gives its characteristics.

0 Length (q+1), in binary, of the Product ID subvector

1 Key: X'11'

2 bits 0-3, reserved

bits 4-7, product classification:

- X'1' IBM machine
- X'3' IBM or non-IBM machine (not distinguished)
- X'4' IBM programming
- X'9' non-IBM machine
- X'C' non-IBM programming
- X'E' IBM or non-IBM programming (not distinguished)

3-q One or more subfields containing product- and installation-specific information on hardware, microcode, and programming (listed by Key value below and described in detail following):

- X'00' Product Instance Identifier
- X'01' Emulated Product Identifier (hardware)
- X'03' Software Product Version and Release-Level Identifier
- X'05' PTF-Level Data

<u>Note:</u> If byte 2, bits 4-7 (product classification) = X'1', X'3', or X'9', subfields X'03' and X'05' are not supported. If byte 2, bits 4-7 (product classification) = X'4', X'C', or X'E', subfield X'01' is not supported. Subfields X'04', X'09', X'0E', X'0F', X'10', and X'9E' are reserved.

### Product Instance Product ID Subfield Identifier (X'00')

This subfield provides sufficient data to identify the product instance uniquely. For hardware, this normally describes the machine type, plant of manufacture, and serial number. For software, this normally is the program number.

| | |
|---|---|
| 0 | Length (r+1), in binary, of the Product Instance Identifier subfield |
| 1 | Key: X'00' |
| 2 | Format type: |

X'10' product instance is identified by a serial number unique by machine type and if required, IBM plant of manufacture
X'11' product instance is identified by a serial number, unique by machine type, model number, and if required, IBM plant of manufacture.
X'12' product instance is identified by a serial number, unique by machine type and if required, IBM plant of manufacture (as in Format X'10' above). This format provides the model number for the purpose of additional information only.
X'13' product instance is identified by a serial number, unique by machine type, model number, and if required, a 3-digit Corporate Accounting Instruction Code
X'40' product instance software identified by the Program Number
X'41' product instance software identified by the Program Number that contains a 3-byte product modifier

Note: FormatsX'10', X'11', X'12', and X'13' are applicable only to hardware, while formats X'40' and X'41' are applicable only to software. One and only one format can be used in a Product Instance Identifier (X'00') subfield.

3-r Product identification
Note: The originator of a message unit (e.g., PU-MSU, XID) reporting for another product that does not supply information required for the Product Instance Identifier subfield inserts binary 0's into the appropriate fields (except for the Machine Type field where EBCDIC 0's [X'F0'] are inserted) of the Product Identification field to indicate that no identification information is available.

- Format X'10'

3-6 Machine type: four numeric EBCDIC characters
7-8 Serial number modifier—plant of manufacture: two numeric EBCDIC characters or two EBCDIC 0's if plant of manufacture is not required to uniquely identify product instance
9-15(=r) Serial number: seven upper-case alphanumeric EBCDIC characters, right justified, with EBCDIC 0's (X'F0') fill on the left

- Format X'11'

3-6 Machine type: four numeric EBCDIC characters
7-9 Machine model number: three upper-case alphanumeric EBCDIC characters
10-11 Serial number modifier—IBM plant of manufacture: two numeric EBCDIC characters or two EBCDIC 0's if plant of manufacture is not required to uniquely identify product instance
12-18(=r) Serial number: seven upper-case alphanumeric EBCDIC characters, right justified, with EBCDIC 0's (X'F0') fill on the left

- Format X'12'

3-6 Machine type: four numeric EBCDIC characters
7-9 Machine model number: three upper-case alphanumeric EBCDIC characters
10-11 Serial number modifier—IBM plant of manufacture: two numeric EBCDIC characters or two EBCDIC 0's if plant of manufacture is not required to uniquely identify product instance
12-18(=r) Serial number: seven upper-case alphanumeric EBCDIC characters, right justified, with EBCDIC 0's (X'F0') fill on the left

- Format X'13'

3-6 Machine type: four numeric EBCDIC character
7-9 Machine model number: three upper-case alphanumeric EBCDIC characters
10-12 Serial number modifier—Corporate Accounting Instruction Code: three upper-case alphanumeric EBCDIC characters or three EBCDIC 0's if Corporate Accounting Instruction Code is not required to uniquely identify product instance
13-19(=r) Serial number: seven upper-case alphanumeric EBCDIC characters, right justified, with EBCDIC 0's (X'F0') fill on the left

- Format X'40'

3-9 Program Number: seven upper-case alphanumeric EBCDIC characters identifying the software Program Information Department (PID) order number as documented in the IBM product announcement documentation
10-r Customer-specified identifier (type-G symbol-string) to allow differentiation among system-definition options, configurations, or capabilities

- Format X'41'

3-9 Program Number: seven upper-case alphanumeric EBCDIC characters identifying the software Program Information Department (PID) order number as documented in the IBM product announcement documentation
10-12 A product-specified modifier to bytes 3-9 to allow unique product instance identification: three upper-case alphanumeric EBCDIC characters
13-r Customer-specified identifier (in type-G symbol-string) to allow differentiation among system-definition options, configurations, or capabilities

### Emulated Product Identifier (X'01') Product ID Subfield

> This subfield describes the hardware of the product being emulated in sufficient detail to allow problem determination.

| | |
|---|---|
| 0 | Length (r+1), in binary, of the Emulated Product ID subfield |
| 1 | Key: X'01' |
| 2-5 | Machine type of product being emulated: four numeric EBCDIC characters |
| 6-8(=r) | Model number of product being emulated: three upper-case alphanumeric EBCDIC characters |

### Software Product Version and Release Level Identifier (X'03') Product ID Subfield

> This subfield provides the version and release-level number of the software running in the product.

| | |
|---|---|
| 0 | Length (r+1), in binary, of the Software Product Release or Level Identifier subfield |
| 1 | Key: X'03' |
| 2-r | Software version and release-level identifier (upper-case alphanumeric EBCDIC characters) identifying the software version and release-level number as documented in the IBM product announcement documentation for IBM products |

### PTF-Level Data (X'05') Product ID Subfield

> This subfield provides the PTF-level related data for the product (software).

| | |
|---|---|
| 0 | Length (r+1), in binary, of the PTF-Level Data subfield |
| 1 | Key: X'05' |
| 2-r | PTF-level product-defined data for the software (in type-AE symbol-string) identifying the software PTF level data as documented in the IBM product announcement documentation for IBM products |

# APPENDIX F. PROFILES

## FUNCTION MANAGEMENT (FM) PROFILES

This section describes the function management (FM) profiles and their use for LU 6.2 sessions. Profile numbers not shown are reserved in these sessions.

Note: If the FM Usage field in BIND specifies a value for a parameter, that value is used unless it conflicts with a value specified by the FM profile. The FM profile overrides the FM Usage field.

### FM PROFILE 0

Profile 0 (used on SSCP-LU sessions) specifies the following session rules:

Primary and secondary half-sessions use immediate request mode and immediate response mode.

Only single-RU chains allowed.

Primary and secondary half-session chains indicate definite response.

No compression.

Primary half-session sends no DFC RUs.

Secondary LU half-session may send LUSTAT.

No brackets.

No FM headers.

No alternate code.

Normal-flow send/receive mode is full-duplex.

## FM PROFILE 6

Profile 6 (used on SSCP-LU sessions) specifies the following session rules:

Primary and secondary half-sessions use delayed request mode and delayed response mode.

Only single-RU chains allowed.

Primary and secondary half-session chains may indicate definite response, exception response, or no response.

No compression.

Primary half-session sends no DFC RUs.

Secondary half-session may send LUSTAT.

No brackets.

No FM headers.

No alternate code.

Normal-flow send/receive mode is full-duplex.

## FM PROFILE 19

Profile 19 (used on LU-LU sessions) specifies the following session rules:

Primary LU half-session and secondary LU half-session use immediate request and immediate response mode.

Multiple RU chains allowed.

Primary LU half-session and secondary LU half-session chains indicate definite or exception response.

No compression.

Primary and secondary half-sessions support the following DFC functions:

SIGNAL
LUSTAT
BIS
RTR

Brackets are used.

FM headers (types 5 and 7 only) are allowed.

Conditional termination for brackets (specified by CEB) will be used--primary and secondary half-sessions may send CEB.

The following combinations of RQE, RQD, CEB, and CD are allowed on end-chain RUs:

RQE*, CD, ¬CEB
RQD2, CD, ¬CEB
RQD3, CD, ¬CEB
RQE1, ¬CD, CEB
RQD*, ¬CD, CEB
RQD*, ¬CD, ¬CEB

Normal-flow send/receive mode is half-duplex flip-flop.

Half-duplex flip-flop reset state is <u>send</u> for the primary LU half-session and <u>receive</u> for the secondary LU half-session after RSP(BIND).

Symmetric responsibility for recovery.

Contention winner/loser polarity is negotiated at BIND time; the contention winner is the first speaker and the contention loser is the bidder.

The only FM Usage field defining options for Profile 19 is Contention Winner/Loser.

## FM PROFILE VS. TYPE OF SESSION

The following table specifies which FM profiles may be used with each type of session.

| FM Profile | Type of Session | |
|---|---|---|
| | SSCP-LU | LU-LU |
| 0 | yes | no |
| 6 | yes | no |
| 19 | no | yes |

LUs in the same node as an SSCP use FM profile 6
for the SSCP-LU session; otherwise, the LU uses FM profile 0.

## TRANSMISSION SERVICES (TS) PROFILES

This section describes the transmission services (TS) profiles and their use for LU 6.2 sessions. Profile numbers not shown are reserved in these sessions.

Note: If the TS Usage field in BIND specifies a value for a parameter, that value is used unless it conflicts with a value specified by the TS profile. The TS profile overrides the TS Usage field.

### TS PROFILE 1

Profile 1 (used on SSCP-LU sessions) specifies the following session rules:

No pacing.

Identifiers rather than sequence numbers are used on the normal flows.

SDT, CLEAR, RQR, STSN, and CRV are not supported.

Maximum RU size on the normal flow for either half-session is 256, unless a different value is specified in RSP(ACTLU).

There is no TS Usage field associated with this profile.

### TS PROFILE 7

Profile 7 (used on LU-LU sessions) specifies the following session rules:

Primary-to-secondary and secondary-to-primary normal flows are optionally paced.

Sequence numbers are used on the normal flows.

SDT, CLEAR, RQR, and STSN are not supported.

CRV is supported when session-level cryptography is selected (via a BIND parameter).

The TS Usage subfields in BIND defining the options for this profile are:

Pacing window sizes

Maximum RU sizes on the normal flows

## TS PROFILE VS. TYPE OF SESSION

The following table specifies which TS profile may be used with each type of session.

| TS Profile | Type of Session | |
|---|---|---|
| | SSCP-LU | LU-LU |
| 1 | yes | no |
| 7 | no | yes |

# APPENDIX G. SENSE DATA

The sense data included with an EXCEPTION REQUEST (EXR), a negative response, an UNBIND request, a function management header type 7 (FMH-7), or a send or receive check is a four-byte field (see Figure G-1) that generally includes a one-byte category value, a one-byte modifier value, and two bytes of sense code specific information, whose format is defined along with the sense code definition, below.

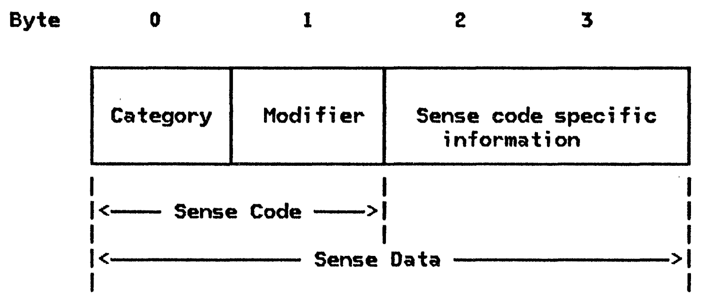


Figure G-1. Sense Data Format

Together, the category byte 0, the modifier byte 1, and the sense code specific bytes 2 and 3 hold the sense data defined for the exception condition that has occurred.

The following categories are defined; all others are reserved:

| VALUE | CATEGORY |
|---|---|
| X'00' | User Sense Data Only |
| X'08' | Request Reject |
| X'10' | Request Error |
| X'20' | State Error |
| X'40' | Request Header (RH) Usage Error |
| X'80' | Path Error |

The category User Sense Data Only (X'00') allows the end users to exchange sense data in bytes 2-3 for conditions not defined by SNA within the other categories (and perhaps unique to the end users involved). The modifier value is also X'00'. In earlier versions of SNA, user data (as well as implementation-specific data) generally could be carried in bytes 2-3 for all categories. This is no longer permitted. Bytes 2-3 are used only for SNA-defined conditions for non-zero categories.

The sense codes for the other categories are discussed below.

## REQUEST REJECT (CATEGORY CODE = X'08')

This category indicates that the request was delivered to the intended half-session component and was understood and supported, but not executed.

Category and modifier (in hexadecimal):

0801 Resource Not Available: The LU, PU, or link specified in an RU is not available.

0805 Session Limit Exceeded: The requested session cannot be activated, as one of the NAUs is at its session limit (e.g., LU-LU session limit, or [LU, mode] session limit). Applies to ACTCDRM, INIT, BIND, and CINIT requests.

Bytes 2 and 3 may contain the following sense code specific information:

0000 No specific code applies.

0001 If accepted, the BIND request would prevent either the receiving LU or the sending LU from activating the number of contention winner sessions to the partner LU that were agreed upon during a change-number-of-sessions procedure.

0806 Resource Unknown: The request contained a name or address not identifying a PU, LU, link, or link station known to the receiver.

0809 Mode Inconsistency: The requested function cannot be performed in the present state of the receiver.

080A Permission Rejected: The receiver has denied an implicit or explicit request of the sender; when sent in response to BIND, it implies either that the secondary LU will not notify the SSCP when a BIND can be accepted, or that the SSCP does not recognize the NOTIFY vector key X'0C'. (See the X'0845' sense code for a contrasting response.)

080E NAU Not Authorized: The requesting NAU does not have access to the requested resource.

080F End User Not Authorized: The requesting end user does not have access to the requested resource.

Bytes 2 and 3 may contain the following sense code specific information:

0000 No specific code applies.

6051 Access Security Information Invalid: The request specifies an Access Security Information field that is unacceptable to the receiver; for security reasons, no further detail on the error is provided. This sense data is sent in FMH-7 or UNBIND.

0812 Insufficient Resource: Receiver cannot act on the request because of a temporary lack of resources.

0813 Bracket Bid Reject--No RTR Forthcoming: BID (or BB) was received while the first speaker was in the in-bracket state, or while the first speaker was in the between-brackets state and the first speaker denied permission. RTR will not be sent.

0814 Bracket Bid Reject--RTR Forthcoming: BID (or BB) was received while the first speaker was in the in-bracket state, or while the first speaker was in the between-brackets state and the first speaker denied permission. RTR will be sent.

0815 Function Active: A request to activate a network element or procedure was received, but the element or procedure was already active.

0816 Function Inactive: A request to deactivate a network element or procedure was received, but the element or procedure was not active.

0819 RTR Not Required: Receiver of READY TO RECEIVE has nothing to send.

081A Request Sequence Error: Invalid sequence of requests.

0820 Control Vector Error: Invalid data for the control vector specified by the target network address and key.

0823 Unknown Control Vector: The control vector specified by a network address and key is not known to the receiver.

0824 Logical Unit of Work Aborted: The current unit of work has been aborted; when sync point protocols are in use, both sync point managers are to revert to the previously committed sync point.

Bytes 2 and 3 may contain the following sense code specific information:

0000 For LU 6.2, Backout Initiated: A transaction program or its LU has initiated backout. The protected resources for the distributed logical unit of work are to be restored to the previously committed sync point. This sense data is sent only in FMH-7.

For non-LU 6.2, no specific code applies.

082C Resource-Sharing Limit Reached: The request received from an SSCP was to activate a half-session, a link, or a procedure, when that resource was at its share limit.

0835 Invalid Parameter (with Pointer Only): The request contained a fixed- or variable-length field whose contents are invalid or not supported by the NAU that received the request. Bytes 2 and 3 are used for sense code specific information:

nnnn Bytes 2 and 3 contain a two-byte binary count that indexes (zero-origin) the first byte of the fixed- or variable-length field having invalid contents.

0836 PLU/SLU Specification Mismatch: For a specified LU-LU session, both the origin LU (OLU) and the destination LU (DLU) have only the primary capability or have only the secondary capability.

0837 Queuing Limit Exceeded: For an LU-LU session initiation request (INIT, CDINIT, or INIT-OTHER-CD) specifying (1) Initiate or Queue (if Initiate not possible) or (2) Queue Only, the queuing limit of either the OLU or the DLU, or both, was exceeded.

0839 LU-LU or SSCP-LU Session Being Taken Down: At the time an LU-LU session initiation or termination request is received, the SSCP of at least one of the LUs is either processing a CDTAKED request or is in the process of deactivating the associated SSCP-LU session.

083A LU Not Enabled: At the time an LU-LU session initiation request is received at the SSCP, at least one of the two LUs, although having an active session with its SSCP, is not ready to accept CINIT or BIND requests.

0842 SSCP-SSCP Session Not Active: The SSCP-SSCP session, which is required for the processing of a network services request, is not active; e.g., at the time an LU-LU session initiation or termination request is received, at least one of the following conditions exists:

- The SSCP of the ILU and the SSCP of the OLU do not have an active session with each other, and therefore INIT-OTHER-CD cannot flow.
- The SSCP of the OLU and the SSCP of the DLU do not have an active session with each other, and therefore CDINIT or CDTERM cannot flow.

0845 Permission Rejected--SSCP Will Be Notified: The receiver has denied an implicit or explicit request of the sender; when sent in response to BIND, it implies that the secondary LU will notify the SSCP (via NOTIFY vector key X'0C') when a BIND can be accepted, and the SSCP of the SLU supports the notification. (See the X'080A' sense code for a contrasting response.)

0846 ERP Message Forthcoming: The received request was rejected for a reason to be specified in a forthcoming request.

0848 Cryptography Function Inoperative: The receiver of a request was not able to decipher the request because of a malfunction in its cryptography facility.

084B Requested Resources Not Available: Resources named in the request, and required to honor it, are not currently available. It is not known when the resources will be made available.

Bytes 2 and 3 may contain the following sense code specific information:

0000 No specific code applies.

6031 Transaction Program Not Available--Retry Allowed: The FMH-5 Attach command specifies a transaction program that the receiver is unable to start. Either the program is not authorized to run or the resources to run it are not available at this time. The condition is temporary. The sender is responsible for subsequent retry. This sense data is sent only in FMH-7.

084C Permanent Insufficient Resource: Receiver cannot act on the request because resources required to honor the request are permanently unavailable. The sender should not retry immediately because the situation is not transient.

Bytes 2 and 3 may contain the following sense code specific information:

0000 For LU 6.2, Transaction Program Not Available -- No Retry: The FMH-5 Attach command specifies a transaction program that the receiver is unable to start. The condition is not temporary. The sender should not retry immediately. This sense data is sent only in FMH-7.

For non-LU 6.2, no additional information is specified.

084D Invalid Session Parameters--BF: Session parameters were not valid or were unacceptable by the boundary function. Bytes 2 and 3 following the sense code contain a binary count that indexes (zero origin) the first byte of the fixed- or variable-length field having invalid contents.

084E Invalid Session Parameters--PRI: A positive response to an activation request (e.g., BIND) was received and was changed to a negative response because of invalid session parameters carried in the response. The services manager receiving the response will send a deactivation request for the corresponding session.

0852 Duplicative Session Activation Request: Two session activation requests have been received with related identifiers. The relationship of the identifiers and the resultant action varies by request. For BIND, it means that the BIND request was received with the same session instance identifier (in the structured subfield X'03' of the User Data field) as an active session's; the current request is refused.

0856 SSCP-SSCP Session Lost: Carried in the Sense Data field in a NOTIFY (Third-Party Notification vector, X'03') or -RSP(INIT_OTHER) sent to an ILU to indicate that the activation of the LU-LU session is uncertain because the SSCP(ILU)-SSCP(OLU) session has been lost. (Another sense code, X'0842', is used when it is known that the LU-LU session activation cannot be completed.)

0857 SSCP-LU Session Not Active: The SSCP-LU session, required for the processing of a request, is not active; e.g., in processing REQECHO, the SSCP did not have an active session with the target LU named in the REQECHO RU.

0859 REQECHO Data Length Error: The specified length of data to be echoed (in REQECHO) violates the maximum RU size limit for the target LU.

0861 Invalid COS Name: The class of service (COS) name, either specified by the ILU or generated by the SSCP of the SLU from the mode table is not in the "COS name to VR identifier list" table used by the SSCP of the PLU.

Bytes 2 and 3 may contain the following sense code specific information:

0000 COS name was generated by the SSCP.

0001 COS name was generated by the ILU.

0003 CDINIT request (or response) contains a Session Initiation control vector that has class of service (COS) name fields that have not been properly specified. If the RU is a positive response, it is changed into a negative response and sent to the request sender; a CDTERM is sent to the CDINIT response sender. (This is to cover a system definition error in the event a gateway SSCP downstream from another gateway SSCP receives a CDINIT or RSP(CDINIT) without valid information in the appropriate COS name fields of the Session Initiation control vector.)

0864 Function Abort: The conversation was terminated abnormally. Other terminations may occur after repeated reexecutions; the request sender is responsible to detect such a loop.

Bytes 2 and 3 may contain the following sense code specific information:

0000 For LU 6.2, Premature Conversation Termination: The conversation is terminated abnormally; for example, the transaction program may have issued a DEALLOCATE_ABEND verb, or the program may have terminated (normally or abnormally) without explicitly terminating the conversation. This sense data is sent only in FMH-7.

For non-LU 6.2, no additional information is specified.

0001 System Logic Error--No Retry: A system logic error has been detected. No retry of the conversation should be attempted. This sense data is sent only in FMH-7.

0002 Excessive Elapsed Time--No Retry: Excessive time has elapsed while waiting for a required action or event. For example, a transaction program has failed to issue a conversation-related protocol boundary verb. No retry of the conversation should be attempted. This sense data is sent in UNBIND when there is no chain to respond to; otherwise, it is sent in FMH-7.

0889 Transaction Program Error: The transaction program has detected an error.

This sense code is sent only in FMH-7.

Bytes 2 and 3 may contain the following sense code specific information:

0000 Program Error--No Data Truncation: The transaction program sending data detected an error but did not truncate a logical record.

Program Error--Purging: The transaction program receiving data detected an error. All remaining information, if any, that the receiving program had not yet received, and that the sending program had sent prior to being notified of the error, is discarded.

0001 Program Error--Data Truncation: The transaction program sending data detected an error and truncated the logical record it was sending.

0100 Service Error--No Data Truncation: The presentation services component for mapped conversations detected an error while sending data but did not truncate a logical record.

Service Error--Purging: The presentation services component for mapped conversations detected an error while receiving data. All remaining information, if any, that the receiving mapped-conversations component had not yet received, and that the sending component had sent prior to being notified of the error, is discarded.

0101 Service Error--Data Truncation: The presentation services component for mapped conversations detected an error while sending data and truncated the logical record it was sending.

088B BB Not Accepted--BIS Reply Requested: Sent in response to a BB (either an LUSTAT bid or an Attach) to indicate that the receiver has sent a BIS request and wishes to terminate the session without processing any more conversations, but without sending an UNBIND. A BIS reply is requested so that the negative response sender may send a normal UNBIND. This sense code is sent only by LUs not supporting change-number-of-session protocols.

088C Missing Control Vector: The RU did not contain a control vector which was expected to appear. The first byte of the sense code specific field contains the hex code of the control vector first noticed to be missing. If more than one control vector is missing, only the first omission is reported. The second byte of the sense code specific field is set to X'00'.

## REQUEST ERROR (CATEGORY CODE = X'10')

This category indicates that the RU was delivered to the intended NAU component, but could not be interpreted or processed. This condition represents a mismatch of NAU capabilities.

Category and modifier (in hexadecimal):

1001 RU Data Error: Data in the request RU is not acceptable to the receiving component; for example, a character code is not in the set supported, a formatted data field is not acceptable to presentation services, a value specified in the length field (LL) of a structured field is invalid, or a required name in the request has been omitted.

1002 RU Length Error: The request RU was too long or too short.

1003 Function Not Supported: The function requested is not supported. The function may have been specified by a formatted request code, a field in an RU, or a control character.

1005 Parameter Error: A parameter modifying a control function is invalid, or outside the range allowed by the receiver.

1007 Category Not Supported: DFC, SC, NC, or FMD request was received by a half-session not supporting any requests in that category; or an NS request byte 0 was not set to a defined value, or byte 1 was not set to an NS category supported by the receiver.

1008 Invalid FM Header: The FM header was not understood or translatable by the receiver, or an FM header was expected but not present. This sense code is sent in FMH-7 or UNBIND.

Bytes 2 and 3 may contain the following sense code specific information:

0000 Reserved.

200E Invalid Concatenation Indicator: The concatenation indicator is <u>on</u> but concatenation is not allowed.

201D FM Header and Associated Data Mismatch: The FM header indicated associated data would or would not follow (e.g., FM header 7 followed by log data, or FM header 5 followed by program initialization parameters), but this indication was in error; or a previously received RU [e.g., -RSP(0846)] implied that an FM header would follow, but none was received.

4001 Invalid FM Header Type: The type of the FM header is other than 5 or 7.

6000 FM Header Length Not Correct: The value in the FM header Length field differs from the sum of the lengths of the subfields of the FM header.

6005 Access Security Information Length Field Not Correct: The value in the Access Security Information Length field differs from the sum of the lengths of the Access Security Information subfields.

6009 Invalid Parameter Length. The field that specifies the length of fixed-length parameters has an invalid setting.

600B Unrecogized FM Header Command Code: The partner LU received an FM header command code that it does not recognize. For LU 6.2, this sense data is sent only in FMH-7.

6011 Invalid Logical Unit of Work: The LUW Length field (in a Compare States GDS variable or an FMH-5) is incorrect or the LUW is invalid or a LUWID is not present but is required by the setting of the synchronization level field.

6021 Transaction Program Name Not Recognized: The FMH-5 Attach command specifies a transaction program name that the receiver does not recognize. This sense data is sent only in FMH-7.

6031 PIP Not Allowed: The FMH-5 Attach command specifies program initialization parameter (PIP) data is present but the receiver does not support PIP data for the specified transaction program. This sense data is sent only in FMH-7.

6032 PIP Not Specified Correctly: The FMH-5 Attach command specifies a transaction program name that requires program initialization parameter (PIP) data and either the FMH-5 specifies PIP data is not present or the number of PIP subfields present does not agree with the number required for the program. This sense data is sent only in FMH-7.

6034 Conversation Type Mismatch: The FMH-5 Attach command specifies a conversation type that the receiver does not support for the specified transaction program. This sense data is sent only in FMH-7.

6040 Invalid Attach Parameter: A parameter in the FMH-5 Attach command conflicts with the statement of LU capability previously provided in the BIND negotiation.

6041 Synchronization Level Not Supported: The FMH-5 Attach command specifies a synchronization level that the receiver does not support for the specified transaction program. This sense data is sent only in FMH-7.

## <u>STATE ERROR (CATEGORY CODE = X'20')</u>

This category indicates a sequence number error, or an RH or RU that is not allowed for the receiver's current session control or data flow control state. These errors prevent delivery of the request to the intended half-session component.

Category and modifier (in hexadecimal):

| | |
|---|---|
| 2001 | Sequence Number: Sequence number received on normal-flow request was not 1 greater than the last. |
| 2002 | Chaining: Error in the sequence of the chain indicator settings (BCI, ECI), such as first, middle, first. |
| 2003 | Bracket: Error resulting from failure of sender to enforce bracket rules for session. (This error does not apply to contention or race conditions.) |
| 2004 | Direction: Error resulting from a normal-flow request received while the half-duplex flip-flop state was not Receive. |
| 2008 | No Begin Bracket: An FMD request specifying BBI=BB was received after the receiver had previously received a BRACKET INITIATION STOPPED request. |
| 2009 | Session Control Protocol Violation: An SC protocol has been violated; a request, allowed only after a successful exchange of an SC request and its associated positive response, has been received before such successful exchange has occurred (e.g., an FMD request has preceded a required CRYPTOGRAPHY VERIFICATION request). The request code of the particular SC request or response required, or X'00' if undetermined, appears in the fourth byte of the sense data. |
| 200A | Immediate Request Mode Error: The immediate request mode protocol has been violated by the request. |
| 200B | Queued Response Error: The Queued Response protocol has been violated by a request, i.e., QRI=¬QR when an outstanding request had QRI=QR. |
| 200E | Response Correlation Error: A response was received that cannot be correlated to a previously sent request. |
| 200F | Response Protocol Error: A violation has occurred in the response protocol; e.g., a +RSP to an RQE chain was generated. |
| 2010 | BIS Protocol Error: A BIS protocol error was detected; e.g., a BIS request was received after a previous BIS was received and processed. |
| 2011 | Pacing Error: A normal-flow request is received by a half-session after the pacing count has been reduced to 0 and before a pacing response has been sent. |
| 2012 | Invalid Sense Code Received: A negative response was received that contains an SNA-defined sense code that cannot be used for the sent request. |

## RH USAGE ERROR (CATEGORY CODE = X'40')

This category indicates that the value of a field or combination of fields in the RH violates architectural rules or previously selected BIND options. These errors prevent delivery of the request to the intended half-session component and are independent of the current states of the session. They may result from the failure of the sender to enforce session rules. Detection by the receiver of each of these errors is optional.

Category and modifier (in hexadecimal):

| | |
|---|---|
| 4003 | BB Not Allowed: The Begin Bracket indicator (BBI) was specified incorrectly, e.g., BBI=BB with BCI=¬BC. |
| 4004 | CEB or EB Not Allowed: The Conditional End Bracket indicator (CEBI) or End Bracket indicator (EBI) was specified incorrectly, e.g., CEBI=CEB when ECI=¬EC or EBI=EB with BCI=¬BC, or by the primary half-session when only the secondary may send EB, or by the secondary when only the primary may send EB. |
| 4005 | Incomplete RH: Transmission shorter than full TH-RH. |

| | |
|---|---|
| 4006 | Exception Response Not Allowed: Exception response was requested when not permitted. |
| 4007 | Definite Response Not Allowed: Definite response was requested when not permitted. |
| 4008 | Pacing Not Supported: The Pacing indicator was set on a request, but the receiving half-session or boundary function half-session does not support pacing for this session. |
| 4009 | CD Not Allowed: The Change Direction indicator (CDI) was specified incorrectly, e.g., CDI=CD with ECI=¬EC , or CDI=CD with EBI=EB. |
| 400A | No-Response Not Allowed: No-response was specified on a request when not permitted. (Used only on EXR.) |
| 400B | Chaining Not Supported: The chaining indicators (BCI and ECI) were specified incorrectly, e.g., chaining bits indicated other than (BC,EC), but multiple-request chains are not supported for the session or for the category specified in the request header. |
| 400C | Brackets Not Supported: The bracket indicators (BBI, CEBI, and EBI) were specified incorrectly, e.g., a bracket indicator was set (BBI=BB, CEBI=CEB, or EBI=EB), but brackets are not used for the session. |
| 400D | CD Not Supported: The Change-Direction indicator was set, but is not supported. |
| 400F | Incorrect Use of Format Indicator: The Format indicator (FI) was specified incorrectly, e.g., FI was set with BCI=¬BC, or FI was not set on a DFC request. |
| 4010 | Alternate Code Not Supported: The Code Selection indicator (CSI) was set when not supported for the session. |
| 4011 | Incorrect Specification of RU Category: The RU Category indicator was specified incorrectly, e.g., an expedited-flow request or response was specified with RU Category indicator = FMD. |
| 4012 | Incorrect Specification of Request Code: The request code on a response does not match the request code on its corresponding request. |
| 4013 | Incorrect Specification of (SDI, RTI): The Sense Data Included indicator (SDI) and the Response Type indicator (RTI) were not specified properly on a response. The proper value pairs are (SDI=SD, RTI=negative) and (SDI=¬SD, RTI=positive). |
| 4014 | Incorrect Use of (DR1I, DR2I, ERI): The Definite Response 1 indicator (DR1I), Definite Response 2 indicator (DR2I), and Exception Response indicator (ERI) were specified incorrectly, e.g., a SIGNAL request was not specified with DR1I=DR1, DR2I=¬DR2, and ERI=¬ER. |
| 4015 | Incorrect Use of QRI: The Queued Response indicator (QRI) was specified incorrectly, e.g., QRI=QR on an expedited-flow request. |
| 4016 | Incorrect Use of EDI: The Enciphered Data indicator (EDI) was specified incorrectly, e.g., EDI=ED on a DFC request. |
| 4017 | Incorrect Use of PDI: The Padded Data indicator (PDI) was specified incorrectly, e.g., PDI=PD on a DFC request. |
| 4018 | Incorrect Setting of QRI with Bidder's BB: The first speaker half-session received a BB chain requesting use of a session (via LUSTAT(X'0006')), but the QRI was specified incorrectly, i.e., QRI = ¬QR. |
| 4019 | Incorrect Indicators with Last-In-Chain Request: A last-in-chain request has specified incompatible RH settings, e.g., RQE*, CEBI=¬CEB, and CDI=¬CD. |
| 4021 | QRI Setting in Response Different From That in Request: The QRI setting in the response differs from the QRI setting in the corresponding request. |

## PATH ERROR (CATEGORY CODE = X'80')

This category indicates that the request could not be delivered to the intended receiver, because of a path outage, an invalid sequence of activation requests, or one of the listed path information unit (PIU) errors. (Some PIU errors fall into other categories, e.g., sequence number errors are category X'20'.) A path error received while the session is active generally indicates that the path to the session partner has been lost.

Category and modifier (in hexadecimal):

| | |
|---|---|
| 8001 | Intermediate Node Failure: Machine or program check in a node providing intermediate routing function. A response may or may not be possible. |
| 8002 | Link Failure: Data link failure. |
| 8003 | NAU Inoperative: The NAU is unable to process requests or responses, e.g., the NAU has been disrupted by an abnormal termination. |
| 8004 | Unrecognized Destination: A node in the path has no routing information for the destination specified either by the SLU name in a BIND request or by the TH.<br><br>Bytes 2 and 3 may contain the following sense code specific information:<br><br>0000 No specific code applies.<br><br>0001 A request was received by a gateway function that could not be rerouted because of invalid or incomplete routing information. |
| 8005 | No Session: No half-session is active in the receiving end node for the indicated origination-destination pair, or no boundary function half-session component is active for the origin-destination pair in a node providing the boundary function. A session activation request is needed.<br><br>Bytes 2 and 3 may contain the following sense code specific information:<br><br>0000 No specific code applies.<br><br>0001 The receiver received a request other than session control request when no LU-LU session was active.<br><br>0002 The receiver received a request other than session control request when no LU-SSCP session was active.<br><br>0003 The receiver received a session control request other than BIND/UNBIND when no LU-LU session was active.<br><br>0004 The receiver received an UNBIND when no LU-LU session was active.<br><br>0005 The receiver received a session control request other than ACTLU/DACTLU for the LU-SSCP session when no LU-SSCP session was active.<br><br>0006 The receiver received DACTLU when no LU-SSCP session was active. |
| 8006 | Invalid FID: Invalid FID for the receiving node. (Note 1) |
| 8007 | Segmenting Error: First BIU segment had less than 10 bytes; or mapping field sequencing error, such as first, last, middle; or segmenting not supported and MPF not set to 11. (Note 2) |
| 8008 | PU Not Active: The SSCP-PU secondary half-session in the receiving node has not been activated and the request was not ACTPU for this half-session; for example, the request was ACTLU from an SSCP that does not have an active SSCP-PU session with the PU associated with the addressed LU. |
| 8009 | LU Not Active: The destination address specifies an LU for which the SSCP-LU secondary half-session has not been activated and the request was not ACTLU. |
| 800B | Incomplete TH: Transmission received was shorter than a TH. (Note 1) |
| 800C | DCF Error: Data Count field inconsistent with transmission length. |
| 800E | Unrecognized Origin: The origin address specified in the TH was not recognized. |
| 800F | Invalid Address Combination: The (DAF',OAF') (FID2) combination or the LSID (FID3) specified an invalid type of session, e.g., a PU-LU combination. |
| 8010 | Segmented RU Length Error: An RU was found to exceed a maximum length, or required buffer allocation that might cause future buffer depletion. |

8013 COS Not Available: A session activation request cannot be satisfied because none of the virtual routes requested for the session is available.

Bytes 2 and 3 may contain the following sense code specific information:

Byte 2 indicates the environment in which the failure was detected:

| | |
|---|---|
| 00 | Single network |
| 01 | Interconnected network: Failure was detected at a node in a subnetwork other than that of the NAU sending the activation request. |

Byte 3 indicates the reason for the session-activation failure:

| | |
|---|---|
| 00 | No Specific Code applies: This means an error occured, but none of the conditions listed below applies. |
| 01 | No Mapping Specified: A session activation request cannot be satisfied because for each VR in the VR identifier list for the session, no VR to ER mapping is specified. |
| 02 | No Explicit Routes Defined: A session activation request cannot be satisfied because each VR in the VR identifier list for the session maps to a corresponding ER that is not defined. |
| 03 | No VR Resource Available: A session activation request cannot be satisfied because each VR specified in the VR identifier list for the session requires a node resource that is not available. |
| 04 | No Explicit Routes Operative: A session activation request cannot be satisfied because no underlying ER is operative for any VR specified in the VR identifier list for the session. |
| 05 | No Explicit Route Can Be Activated: A session activation request cannot be satisfied because no VR specified in the VR identifier list for the session mapped to a defined and operative ER that could be activated. |
| 06 | No Virtual Route Can Be Activated: A session activation request cannot be satisfied because no VR specified in the VR identifier list for the session can be activated by the PU, though for at least one VR an underlying ER is defined, operative, and activated. |
| 07 | No Virtual Route Identifier List Available: A session activation request cannot be satisfied because a VR identifier list is not available. |

Note: If none of the virtual routes specified in the VR identifier list for the session is active or can be activated, the reported reason is set based on a hierarchy of failure events. The "highest" of the failures that occurred within the set of virtual routes is returned on the response. For example, if the VR manager receives a negative response to an NC_ACTVR request for a VR specified in the VR identifier list and for all other VRs in the list no VR to ER mapping is specified, then reason X'06' is reported. The hierarchy of the failure reasons is in ascending numeric order (e.g., reason X'02' is higher than reason X'01').

Notes:

1. It is generally not possible to send a response for this exception condition, since information (FID, addresses) required to generate a response is not available. It is logged as an error if this capability exists in the receiver.

2. If segmenting is not supported, a negative response is returned for the first segment only, since this contains the RH. Subsequent segments are discarded.

# APPENDIX H. FM HEADER AND LU SERVICES COMMANDS

Throughout this appendix the same symbol-string types are used as defined in "Appendix E. Request-Response Unit (RU) Formats". The symbol-string types define the character sets that LU's and transaction programs use to specify the symbol strings used in RU's. Depending on the symbol string, support is defined by either a single type or a range of types.

The range of types identifies a lower and upper bound for the set of characters that an implementation can support for a specific symbol string. The type bounding the lower end of a range is always a subset of the type bounding the upper end of the range. (In order of increasing generality, the types are ranked: A, AE, GR, and G.) The support a product provides for each of these symbol strings is implementation-defined within the range.

Figure H-1 defines the send and receive support for each symbol string in terms of the symbol-string types. Where support is defined to be within a range of types, the range is given as "lower-type<->upper-type," which identifies the lower and upper bounds of the range.

| Symbol String | Type | |
|---|---|---|
| | Send Support | Receive Support |
| Network ID | A | A |
| LU Name | A | A |
| Fully Qualified LU Name [1] | A.A | A.A |
| Mode Name | A | A |
| Transaction Program Name [2] | AE<->GR | A<->GR |
| Access Security Subfields | AE<->GR | A<->GR |
| Program Initialization Parameters (PIP) | G | G |
| Session Instance ID | G | G |
| Map Name | A<->GR | A<->GR |

NOTES:

1. The fully qualified LU name consists of two symbol strings of type A concatenated by a period (.). The lefthand symbol string represents the network ID; the righthand symbol string represents the network LU name. The period is not part of the network ID or the network LU name.

2. The first character of an SNA-defined service transaction program name is not a type-A, type-AE, or type-GR character; it is a character ranging in value from X'00' through X'3F'. A list of SNA-defined service programs is given in "SNA-Defined Transaction Program Names" on page H-14.

Figure H-1. Symbol-String Types

## SYMBOL-STRING LENGTH

The symbol-string length represents the number of characters a symbol string can contain. Three symbol-string lengths are defined:

- Minimum specification length: the minimum number of characters that a transaction program is allowed to use to specify the symbol string. For some symbol strings, the minimum specification length is 0. Zero-length strings are valid symbol strings and are subject to the same usage conditions as nonzero-length strings that fulfill the definition of the specific symbol-string type (or range of types) allowed.

- Maximum send support: the maximum number of characters that every implementation can send in the symbol string.

- Maximum receive support: the maximum number of characters that every implementation can receive in the symbol string.

The maximum send or receive support for a symbol-string's length is defined either by a single value or within a range of values, depending on the symbol string:

- The single value is the maximum number of characters in a symbol string that every implementation can send or receive.

- The range of values represents a lower and upper bound of the maximum number of characters in a symbol string that an implementation can send or receive. The specific maximum number of characters an implementation can send or receive for each of these symbol strings is implementation-defined within the range. Compatibility in the maximum lengths allowed by sender and receiver is a concern of system definition and program design.

Figure H-2 on page H-3 defines the product maximum send and receive support for each symbol string in terms of the symbol-string lengths. Where support is defined to be within a range of values, the range is given as "lower-value<->upper-value," which identifies the lower and upper bounds of the range.

The variable to which a type-A, type-AE, or type-GR symbol string is assigned may be longer than the symbol string; in this case, the symbol string is left-justified within the variable and the variable is filled out to the right with space (X'40') characters. Space characters, if present, are not part of the symbol string. If the symbol string is formed from the concatenation of two or more individual symbol strings, such as the fully qualified LU name, the concatenated symbol string as a whole is left-justified within the variable and the variable is filled out to the right with space characters. Space characters, if present, are not part of the concatenated symbol string.

| Symbol String | Length | | |
|---|---|---|---|
| | Minimum Specification | Maximum Send Support | Maximum Receive Support |
| Network ID | 0 | 8 | 8 |
| LU Name | 0 | 8 | 8 |
| Fully Qualified LU Name | 1 | 17 | 17 |
| Mode Name | 0 | 8 | 8 |
| Transaction Program Name | 1 | 8<->64 | 8<->64 |
| Access Security Subfields | 0 | 8<->10 | 0<->10 |
| PIP Subfields [1] | 0 | 64<->≥64 [2] | 64<->≥64 [3] |
| Session Instance ID | 2 | 8 | 8 |
| Map Name | 0 [4] | 8<->64 | 7<->64 |

NOTES:

1. Support of PIP subfields is optional, and send support is independent of receive support. The maximum number of PIP subfields an implementation can send or receive is implementation-defined; it is any number greater than or equal to 16.
2. The maximum send support for PIP subfields is implementation-defined.
3. The maximum receive support for PIP subfields is implementation-defined.
4. The zero-length map name has a special meaning: it indicates mapping is not to be performed by the LU.

Figure H-2. Symbol String Lengths

## FM HEADERS

This section explains how function management (FM) headers are used to exchange information by LU 6.2. It also defines the formats of the FM headers and how they are related to other data in request units.

The request header (RH) contains a format indicator (FI) that, when on, indicates that an FM header is at the beginning of the RU; FM headers may appear only singly at the beginning of an RU. The RU containing the FM header may appear anywhere within a chain.

The placement of FM headers within a request unit is shown below:

FM Header Contained in One RU:

| | | |
|---|---|---|
| RH: FMH, *BC,*EC | FM header | Data |

FM Header Contained in Two Contiguous RUs of a Chain:

| | |
|---|---|
| RH: FMH, *BC,¬EC | First of FM header |

| | | |
|---|---|---|
| RH: ¬FMH,¬BC,*EC | Rest of FM header | Data |

NOTE: FM headers are placed at the beginning of a request unit, but not necessarily the first or last request unit of a chain. When the FM header is longer than one RU will hold, the FM header is continued in as many additional RUs as are needed to hold it.

Figure H-3. Examples of FM Header Placement

The following FM headers are used by LU 6.2:

- FMH-5 carries a request for a conversation to be established between two transaction programs. The FMH-5 identifies the transaction program that is to be put into execution and connected to the receiving half-session.

  When a transaction program issues an ALLOCATE verb (see _SNA Transaction Programmer's Reference Manual for LU Type 6.2_ for details) naming a transaction program to be run at the other end of the conversation, an Attach FMH-5 carries the transaction program name (TPN) to the receiving LU.

- FMH-7 carries information that relates to an error on the session or conversation. For example, an FMH-7 and additional error information are sent when an FMH-5 specifies a nonexistent transaction program name.

The formats for these FM headers are shown below.

## Function Management Header 5: Attach

The function management header 5 (FMH-5), with a command of Attach, has the following format.

| | |
|---|---|
| 0 | Length, in binary, of FMH-5, including this Length byte |
| 1 | bit 0, reserved |
| | bits 1-7, FMH type: 0000101 |
| 2-3 | Command code: X'02FF' (Attach) |
| 4 | bits 0-3, reserved |
| | bit 4, program initialization parameter (PIP) presence: |
| | 0 PIP not present following this FMH-5 |
| | 1 PIP present following this FMH-5 (see "PIP Variable " on page H-7 for format) |
| | bits 5-7, reserved |
| 5 | Length (j-5), in binary, of Fixed Length Parameters field (currently 3--future expansion possible) |
| 6-j | <u>Fixed Length Parameters</u> |
| 6 | Resource type: |
| | X'D0' basic conversation |
| | X'D1' mapped conversation |
| 7 | Reserved |
| 8(=j) | bits 0-1, synchronization level: |
| | 00 none |
| | 01 confirm |
| | 10 confirm, sync point, and backout |
| | 11 reserved |
| | bits 2-7, reserved |
| j+1-p | <u>Variable Length Parameters</u> |
| j+1 | Length (values 1 to 64 are valid), in binary, of transaction program name |
| j+2-k | Transaction program name (see Figure H-1 on page H-1 for valid formats) |
| k+1 | Length (0 or m-k-1), in binary, of Access Security Information subfields |
| k+2-m | Zero or more Access Security Information subfields (see "Access Security Information Subfields " on page H-7 for format) |
| m+1 | Length (values 0 and 10 to 26 are valid), in binary, of Logical-Unit-of-Work Identifier field |
| m+2-n | <u>Logical-Unit-of-Work Identifier</u> |
| m+2 | Length (values 1 to 17 are valid), in binary, of fully qualified LU network name |
| m+3-w | Fully qualified LU network name (format described in "User Data Structured Subfield Formats" in "Appendix E. Request-Response Unit (RU) Formats") |
| w+1-w+6 | Logical-unit-of-work instance number, in binary |
| w+7-w+8(=n) | Logical-unit-of-work sequence number, in binary |
| n+1 | Reserved |

<u>Note:</u> Trailing Length fields (bytes n+1, m+1, and k+1) that have value X'00' can be omitted.

## Access Security Information Subfields

The Access Security Information subfields in FMH-5 have the following formats:

| | |
|---|---|
| 0 | Length (valid values are 1 to 11), in binary, of remainder of subfield--does not include this Length byte |
| 1 | Subfield type:<br>X'00' profile<br>X'01' password<br>X'02' user ID |
| 2-i | Data (see Figure H-1 on page H-1 for the symbol-string type and Figure H-2 on page H-3 for the symbol-string length restrictions) |

Note 1: The Access Security Information subfields may appear in any order in the Access Security Information field of the FMH-5.

Note 2: In FMH-5, the offset "m" represents the end of the last subfield.

## PIP Variable

The PIP variable following FMH-5 Attach has the following format:

| | |
|---|---|
| 0-1 | Length (4 or n+1), in binary, of PIP variable, including this Length field |
| 2-3 | GDS indicator: X'12F5' |
| 4-n | Zero or more PIP subfields, each of which has the following format (shown using zero-origin) |
| 0-1 | Length, in binary, of PIP subfield, including this Length field |
| 2-3 | GDS indicator: X'12E2' |
| 4-m | PIP subfield data (see Figure H-1 on page H-1 for the symbol-string type and Figure H-2 on page H-3 for the symbol-string length restrictions) |

=

## Function Management Header 7: Error Description

The function management header 7 (FMH-7) has the following format:

| | |
|---|---|
| 0 | Length (7), in binary, of FMH-7, including this Length byte |
| 1 | bit 0, reserved<br>bits 1-7, type: 0000111 |
| 2-5 | SNA-defined sense data (see below) |
| 6 | bit 0, error log variable presence:<br>0 no error log variable follows this FMH-7<br>1 error log GDS variable follows this FMH-7<br>bits 1-7, reserved |

The following sense data (in hexadecimal) can be sent in an FMH-7; see "Appendix G. Sense Data" for additional details on the sense data; The phrases following the sense data are the symbolic return codes provided to the application program in LU 6.2 verbs (see _SNA Transaction Programmer's Reference Manual for LU Type 6.2_) when the sense data is received.

| Sense Data | Return Code |
|---|---|
| 1008600B | RESOURCE_FAILURE_NO_RETRY |
| 10086021 | ALLOCATION_ERROR--TPN_NOT_RECOGNIZED |
| 10086031 | ALLOCATION_ERROR--PIP_NOT_ALLOWED |
| 10086032 | ALLOCATION_ERROR--PIP_NOT_SPECIFIED_CORRECTLY |
| 10086034 | ALLOCATION_ERROR--CONVERSATION_TYPE_MISMATCH |
| 10086041 | ALLOCATION_ERROR--SYNC_LEVEL_NOT_SUPPORTED_BY_PGM |
| 10086042 | ALLOCATION_ERROR--RECONNECT_LEVEL_NOT_SUPPORTED_BY_PGM |
| 10086043 | ALLOCATION_ERROR--TRANS_PGM_NOT_AVAIL_NO_RETRY |
| 10086044 | ALLOCATION_ERROR--TRANS_PGM_NOT_AVAIL_RETRY |
| 080F6051 | ALLOCATION_ERROR--ACC_NOT_VALID |
| 08240000 | BACKED_OUT |
| 084B6031 | ALLOCATION_ERROR--TRANS_PGM_NOT_AVAIL_RETRY |
| 084C0000 | ALLOCATION_ERROR--TRANS_PGM_NOT_AVAIL_NO_RETRY |
| 08640000 | DEALLOCATE_ABEND_PROG |
| 08640001 | DEALLOCATE_ABEND_SVC |
| 08640002 | DEALLOCATE_ABEND_TIMER |
| 08890000 | PROG_ERROR_NO_TRUNC or PROG_ERROR_PURGING |
| 08890001 | PROG_ERROR_TRUNC |
| 08890100 | SVC_ERROR_NO_TRUNC or SVC_ERROR_PURGING |
| 08890101 | SVC_ERROR_TRUNC |

## PRESENTATION SERVICES (PS) HEADERS

Presentation services (PS) headers convey information between PS component sync point managers when the conversation using the session is allocated with the sync-point synchronization level. "Chapter 5.3. Presentation Services--Sync Point Services Verbs" describes the use of these PS headers.

Transaction program data is delimited using a 2-byte length field called an LL, containing a value that is the number of bytes contained in the transaction program data plus 2 (the length of the LL field itself).

| | |
|---|---|
| LL | transaction program data |

All PS headers are identified by an LL of X'0001' immediately preceding the header. X'0001' is an illegal LL value for use by transaction programs because the LL's value must include the length of itself, which is 2 bytes. Therefore, all LLs indicating a length of less than 2 are reserved for use by the LU. The format of PS headers is shown below.

### Presentation Services Header 10: Sync Point Control

Presentation Services Header 10 (Sync Point Control) has the following format:

| | |
|---|---|
| 0 | Length, in binary, of PS header, including this length field |
| 1 | bit 0, reserved<br>bits 1-7, type: 0001010 sync point control (only value defined) |
| 2-3 | Sync point command type:<br>X'0005' Prepare<br>X'0006' Request Commit<br>X'0007' Committed<br>X'0008' Forget<br>X'0009' Heuristic Mixed |
| 4-5 | Modifier specifying next flow (present only if bytes 2-3 = X'0005' or X'0006'; reserved when bytes 2-3 = X'0006' and 2-phase sync point being used):<br>X'0000' request PREPARE TO RECEIVE<br>X'0001' request DEALLOCATE<br>X'0002' request SEND<br>Note: Bytes 4-5 affect the CD and CEB settings generated by data flow control on the last PS header in the sync point sequence, i.e., Forget if Prepare was received, and Committed if Request Commit was the first PS header received (see "Chapter 5.3. Presentation Services--Sync Point Services Verbs" for details). |

=

## FORMATS OF RECORDS USED BY LU 6.2 SERVICE TRANSACTION PROGRAMS

A TPN value starting with X'06' in the Attach header indicates that the LU resources manager is to initiate execution of one of the LU service transaction programs.

LU services managers exchange data directly via GDS variables. A group of GDS IDs of the form X'12**' is assigned for use by the LU service transaction programs; the commands use a Service Flag (SF) byte (following the GDS ID) to denote the processing options of the command; the specific options are encoded in the last four bits of the Service Flag byte as shown in the individual GDS variables. Requests have bit 4 set to 0 and replies have bit 4 set to 1; therefore, requests and replies have the same ID values. The first four bits are unique to the command.

## Change Number of Sessions (CNOS)

See "Chapter 5.4. Presentation Services--Control-Operator Verbs" for a detailed description of the use of this command.

| | |
|---|---|
| 0-1 | Length (17 or n+1), in binary, of Change Number of Sessions GDS variable, including this Length field |
| 2-3 | GDS ID: X'1210' |
| 4 | Service flag: |
| | bits 0-3, reserved |
| | bits 4-7, request/reply indicator: |
| | 0010 request |
| | 1000 reply, function completed abnormal |
| | 1010 reply, function accepted but not yet completed |
| 5 | Reply modifier (reserved if byte 4, bits 4-7 = 0010): |
| | X'00' normal--no negotiation performed |
| | X'01' abnormal--command race detected |
| | X'02' abnormal--mode name not recognized |
| | X'03' reserved |
| | X'04' normal--negotiated reply |
| | X'05' abnormal--(LU,mode) session limit is 0 |
| 6 | Action: |
| | X'00' set (LU,mode) session limits |
| | X'01' reserved |
| | X'02' close |
| 7 | Drain immediacy: |
| | bits 0-2, reserved |
| | bit 3, source LU drain (reserved if byte 6 ¬= 02): |
| | 0 no (send BIS at next opportunity) |
| | 1 yes |
| | bits 4-6, reserved |
| | bit 7, target LU drain (reserved if byte 6 ¬= 02): |
| | 0 no (send BIS at next opportunity) |
| | 1 yes |
| 8 | Action flags: |
| | bits 0-6, reserved |
| | bit 7, session deactivation responsibility: |
| | 0 sender of Change Number of Sessions request (source LU) |
| | 1 receiver of Change Number of Sessions request (target LU) |
| | <u>Note:</u> Bytes 9-14 are reserved if byte 6 ¬= 0. |
| 9-10 | (LU,mode) session limit: |
| | bit 0, reserved |
| | bits 1-15, maximum (LU,mode) session count, in binary |
| 11-12 | Source LU contention winners: |
| | bit 0, reserved |
| | bits 1-15, guaranteed minimum number of contention winner sessions at source LU, in binary |
| 13-14 | Target LU contention winners: |
| | bit 0, reserved |
| | bits 1-15, guaranteed minimum number of contention winner sessions at target LU, in binary |
| 15 | Mode name selection: |
| | bits 0-6, reserved |
| | bit 7, mode names affected by this command: |
| | 0 a single mode name is affected |
| | 1 all mode names are affected |
| 16 | Length (values 0 to 8 are valid; reserved if byte 15, bit 7 = 1), in binary, of mode name |
| 17-n | Mode name (omitted if byte 16 = X'00') |

=

Exchange Log Name

See "Chapter 5.3. Presentation Services--Sync Point Services Verbs" for a detailed description of the use of this command.

| | |
|---|---|
| 0-1 | Length (p+1), in binary, of Exchange Log Name GDS variable, including this Length field |
| 2-3 | GDS ID: X'1211' |
| 4 | Service flag:<br>bits 0-3, reserved<br>bits 4-7, request/reply indicator:<br>0010 request<br>1000 reply, function completed abnormally<br>1001 reply, function completed normally |
| 5 | Sync point manager flags:<br>bit 0, program reconnect support:<br>0 no<br>1 yes<br>bits 1-6, reserved<br>bit 7, log status:<br>0 cold<br>1 warm |
| 6 | Length (values 1 to 17 are valid), in binary, of fully qualified LU network name |
| 7-n | Fully qualified LU network name (format described in "User Data Structured Subfield Formats" in "Appendix E. Request-Response Unit (RU) Formats") |
| n+1 | Length (values 1 to 64 are valid), in binary, of log name |
| n+2-p | Log name (symbol-string type-AE) |

Compare States

See "Chapter 5.3. Presentation Services--Sync Point Services Verbs" for a detailed description of the use of this command.

| | |
|---|---|
| 0-1 | Length (p+1), in binary, of Compare States GDS variable, including this Length field |
| 2-3 | GDS ID: X'1213' |
| 4 | Service flag:<br>bits 0-3, reserved<br>bits 4-7, request/reply indicator:<br>0010 request<br>1000 reply, function completed abnormally<br>1001 reply, function completed normally |
| 5 | Sync point manager state:<br>X'01' RESET<br>X'02' SYNC_POINT_MANAGER_PENDING<br>X'03' IN_DOUBT<br>X'04' COMMITTED<br>X'05' HEURISTIC_RESET<br>X'06' HEURISTIC_COMMITTED<br>X'07' HEURISTIC_MIXED |
| 6 | bit 0, transaction program status:<br>0 transaction program not restartable<br>1 transaction program restartable<br>bits 1-7, reserved |
| 7 | Length, in binary, of Logical-Unit-of-Work Identifier field (values 10 to 26 are valid) |
| 8-n | <u>Logical-Unit-of-Work Identifier</u> |
| 8 | Length, in binary, of fully qualified LU network name (values 1 to 17 are valid) |
| 8-w | Fully qualified LU network name (format described in "User Data Structured Subfield Formats" in "Appendix E. Request-Response Unit (RU) Formats") |
| w+1-w+6 | Logical-unit-of-work instance number, in binary |
| w+7-<br>w+8(=n) | Logical-unit-of-work sequence number, in binary |
| n+1 | Length (values 0 to 8 are valid), in binary, of conversation correlator |
| n+2-q | Conversation correlator of transaction program to restart: see FMH-5 for format of this correlator |
| q+1 | Length (values 2 to 8 are valid), of session instance identifier |
| q+2-p | Session instance identifier of session being used by conversation at time of failure (See "User Data Structured Subfield Formats" in "Appendix E. Request-Response Unit (RU) Formats" for the format of this identifier.) |

=

## SNA-DEFINED TRANSACTION PROGRAM NAMES

The following transaction program names (TPNs) specify SNA-defined service transaction programs discussed in this book.

| TPN | Service Transaction Program |
|---|---|
| X'06F1' | Change Number of Sessions |
| X'06F2' | Sync Point Resynchronization |

The following chart indicates (using an "X") each GDS variable code point (with first byte = X'12') used by LU 6.2.

First hexadecimal digit
Second hexadecimal digit

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1 | X | X |   | X |   |   |   |   |   |   |   |   |   |   |   |   |
| 2 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 3 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 4 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 5 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 6 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 7 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 8 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 9 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| A | X |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| B |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| C |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| D |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   | X |
| E |   | X | X |   |   |   |   |   |   |   |   |   |   |   |   |   |
| F |   | X | X | X | X | X |   |   |   |   |   |   |   |   |   | X |

The code points used by LU 6.2 are:

| | |
|---|---|
| X'1210' | Change Number of Sessions |
| X'1211' | Exchange Log Name |
| X'1213' | Compare States |
| X'12A0' | Workstation Display Passthrough |
| X'12E1' | Error Log |
| X'12E2' | PIP Subfield Data |
| X'12F1' | Null Structured Data |
| X'12F2' | User Control Data |
| X'12F3' | Map Name |
| X'12F4' | Error Data |
| X'12F5' | PIP Data |
| X'12FF' | Application Data |

## Format of Application Data GDS Variable

The Application Data GDS variable, ID X'12FF', contains application data. The application transaction program's data as specified in the MC_SEND_DATA verb is (optionally) mapped and then sent as X'12FF' variables.

## Format of Null Structured Data Variable

The Null Structured Data GDS variable, ID X'12F1', contains no application data. This variable may optionally be generated (see "Chapter 5.2. Presentation Services--Mapped Conversation Verbs") to carry certain control information (e.g., Confirm) when no application data is available.

## Format of User Control Data GDS Variable

The User Control Data GDS variable, ID X'12F2', contains user control data. The meaning of this data is known only to the LU Services Component Programs or the transaction programs and their mapping programs. This data can be used, for example, as prefix control information for an Application Data GDS variable that follows it or to carry FM Header Data for a mapped conversation transaction.

## Format of Map Name GDS Variable

The Map Name GDS variable, ID X'12F3', is followed by a 0- to 64-byte map name. See Figure H-1 on page H-1 for the valid map name symbol-string type.

## Format of an Error Data GDS variable

The Error Data GDS variable, ID X'12F4', is used to convey information about mapping errors. It is sent using the SEND_DATA verb following a SEND_ERROR verb. Its format is:

| | |
|---|---|
| 0-1 | Length (n+1), in binary, of Error Data GDS variable, including this Length field |
| 2-3 | GDS ID: X'12F4' |
| 4-7 | Error code:<br>X'00010000' Invalid GDS ID: The mapped conversation verb component (see "Chapter 5.2. Presentation Services--Mapped Conversation Verbs") encountered a GDS ID that it did not recognize.<br>X'00030001' Map Not Found: The specified map was not available at the target, or access to the referenced map could not be completed.<br>X'00030002' Map Execution Failure: The map program was not able to process the data stream. |
| 8 | Length (n-8), in binary, of error parameter |
| 9-n | Error parameter: for a mapping failure, the map name carried in the GDS variable for which the error occurred; for an invalid GDS ID, the 2-byte GDS ID that was not recognized |

## Format of Error Log GDS Variable

The Error Log GDS variable, ID X'12E1', following an FMH-7 conveys implementation-specific error information to an LU, where it is added to the system error log for use in debugging and error recovery. It is not used by SNA-defined service transaction programs (other than to log it) since it contains implementation-specific data. The Error Log variable is sent as a consequence of issuing the SEND_ERROR verb, but is not passed to the receiving transaction program. Its format is:

| | |
|---|---|
| 0-1 | Length (n+1), in binary, of Error Log GDS variable, including this Length field |
| 2-3 | GDS ID: X'12E1' |
| 4-m | <u>Product ID</u> |
| 4-5 | Length, in binary, of product ID, including this Length field (values 2 to 32,767 are valid)<br><u>Note:</u> The Length field is always present; a value of 2 indicates no Product ID vector follows. |
| 6-m | Product ID vector (format described in "Product Set ID" CNM subvector in "Appendix E. Request-Response Unit (RU) Formats") |
| m+1-n | <u>Message Text</u> |
| m+1-m+2 | Length, in binary, of message text, including this Length field (values 2 to 32,767 are valid)<br><u>Note:</u> The Length field is always present; a value of 2 indicates no message text follows. |
| m+3-n | Message text data: implementation-specific data |

# APPENDIX I. GENERAL DATA STREAM

This appendix defines the general data stream (GDS), which is used in a variety of ways by SNA products. For instance, it is used to encode the Document Interchange Architecture (DIA) message units. The basic structural unit in GDS is the structured field, a string of bytes beginning with a length and followed by a GDS identifier (ID) that defines the structure of the remainder of the field. Some structured fields are used by components of SNA that are defined in this book; these uses are defined in "Appendix E. Request-Response Unit (RU) Formats" and "Appendix H. FM Header and LU Services Commands".

GDS IDs are assigned, generally in blocks of consecutive values, to different layers and components of SNA and to other interconnection architectures. For a complete listing of these block assignments, see the SNA Reference Summary.

The general data stream applies to data exchanged between nodes over SNA links, over non-SNA links, and to data exchanged via removable storage media or shared storage facilities.

## STRUCTURED FIELDS

Each structured field has the format shown in Figure I-1.

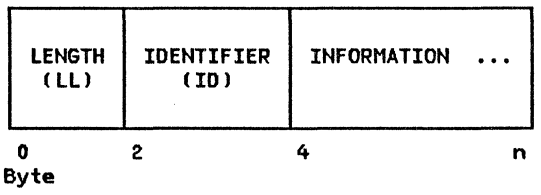


Figure I-1. GDS Structured Field

### LENGTH (LL) DESCRIPTION

Bits 1 to 15 of the LL contain a binary number from 4 through 32767, which is the length, in bytes, of the entire GDS structure field, including the LL bytes. Length values of 0 through 3 are reserved for use as escape sequences. For example, a value of X'0001' indicates a presentation services header follows, which is used for sync point management.

Bit 0 of byte 0 (high-order bit) is used for a continuation indicator, where a value of 0 means last GDS variable segment, and a value of 1 means not-last segment. Some data streams built from structured fields use other methods to create data objects that are longer than a 15-bit length can specify.

### IDENTIFIER (ID) DESCRIPTION

The 2-byte identifier that follows the length field describes the format and meaning of the data that follow. Sometimes additional values appearing in the information field are needed to completely specify the information field's content.

# 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."

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 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.

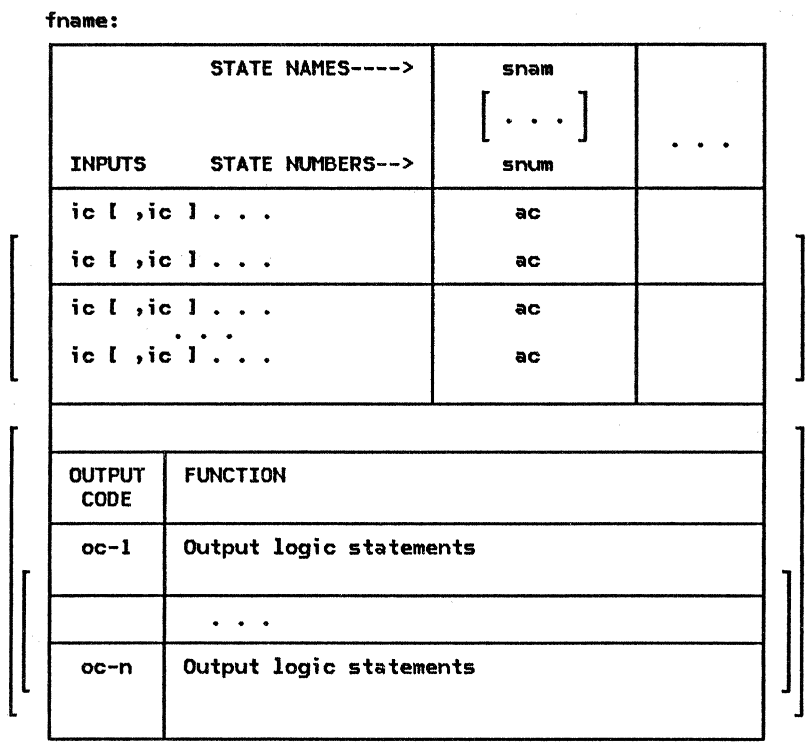
The input lines within the matrix are scanned from top to bottom at execution time. The first input line found with all its conditions true is used to address the matrix for the next state and the output code. No more than one input line in a matrix has all its conditions true during a scan.

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.



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 |
| ACTLU | ACTIVATE LOGICAL UNIT |
| API | application programming interface |
| ASCII | American Standard Code for Information Interchange |
| BB | Begin Bracket |
| BBI | Begin Bracket indicator |
| BC | Begin Chain |
| BCI | Begin Chain indicator |
| BETB | between brackets |
| BIND | BIND SESSION |
| BINDF | BIND FAILURE |
| BIS | BRACKET INITIATION STOPPED |
| BIU | basic information unit |
| CD | Change Direction |
| CDI | Change Direction indicator |
| CEB | Conditional End Bracket |
| CINIT | CONTROL INITIATE |
| CLEANUP | CLEAN UP SESSION |
| CNOS | change number of sessions |
| COPR | control operator services |
| COS | class of service |
| CP | control point |
| CR | conversation resource |
| CRV | CRYPTOGRAPHY VERIFICATION |
| CSI | Code Selection indicator |
| CT | correlation table |
| CTERM | CONTROL TERMINATE |
| DACTLU | DEACTIVATE LOGICAL UNIT |
| DES | Data Encryption Standard |
| DFC | data flow control |
| DIA | Document Interchange Architecture |
| DLC | data link control |
| DLU | destination LU |
| DR1 | Definite Response 1 |
| DR1I | Definite Response 1 indicator |
| DR2 | Definite Response 2 |
| DR2I | Definite Response 2 indicator |
| DSU | distribution service unit |
| EC | End Chain |
| ECHOTEST | ECHO TEST |
| ECI | End Chain indicator |
| ED | Enciphered Data |

| | |
|---|---|
| EDI | Enciphered Data indicator |
| EFI | Expedited Flow indicator |
| ERI | Exception Response indicator |
| ERP | error recovery procedure(s) |
| 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 |
| 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 |
| ILU | initiating LU (LU sending INIT-SELF) |
| INIT | initiate |
| INIT-SELF | INITIATE-SELF |
| IPR | ISOLATED PACING RESPONSE |
| LIC | last in chain (¬BC, EC) |
| LL | logical record length (prefix) |
| LLID | logical record length and GDS ID (prefix) |
| LNS | LU network services |
| 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 |
| MIC | middle in chain (¬BC,¬EC) |
| MSG | message |
| MU | message unit |
| NAU | network addressable unit |
| NC | network control |
| NEG | negative |
| NG | no good |
| NNM | nodal NAU manager |
| NS | network services |

| | |
|---|---|
| NTWK | network |
| OIC | only in chain (BC, EC) |
| OLU | origin LU |
| P | primary |
| PAC | Pacing Request, Pacing Response (value of PI in RH) |
| PC | path control |
| 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 |
| 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 |
| REQECHO | REQUEST ECHO TEST |
| RES | resource(s) |
| RESYNC | sync point resynchronization service TP |
| RH | request/response header |
| 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 |
| S | secondary, sending |
| SCB | session control block |
| SCS | SNA character string |
| SD | Sense Data Included |
| SDI | Sense Data Included indicator |
| SEC | secondary |
| SESS | session |
| SESSEND | SESSION ENDED |
| SESSST | SESSION STARTED |

| | |
|---|---|
| SETCV | SET CONTROL VECTOR |
| SIG | SIGNAL |
| SLDLM | session-limit data-lock manager |
| SLU | secondary LU |
| SNA | Systems Network Architecture |
| SNADS | 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 |
| SYNCPT | synchronization point |
| TC | transmission control |
| TCB | transaction control block |
| TCCB | transmission control control block |
| TERM | terminate, terminating, termination, terminal |
| TERM-SELF | TERMINATE-SELF |
| TH | transmission header |
| TLU | terminating logical unit (LU sending TERM) |
| TP | transaction program |
| TS | transmission services |
| TSLS | target-LU session-limit services |
| TSP | TS profile |
| UNBIND | UNBIND SESSION |
| UNBINDF | UNBIND FAILURE |
| UPM | undefined protocol machine |
| URC | user request correlation |

**Systems Network Architecture**
**Format and Protocol**
**Reference Manual:**
**Architecture Logic for LU Type 6.2**

**READER'S COMMENT FORM**

**Order No. SC30-3269-2**

SC30-3269-2

## 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
Dept. E01
P.O. Box 12195
Research Triangle Park, N.C. 27709-2195

**Fold and tape** **Please Do Not Staple** **Fold and tape**

IBM®

**Systems Network Architecture**
**Format and Protocol**
**Reference Manual:**
**Architecture Logic for LU Type 6.2**

**READER'S COMMENT FORM**

**Order No. SC30-3269-2**

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 Legibility

If you wish a reply, give your name, company, mailing address, and date:

______________________________

______________________________

______________________________

______________________________

What is your occupation?______________________________

Number of latest Newsletter associated with this publication:____________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or 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.)

SC30-3269-2

**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
Dept. E01
P.O. Box 12195
Research Triangle Park, N.C. 27709-2195

**Fold and tape** **Please Do Not Staple** **Fold and tape**

IBM®

# APPENDIX T. TERMINOLOGY: ACRONYMS AND ABBREVIATIONS

| | |
|---|---|
| ACT | activate |
| ACTLU | ACTIVATE LOGICAL UNIT |
| API | application programming interface |
| ASCII | American Standard Code for Information Interchange |
| BB | Begin Bracket |
| BBI | Begin Bracket indicator |
| BC | Begin Chain |
| BCI | Begin Chain indicator |
| BETB | between brackets |
| BIND | BIND SESSION |
| BINDF | BIND FAILURE |
| BIS | BRACKET INITIATION STOPPED |
| BIU | basic information unit |
| CD | Change Direction |
| CDI | Change Direction indicator |
| CEB | Conditional End Bracket |
| CINIT | CONTROL INITIATE |
| CLEANUP | CLEAN UP SESSION |
| CNOS | change number of sessions |
| COPR | control operator services |
| COS | class of service |
| CP | control point |
| CR | conversation resource |
| CRV | CRYPTOGRAPHY VERIFICATION |
| CSI | Code Selection indicator |
| CT | correlation table |
| CTERM | CONTROL TERMINATE |
| DACTLU | DEACTIVATE LOGICAL UNIT |
| DES | Data Encryption Standard |
| DFC | data flow control |
| DIA | Document Interchange Architecture |
| DLC | data link control |
| DLU | destination LU |
| DR1 | Definite Response 1 |
| DR1I | Definite Response 1 indicator |
| DR2 | Definite Response 2 |
| DR2I | Definite Response 2 indicator |
| DSU | distribution service unit |
| EC | End Chain |
| ECHOTEST | ECHO TEST |
| ECI | End Chain indicator |
| ED | Enciphered Data |

| | |
|---|---|
| EDI | Enciphered Data indicator |
| EFI | Expedited Flow indicator |
| ERI | Exception Response indicator |
| ERP | error recovery procedure(s) |
| 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 |
| 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 |
| ILU | initiating LU (LU sending INIT-SELF) |
| INIT | initiate |
| INIT-SELF | INITIATE-SELF |
| IPR | ISOLATED PACING RESPONSE |
| LIC | last in chain (¬BC, EC) |
| LL | logical record length (prefix) |
| LLID | logical record length and GDS ID (prefix) |
| LNS | LU network services |
| 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 |
| MIC | middle in chain (¬BC,¬EC) |
| MSG | message |
| MU | message unit |
| NAU | network addressable unit |
| NC | network control |
| NEG | negative |
| NG | no good |
| NNM | nodal NAU manager |
| NS | network services |

| | |
|---|---|
| NTWK | network |
| OIC | only in chain (BC, EC) |
| OLU | origin LU |
| P | primary |
| PAC | Pacing Request, Pacing Response (value of PI in RH) |
| PC | path control |
| 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 |
| 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 |
| REQECHO | REQUEST ECHO TEST |
| RES | resource(s) |
| RESYNC | sync point resynchronization service TP |
| RH | request/response header |
| 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 |
| S | secondary, sending |
| SCB | session control block |
| SCS | SNA character string |
| SD | Sense Data Included |
| SDI | Sense Data Included indicator |
| SEC | secondary |
| SESS | session |
| SESSEND | SESSION ENDED |
| SESSST | SESSION STARTED |

| | |
|---|---|
| SETCV | SET CONTROL VECTOR |
| SIG | SIGNAL |
| SLDLM | session-limit data-lock manager |
| SLU | secondary LU |
| SNA | Systems Network Architecture |
| SNADS | 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 |
| SYNCPT | synchronization point |
| TC | transmission control |
| TCB | transaction control block |
| TCCB | transmission control control block |
| TERM | terminate, terminating, termination, terminal |
| TERM-SELF | TERMINATE-SELF |
| TH | transmission header |
| TLU | terminating logical unit (LU sending TERM) |
| TP | transaction program |
| TS | transmission services |
| TSLS | target-LU session-limit services |
| TSP | TS profile |
| UNBIND | UNBIND SESSION |
| UNBINDF | UNBIND FAILURE |
| UPM | undefined protocol machine |
| URC | user request correlation |

SC30-3269-2
Printed in U.S.A.

SC30-3269-02