SC30-3112-2
File No. S370-30

Systems

Systems Network Architecture Format and Protocol Reference Manual: Architectural Logic

Third Edition (November 1980)

This is a major revision of, and obsoletes, SC30-3112-1. Before using this manual in connection with the operation of IBM systems or equipment, refer to the latest IBM System/370 Bibliography, GC20-0001, and associated Technical Newsletters, for the editions that are applicable and current.

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

This manual has been written by the IBM System Communications Division, Communication Systems Architecture, Department E99, P.O. Box 12195, Research Triangle Park, North Carolina 27709, and published by the IBM System Communications Division, Publications Development, Department E02, P.O. Box 12195, Research Triangle, Park North Carolina 27709. A reader's comment form is provided at the back of this manual. If the form has been removed, comments may be sent to the publisher. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Preface

This book is intended for system programmers and others who need detailed information about Systems Network Architecture (SNA) in order to develop or adapt a product or program to function within an SNA network. The book provides a comprehensive reference to the formats and protocols of SNA from a design viewpoint.

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

- SNA Concepts and Products, GC30-3072 (when available)--tutorial information.
- SNA Technical Overview, GC30-3073 (when available)--tutorial information.
- IBM SDLC General Information, GA27-3093--supplementary details of Synchronous Data Link Control.
- SNA Reference Summary, GA27-3136--summary information on SNA formats and sequences.
- SNA--Sessions Between Logical Units,[1] GC20-1868 (when available)--supplementary details of services provided for communication between end users (terminal operators and application programs) of an SNA network.

This book does not describe any specific equipment or programs that may implement SNA, nor does it describe any implementation 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 equipment or programs to be used.

SNA is an open-ended architecture and may be altered from time to time by IBM. Extensions and modifications to SNA will be described in future editions of this book.

This edition differs considerably from the previous edition and should be reviewed in its entirety for changes.

---

[1] Referred to by the title, SNA LU-LU Session Types, elsewhere in this book; it was renamed after this book had gone to press.

This page intentionally left blank

CHAPTER 1. INTRODUCTION

CHAPTER 2. MESSAGE UNITS AND HEADER FORMATS

CHAPTER 3. PATH CONTROL

CHAPTER 4. TRANSMISSION CONTROL

CHAPTER 5. DATA FLOW CONTROL

CHAPTER 6. OVERVIEW OF NETWORK SERVICES

CHAPTER 7. SSCP.SVC_MGR--CONFIGURATION SERVICES

CHAPTER 8. SESSION SERVICES

CHAPTER 13. PU.SVC MGR.CSC_MGR

APPENDIX A. NODE DATA STRUCTURES AND CONSTANTS

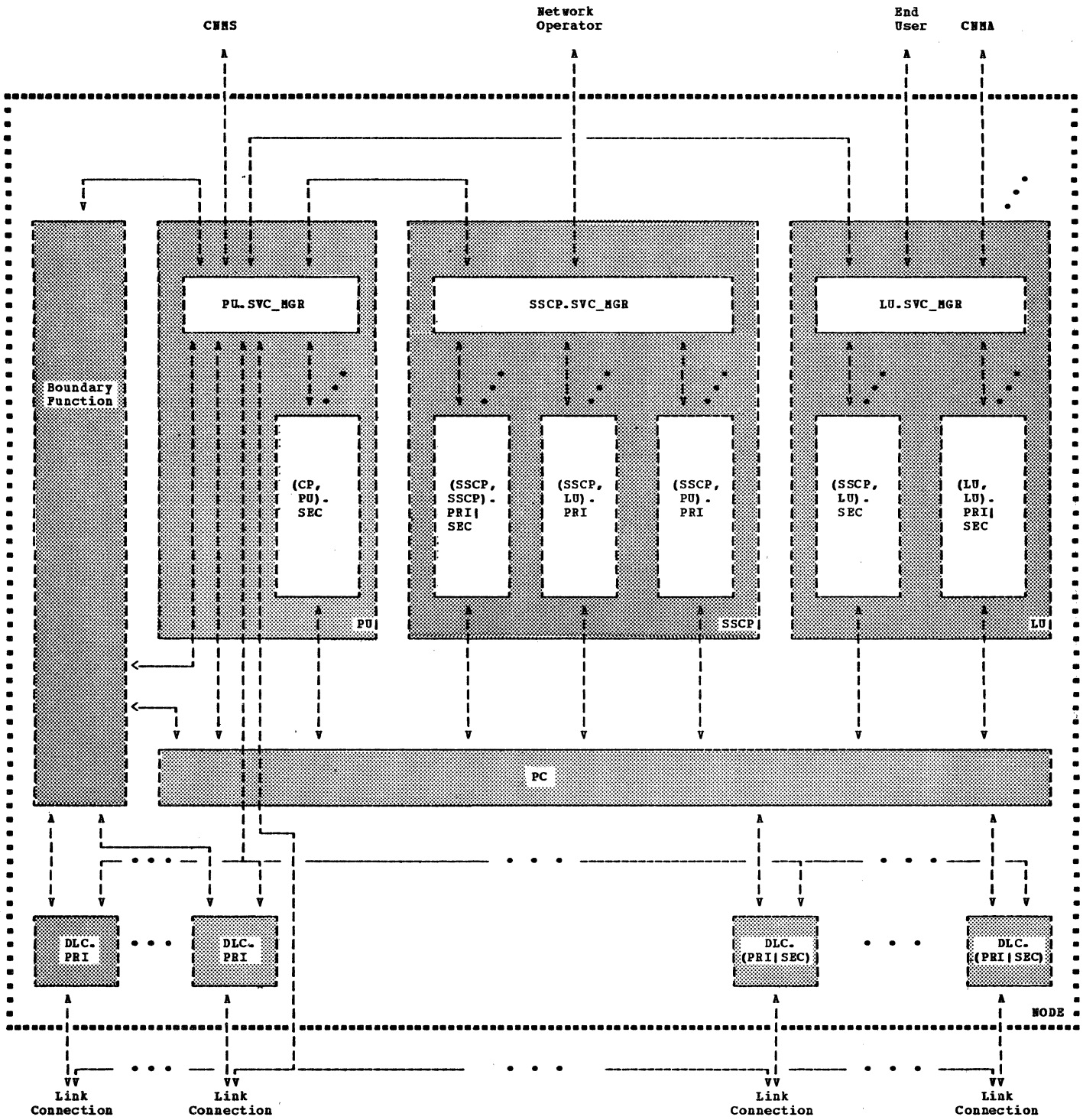
APPENDIX C. THE EXECUTION MODEL

APPENDIX D. TH AND RH FORMATS

APPENDIX E. REQUEST-RESPONSE UNIT (RU) FORMATS

APPENDIX G. SENSE DATA

APPENDIX N. NOTATION AND DEFINITIONS



Note: Only a type 5 node contains an SSCP; a type 1, 2, or 4 node contains a PUCP (not shown), which is a subset of an SSCP.

Frontispiece - Structural Overview of a Node

This page intentionally left blank

USE AND ORGANIZATION OF THIS BOOK

This book, in conjunction with the companion books, SNA LU-LU Session Types and SDLC General Information, 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; such information should be sought in the product publications.

The definition of SNA requires:

- Defining formats of information transferred between distinct SNA nodes over links connecting them.
- Making explicit any coupling between distinct information transfers; that is, defining the protocols, or rules, associated with the transfers.

Although it is possible to represent protocols by sets of valid sequences of data and control-information transfers, for SNA this would require too lengthy an enumeration to be practical, or even useful to a designer. Sequences are used only tutorially within this book.

SNA is defined here in the form of a functionally layered system, represented in the form of a meta-implementation,[1] that is decomposable into components called protocol machines. Protocol machines generate the valid output sequences in response to input sequences, subject to the associated protocols for distinct information transfers into, out of, and within the system.

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

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

---

[1] A meta-implementation resembles an actual implementation, in that it is defined in terms of a formal, human- or machine-executable notation, or programming language, using explicit data structures, and having an underlying abstract machine environment. By its modular, sequence-generating description of protocol machines, the meta-implementation provides a concrete model for actual implementations.

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

- Block diagrams: A block diagram represents the decomposition of a protocol machine into its component submachines (which themselves are protocol machines) and the signaling paths between them. Each block in the diagram can be further decomposed into its constituent submachines. At the most detailed level, a block can be shown as interconnected sets of routing and checking logic, finite-state machines, queues, and other primitive protocol machines.

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

Routing and checking logic is represented in the form of PL/I-like procedures, using a descriptive language called Format and Protocol Language (FAPL). FSMs are generally defined by FAPL state-transition matrices, procedures, and control blocks. In Chapters 8-9, FSMs are represented also in the form of state-transition graphs, as in previous editions. (In this book, "FSM" is frequently applied only to the state-transition matrix or graph representation.) Appendix N defines, in detail, the syntax and semantics of FAPL and the descriptive techniques and notational conventions for representing combinational logic and FSMs.

A naming convention, using qualifiers separated by periods to denote more specific components of a composite FSM, 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 also applies to phrases identifying state names and FAPL variables.

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

The rest of this chapter presents an orderly development of the structural and functional properties of SNA networks. It begins with the fundamental concepts of network addressable units, the path control network, links, nodes, domains, and message units, and successively refines, within this context, the concepts of sessions, flows, network layers, pairings, and services. These concepts serve as a prologue for the detailed format and protocol descriptions given later, in which the material is presented on the basis of session, layer, network services category, and layer manager.

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

- Chapter 2 describes the contents and formats of the major message units and headers used throughout the book.

- Chapter 3 describes routing and flow control within path control.

- Chapters 4 and 5 describe the transmission control and data flow control protocols, respectively, within half-sessions.

- Chapters 6-9 describe the network services protocols from the viewpoints of the SSCPs and LUs. Chapter 6 contains an overview for Chapters 7-9 and session network services logic common to Chapters 7-13. Chapters 7-9 deal with the specific network services categories.

- Chapters 10-13 describe the PU services manager. Chapter 10 provides an overview for Chapters 11-13. Chapter 11 describes the PU services manager component concerned with network services (SSCP-PU) protocols and with management of link-level and other resources local to the PU. Chapter 12 describes PU-PU protocols concerned with managing path control connections (virtual and explicit routes) and describes PU-SSCP network services protocols, from the viewpoint of the PU, for reporting test status and inoperative conditions of these connections. Chapter 13 describes the component that manages session-activation, -deactivation, and -outage notification.

- Appendixes A-C describe the data structures, utility procedures, and execution model or environment used for the meta-implementation. Appendix A is particularly useful to a reader wanting a detailed knowledge of the relationships among the control blocks used by the meta-implementation.

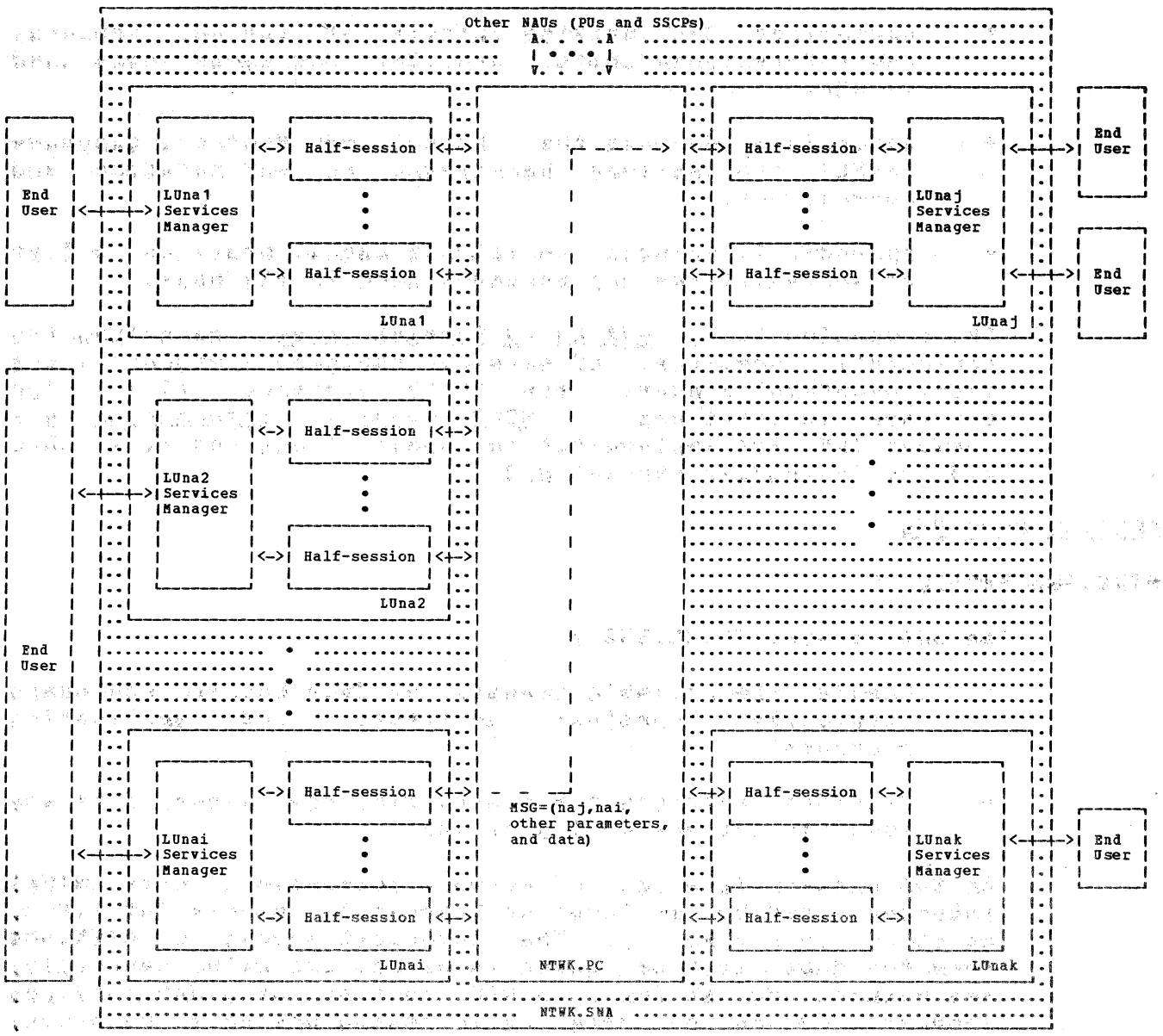


Figure 1-1. Overview of NTWK.SNA

- Appendixes D-G provide details of various headers, request-response units, profiles, and sense data used in SNA.

- Appendix N defines the Format and Protocol Language (FAPL) and provides background on FSM notation and conventions.

- Appendix T, printed on foldout pages, provides a list of abbreviations and acronyms used in the book.

The companion book, SNA LU-LU Session Types, describes the presentation services, LU services manager components, and session-option subsets for LU-LU sessions (i.e., for end-user interactions). SDLC General Information and various IBM SNA implementation publications describe SDLC and the System/370 channel DLC.

GENERAL CONCEPTS

NTWK.SNA PROTOCOLS

An SNA network (NTWK.SNA):

- 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 is a set of network addressable units (NAUs) interconnected by an inner path control network (NTWK.PC), as shown in Figure 1-1. The outermost layers of NTWK.SNA form the NAUs, each of which is associated with, generally, one network address (na). A NAU consists of a NAU services manager and one or more half-session protocol machines, depending on the number of other NAUs with which it can be paired to form sessions. Details of NAU structure, function, and sessions are given in later sections.

Those NAUs having protocol boundaries with end users are called logical units (LUs). An LU allows an attached end user to gain access to network resources and to communicate with other end users. An LU may also provide a service 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-LU session types, product design, and customer options. Services unique to LU-LU sessions are described in detail in SNA LU-LU Session Types and generally are not described further in this book.

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. For example, whether an application program end user can concurrently access the network through multiple LUs or is constrained to use a single LU is not specified in this book.

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; 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 network services related to session, configuration, maintenance and management, and network-operator services.

Message units are transported between NAUs by NTWK.PC, which consists of all the path control (PC) and data link control (DLC) components in the SNA network. (PC and DLC are described individually in later sections.) These message units are of the form:

MSG = (naj,nai,other parameters, and data),

where naj is an address of the destination NAU, and nai that of the origin NAU. NTWK.PC routes and delivers message units to naj in the same order as sent from nai.

The message units transferred within NTWK.SNA generally have two components: end-user information and control information. The end-user information is passed by the SNA network and does not affect the state of NTWK.SNA. 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 NTWK.SNA, thus effecting a normal control change (such as a change to a path control routing table) or a recovery from an exception condition.

NTWK.SNA--NODES AND THEIR PHYSICAL AND LOGICAL INTERCONNECTIONS

NTWK.SNA consists of node protocol machines physically interconnected via link-connection protocols (see Figure 1-2). 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, customer-defined protocol machines that use one or more SNA product nodes. These relationships are shown in Figure 1-3.

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.

Link-connection protocols--such as EIA RS-232-C, CCITT X.21, and System/370 channel input/output interface--are also not described in this book. The protocol boundaries between link-connection protocol machines and node protocol machines are described here to some extent, as well as in SDLC General Information and IBM SNA implementation publications.

Four node types are defined in SNA: types 1, 2, 4, and 5. They are distinguished by varying capabilities, such as for interconnection, and by the presence or absence of different NAU types. Types 1 and 2 nodes are also referred to as peripheral nodes, because of their limited addressing and routing capabilities. They are solely sources and sinks of data, and 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, discussed in a later section. It contains a subarea node and all the peripheral nodes attached to the subarea node.) Subarea nodes, besides 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 flow control of traffic within the subarea routing portion of the network.

An SNA product node containing multiple SNA nodes can provide product-defined protocols for routing between them. This has been done in some SNA products by connecting an SNA type 2 peripheral node to one SNA network and providing product-defined end-user protocols to connect the peripheral

node in the product node to a type 5 subarea node in the same product node, where the subarea node is part of a separate SNA network. Product-defined "pass-through" protocols connecting the two SNA nodes allow the peripheral node to act, in effect, as an intermediate routing node between the two SNA networks.

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.

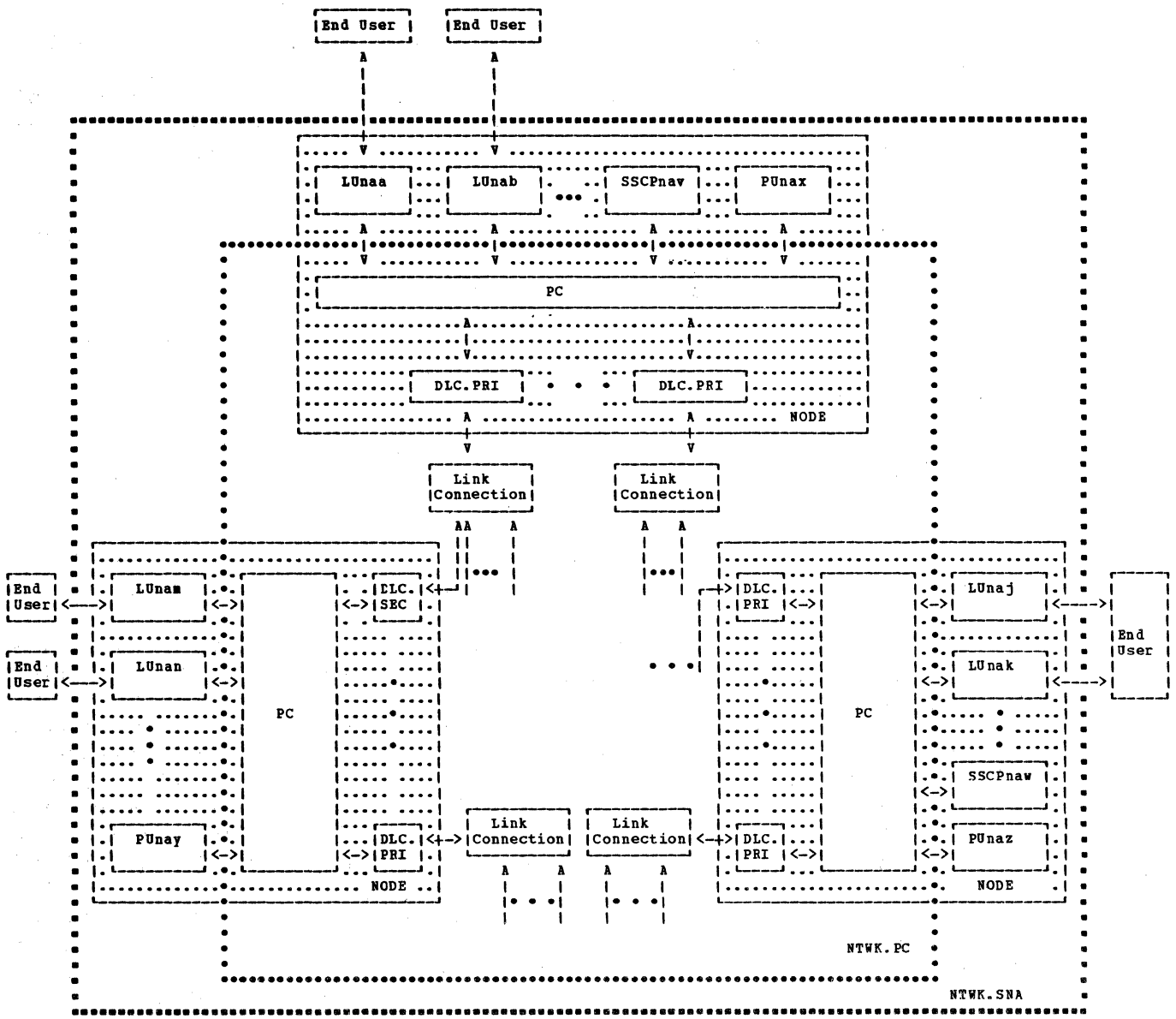
A node always includes a physical unit (PU), which controls the attached links and various other resources of the node. A PU has a type designation--PU_T1, PU_T2, PU_T4, or PU_T5--corresponding to the type (1, 2, 4, or 5, respectively) of node in which it resides. In this book, "type i node" (i = 1, 2, 4, 5) and "PU_Ti node" mean the same thing, and are used interchangeably.

A node typically also includes logical units (LUs), through which end users attach to the node, and thus to NTWK.SNA.

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

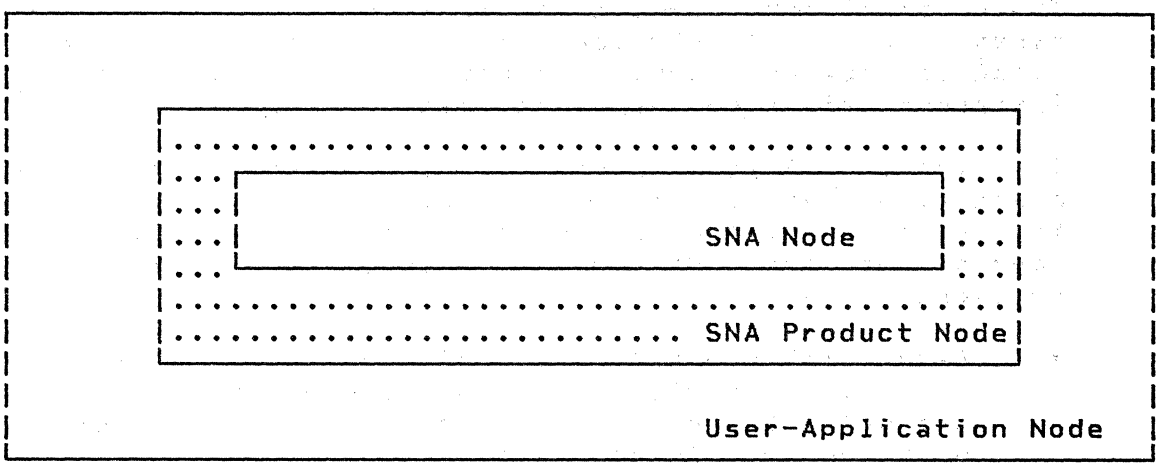
Type 5 nodes each contain a system services control point (SSCP). (Type 4 nodes do not--the primary architectural distinction between subarea node types.) An SSCP supports protocols for management and control of a domain. A domain consists of one SSCP and the PUs, LUs, links, and link stations (discussed in the next section, "Data Link Control Protocols and Links") 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 capability discussed in a later section ("Shared Control"). Each SSCP provides network services within its domain through protocols supported in conjunction with the PUs or LUs in the domain. The multiple SSCPs in a network jointly support cross-domain network services.

Types 1, 2, and 4 nodes each have an associated physical unit control point (PUCP), which provides a subset of SSCP functions, e.g., those relating to activation and deactivation of resources (such as link connections) local to the node. The PUCP is a product-defined subset of the functions described in Chapter 7, which discusses the configuration services component of the SSCP. A PUCP's relationship to product node services (e.g., for node operator interactions) varies according to product-specific requirements.

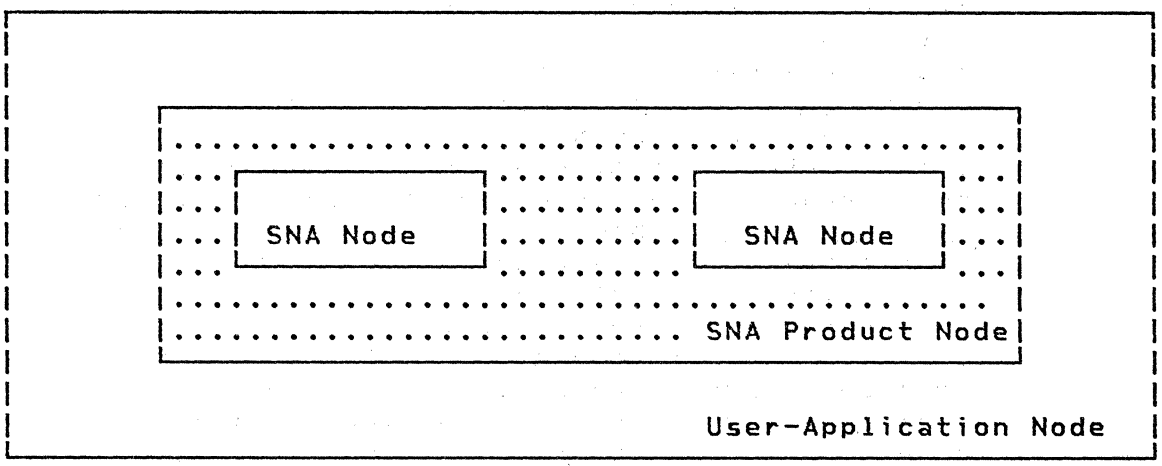


Note: Adjacent subarea nodes can be interconnected by multiple (or parallel) link connections (and DLC elements); a peripheral node can be interconnected with a subarea node by only one link connection.

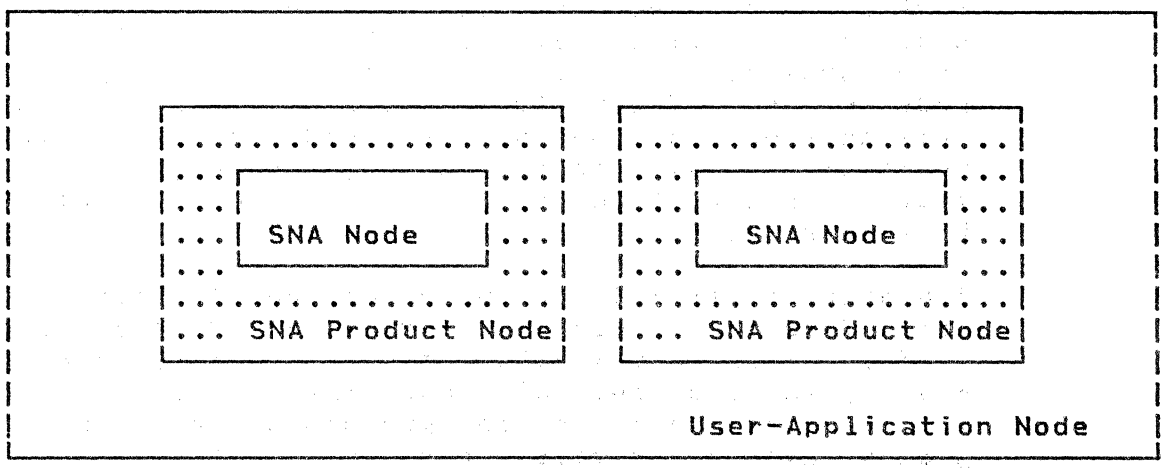
Figure 1-2. Node and Link Connection Structure of NTWK.SNA



(a) Typical Case



(b) Two SNA Nodes within an SNA Product Node



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

Figure 1-3. Examples of Nested Nodes

The physical interconnections between nodes of the different types are illustrated in Figures 1-4 and 1-5 (link-connection blocks are not shown). Multiple physical interconnections can exist between subarea nodes.

The various layers within nodes also provide logical connections between peer (equivalent layer) components in the same or different nodes (the layers and logical connections are discussed in additional detail in later sections):

- Data link control (DLC) elements in adjacent nodes, using various DLC protocols (such as SDLC or System/370 channel DLC), provide a common link appearance to the path control elements above them.

- The path control layer in subarea nodes has three sublayers: transmission group control (TGC), explicit route control (ERC), and virtual route control (VRC). TGC (the inner sublayer) elements provide one or more transmission group connections between adjacent subarea nodes; a transmission group can include one or more links. ERC (the middle sublayer) elements in two or more sequentially connected subarea nodes provide an explicit route connection between the two subarea nodes (not necessarily adjacent) at the termination points of the explicit route; the explicit route uses a set of transmission groups over which to transfer message units between the two ends of the explicit route. VRC (the outer sublayer) elements in subarea nodes provide a virtual route connection between half-sessions in the nodes; a virtual route has an underlying explicit route and a fixed transmission priority within the subarea routing portion of the path control network. Virtual routes connect half-sessions in the same or different subarea nodes directly; the path between a half-session in a subarea node and a half-session in a peripheral node consists of a virtual route from the subarea node half-session to the subarea node adjacent to the peripheral node, and then a route extension from the latter subarea node to the peripheral node half-session.

  The path control layer in a peripheral node provides basic functions, such as routing to and from multiple half-sessions within its node, but is restricted to routing to and from only one link, rather than many. It also does not provide protocols for transmission groups, explicit routes, or virtual routes.

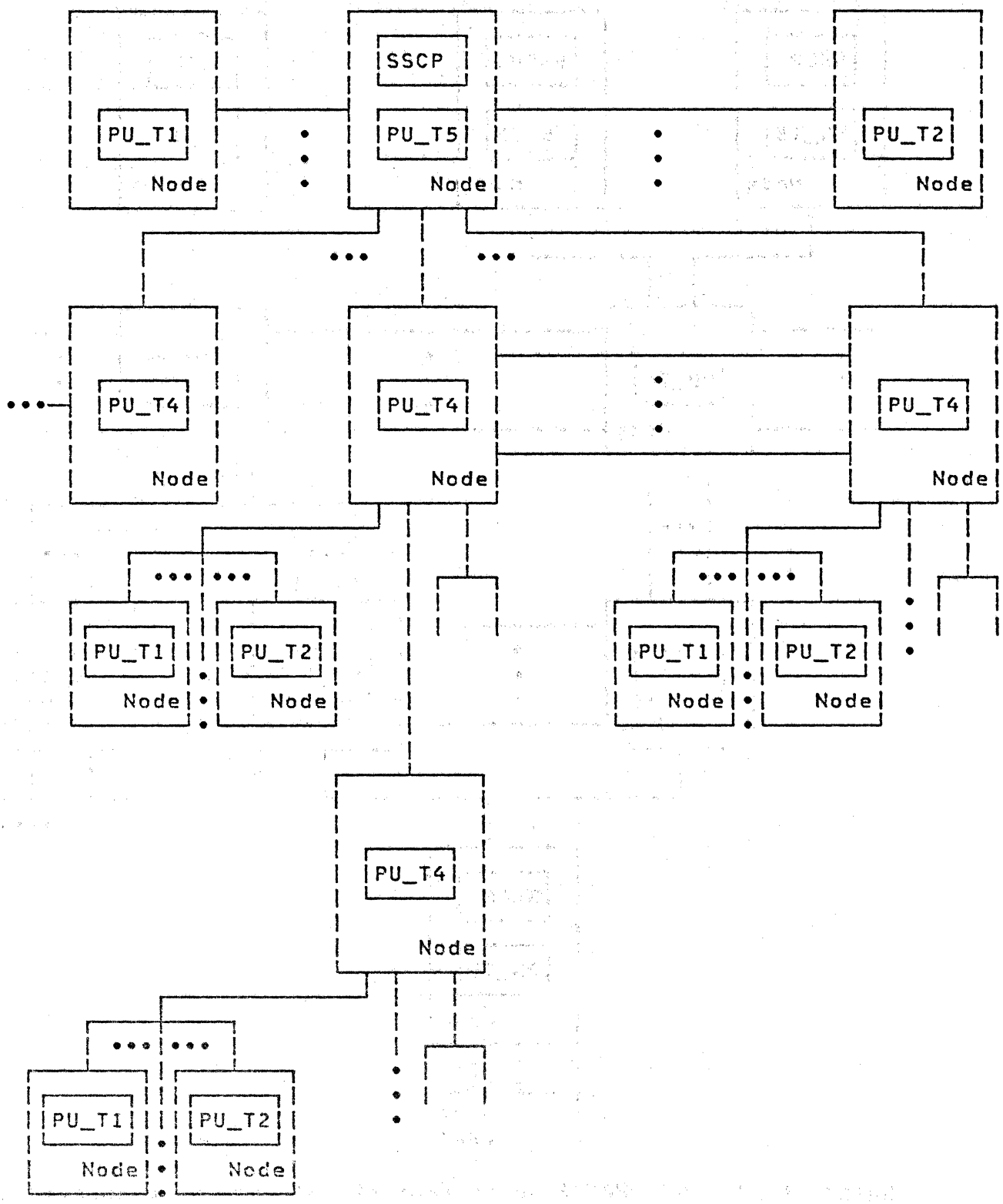
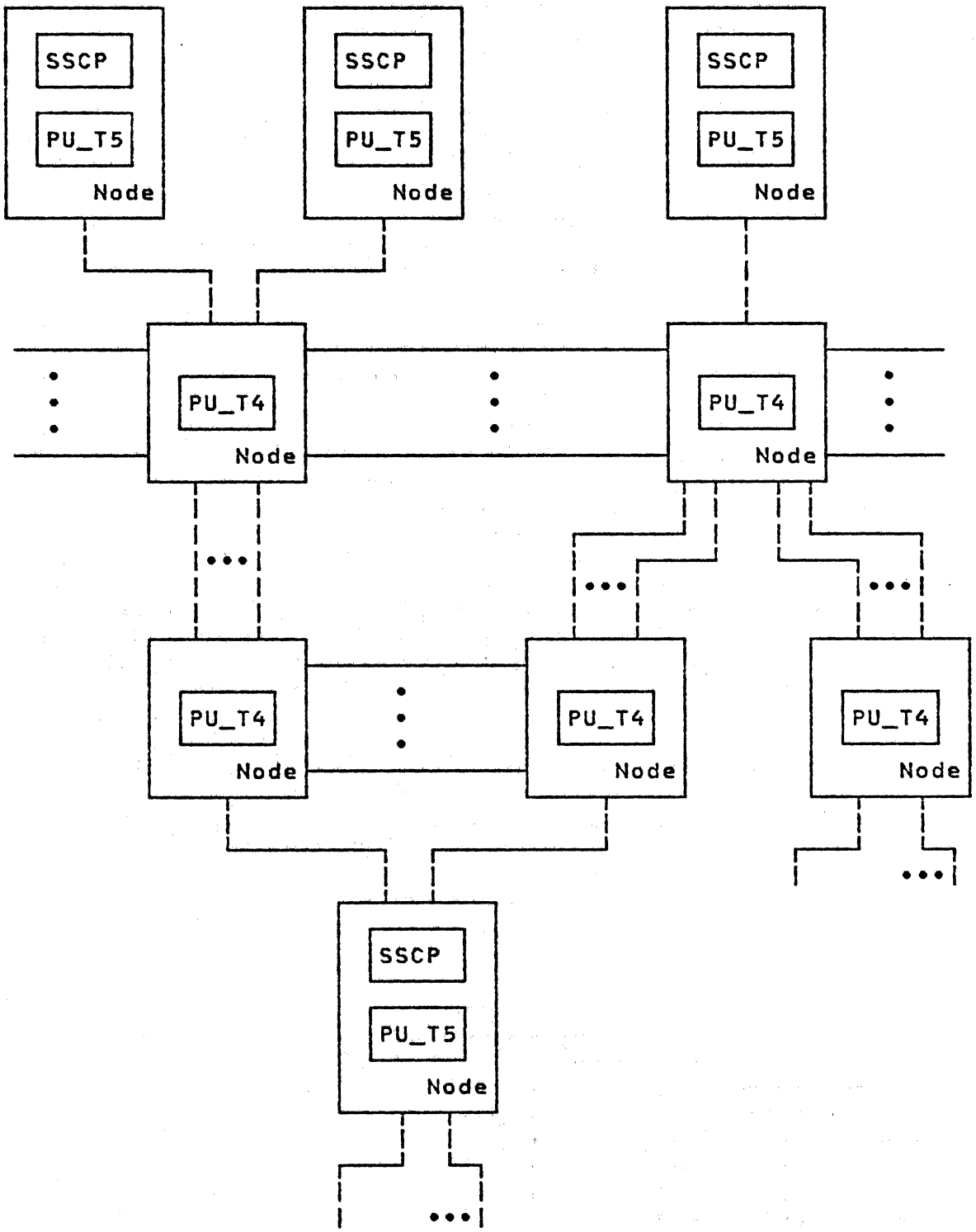


Figure 1-4. Node Interconnections within a Single-SSCP Network



Note: PU_T1 and PU_T2 nodes (not shown) may be connected to any PU_T4 or PU_T5 node.

Figure 1-5. Node Interconnections within a Multiple-SSCP Network

- Paired half-sessions provide a session connection between their using NAU services managers (and end users, if any), which provides various session options (discussed later) and a class of service provided by a specific virtual route. The class of service is derived by the SSCP, using a class of service name provided at session initiation; the SSCP maps the class of service name to a virtual route identifier list, from which the first available virtual route is chosen for the session to be assigned to, as part of session activation.

The table in Figure 1-6 lists the salient features of NAUs and of the various logical and physical connections defined in SNA. These concepts are defined further in this chapter. Much of this book is concerned with describing the initialization, activation, operation, testing, status reporting, and deactivation of these connections.

Definitions associated with link protocols are presented in the following section, preliminary to developing the details of the node protocols, which constitute the major portions of the book.

SNA CONSTRUCTS	SALIENT FEATURES
end users	End users may be terminal operators, application programs, or device media; they are not architected by SNA and are outside the SNA node. They gain access to the SNA network through LUs.
NAUs	SSCPs and PUs provide focuses for domain and node management; LUs provide access ports to the network for end users and can be used within a node to delimit distinct application subsystems' access to the network (e.g., within the same node, a data-base/data-communication subsystem typically uses one LU, while a remote job entry subsystem uses a different LU to access the network). LUs provide "anchor" points for resources such as files, device media, and transaction services, and protocols for accessing them--both by the attached end user and an end user attached to an LU at the other end of a session.
sessions	Sessions provide a means for NAUs to multiplex separate, independent interactions with other NAUs. Alternatively, multiple sessions from one LU to different LUs can be dependent and coordinated, e.g., via sync points (discussed below). Parallel (concurrent) sessions between the same two LUs are also possible. Different session activations can provide different options relating to data presentation services, traffic pacing, and concurrency of sending and receiving. Related data units can be chained in each direction for error recovery purposes. (They fail or succeed as a unit.) Brackets, consisting of one or more chains (and their responses) in both directions, can be used to divide a session's active period sequentially into transactions. One or more chains (within the same bracket) can be delimited by sync points so that session restart (after failure) can begin from a well-defined roll-back point, to limit duplicate processing. (Multiple sync points can be taken per bracket.) A class of service, realized by a specific virtual route (from a set of possible virtual routes) at session activation, provides a desired level of performance or security until session deactivation.
virtual routes	Each virtual route uses a fixed transmission priority on an underlying explicit route to provide a desired class of service for one or more active sessions. A session is assigned to a virtual route at session activation. Flow control within the subarea routing portion of the path control network is adaptive (using traffic pacing) and is based on virtual routes. Congestion control affects virtual-route (pacing) window sizes, based on queue depths for transmission groups traversed by the underlying explicit routes. Thus, window sizes are adaptively and selectively controlled by the load on the transmission groups that the virtual routes use.
route extensions	A route extension completes the path between two half-sessions when one is in a peripheral node; it includes the path control elements (one in the peripheral node and one in the boundary function support in the subarea node) and the link connecting the peripheral node to the subarea node at one end of a virtual route.
explicit routes	An explicit route connects two subarea nodes (not necessarily adjacent), using a fixed set of transmission groups (the same in both directions). Multiple virtual routes can have the same underlying explicit route.
transmission groups	A transmission group is a dynamic association of one or more links used to connect two adjacent subarea nodes. Traffic is routed evenly over the active links in the transmission group. The association of parallel links into a larger composite logical connection allows the latter to provide bandwidth and availabilty properties that exceed those provided by any component link. A transmission group is operative as long as at least one of its links is operative. A link can be dynamically assigned to one or another transmission group (via XID processing), but to only one at a time. A transmission group (and thus the explicit and virtual routes using it) fails only when its last remaining link fails. Session outage is then reported, and affected sessions can be restarted after being assigned to different virtual routes to bypass the failure. Multiple transmission groups can connect the same two adjacent subarea nodes. A transmission group can be used by multiple explicit routes. Transmission group queues are managed according to the transmission priorities that virtual routes impose on the explicit routes using the transmission groups.
links	A link connects two or more nodes using a link connection and a DLC protocol. A link, whatever its DLC, provides a view to its users of one or more adjacent link stations, which can be separately and concurrently accessed by its user.
link connections	A link connection physically connects two or more nodes, using various signaling protocols, such as bit-serial or bit-parallel transfer and analog or digital conventions, depending on the underlying physical media. It can be switched (using privately supplied or common-carrier dialing facilities) or nonswitched.

Note: Pacing is fundamental to SNA flow control. A pacing group (or window) consists of all the message units that a sending component can send to its peer component before a pacing response is received, indicating its receiving peer is ready to accept the next window of message units.

Session-level pacing applies to LU-LU and SSCP-SSCP flows and determines the rate at which each half-session can introduce data into the path control network. Window sizes in the two directions are fixed at session activation. Session-level window sizes are usually determined by the intrinsic rate at which each end can process received data.

Virtual-route pacing applies to all message-unit traffic introduced into the two ends of a virtual route by all (not just LU-LU and SSCP-SSCP) sessions concurrently assigned to the same virtual route. Window sizes in the two directions are set at virtual-route activation and adaptively changed thereafter, according to the congestion in their underlying transmission group queues. Virtual-route activation fixes the minimum and maximum window sizes for the virtual route.

To the extent that sessions share virtual routes (with common pacing), underlying explicit routes (with different transmission priorities imposed), underlying transmission groups (with common queues and congestion control), or underlying links (with multiple adjacent link stations), the sessions can affect each other's throughput and response time. Each link connection within a path between two half-sessions also has its own bandwidth and delay characteristics that affect performance.

Figure 1-6. Summary of SNA Constructs.

Data link control (DLC) supports protocols for (1) executing and coordinating the transfer of message units across a link connection between a single primary DLC user and a set of secondary DLC users, and for (2) performing link-level flow management and error recovery procedures.

A link (see Figure 1-7) is a composite protocol machine for executing and coordinating the transfer of message units between a single primary link-station user and a set of addressed secondary link-station users.

Associated with each secondary link-station user are two link-level address parameters, Ak and ak. Ak is a set of receive addresses and ak is a unique send address; ak may be in Ak (i.e., the send address may be one of the receive addresses).

A link is composed of:

- A single primary link station, DLC.PRI_LINK_STA (usually abbreviated "DLC.PRI" elsewhere in this book), associated with the primary link-station user

- A set of secondary link stations, DLC.SEC_LINK_STA_(Ak,ak) (usually abbreviated "DLC.SEC"), one associated with each secondary link-station user; each secondary link station is qualified by a receive-address set and a unique send address

- A single link connection that transmits message units between the primary link station and each secondary link station

Thus, a link consists of a particular DLC and the link connection underlying it.

Regardless of the DLC protocols used, a link always presents its using path control component a view of one or more adjacent link stations. Within a node containing a primary link station for a given link, each secondary link station of the link is represented (by a control block) as an adjacent link station. Within each node containing a secondary link station of the given link, only the primary link station of the link is so represented--the other secondary link stations of the link are unknown.

A link-station user identifies a particular adjacent link-station control block in passing a message unit to DLC to transmit. (Similarly, DLC identifies to its link-station user the adjacent link station associated with a message unit received over the link connection.) DLC uses the

identified control block to extract link-level addressing information for its use in transmitting over the link connection.

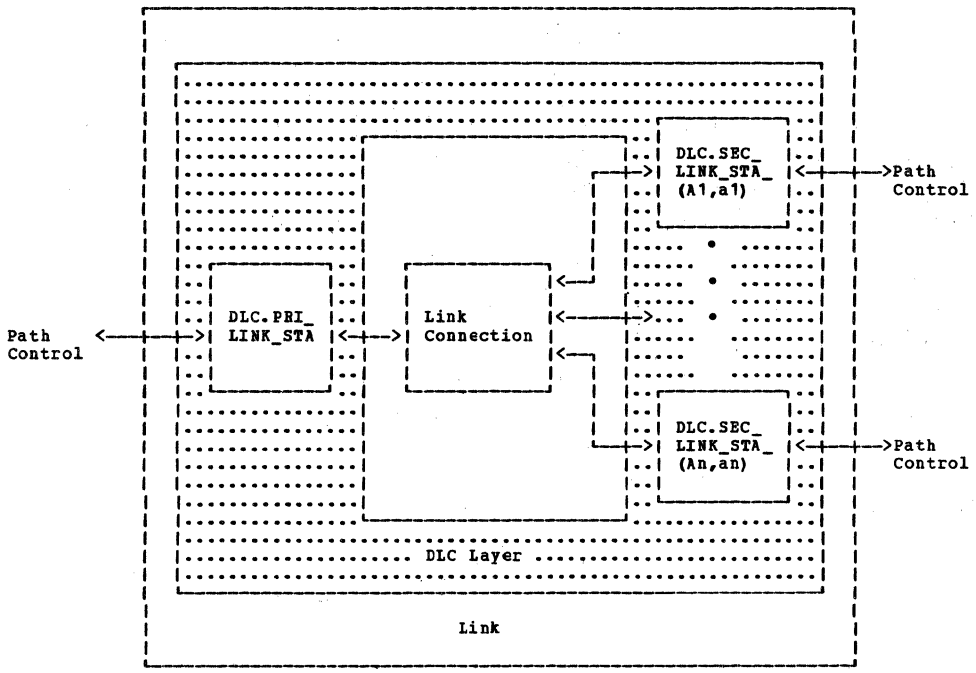
Inputs to a link connection from a primary link-station are of the form (a, MSG) where "a" is a single secondary link-station address; this results in MSG being accepted by the DLC.SEC_LINK_STA_(Ak,ak) for which "a" belongs to Ak. DLC keeps multiple message units to the same secondary link-station user in order; i.e., message units are delivered to each in the order submitted by the primary link-station user.

For its input to the link connection, the seconuary link station appends the appropriate ak to MSG and transmits the resulting (ak, MSG) to the primary link-station. A link keeps multiple message units from the same secondary link-station user in order.

In SNA, the link-station users are the path control components. The message units passed by a link generally have two components: link user information and link control information. Link user information is passed transparently by a link and does not affect the state of the link. Link control information may sometimes be passed to the link users; however, its main purpose is to change the state of a link, thus effecting either a normal control change (to synchronize user interaction) or a recovery from an error situation.

A link manager (PU.SVC_MGR.LINK_MGR) within the PU (but not discussed in detail in this book) also has a protocol boundary with both the DLC.(PRI|SEC) in its node and the link connection. (One link manager exists for each attached link.) The link manager exchanges signals with DLC relating to link-level initialization to carry out SSCP-to-PU requests (e.g., CONTACT for an adjacent link station), and receives failure reports from DLC. The link manager also exchanges signals with the link connection; for example, a CONNECT OUT request received from an SSCP causes the link manager to initiate a dialing operation to a switched link connection.

DLC layers within links can be implemented using various physical and logical means, thus producing specific DLC protocols, such as System/370 channel DLC and SDLC (with point-to-point, multipoint, or loop configurations). DLC protocols can be distinguished from each other by their specific link station protocols, and by the effects that the DLC control-portions of message units can have on these protocols. Details of specific DLC architectures are not discussed in this book.



Note: The DLC layer (shaded portion) consists of the DLC.PRI_LINK_STA and all the DLC.SEC_LINK_STAs attached to the same link connection; the link is the composite protocol machine consisting of the DLC layer and the link connection.

Informally, the DLC components are referred to simply as link stations. Within the node containing the primary link station, all the secondary link stations of the given link are represented as adjacent link stations. Within each node containing a secondary link station, the primary link station is represented as an adjacent link station; the other secondary link stations of the given link are unknown to the node.

Figure 1-7. Structure of a Link

ADDRESSING RULES

The set of network addresses is partitioned based on the network addressable units (NAUs)--where a NAU is defined to be an SSCP, a PU, or an LU--and the links and link stations in the network as follows:

- Each system services control point (SSCP) is assigned a network address. Each NTWK.SNA contains at least one SSCP, which resides in a type 5 node (see Figures 1-4 and 1-5), and has special responsibilities to monitor and control all the resources of its domain; together, the SSCPs provide this capability for the network as a whole. (A physical unit control point (PUCP) is a subset version of an SSCP, and exists in each type 1, 2, or 4 node. It is known, only within its node, by a unique network address (type 4 node) or signaling convention (peripheral node); it is not addressed from outside its node. Similarly, a PUCP does not address anything outside its node.)

- Each physical unit (PU) is assigned a network address. (The network address of a PU in a peripheral node is the same as the network address used in the subarea node to which it is attached to identify the adjacent link station associated with the peripheral node.) A PU monitors and controls various resources of its node.

- Each logical unit (LU) in a peripheral node is assigned a single network address and each LU in a subarea node is assigned one or more network addresses. LUs provide the ports by which end users of the network can access network services and communicate with other end users.

- Each link and adjacent link station attached to a subarea node is associated with a distinct network address in that node. For a switched link connection, the network address of an adjacent link station is 1 greater than the network address used for the link. (Within the subarea node, the subarea address of an attached link or adjacent link station is implied--being the same as for the node itself--and, thus, the element address is sufficient to identify a particular link or adjacent link station.) The link network address is used in such control requests from an SSCP as ACTIVATE LINK, which causes the protocol boundary between a specific link connection and DLC.PRI|SEC (link station) within the node to be activated. The network address of an adjacent link station attached to a subarea node is carried in CONTACT, which initiates a DLC-level activation interchange between the link station in the subarea node and the specified adjacent link station. (Network

addresses identifying links and adjacent link stations are carried only within request/response units (RUs), never in transmission headers (THs)--see the next section, "Message-Unit Formats and Parameters.")

The nodes of NTWK.SNA are grouped into addressing subareas (Figure 1-8); each subarea node is assigned to a unique subarea. All the peripheral nodes attached to a subarea node are also assigned to its subarea. Each network address (na) is 16 bits long and consists of two parts:

- A subarea address
- An element address

The lengths of the subarea address field and the element address field are network dependent (i.e., can vary from one network to another) subject to the constraints:

- The subarea address field can range in size from 1 to 8 bits; the element address field, from 8 to 15 bits (the sum of the two fields being always 16 bits).
- The two lengths are constant everywhere in the network.

Chapter 2 provides details of their formats and uses within headers.

All NAUs within nodes in the same subarea, and links and adjacent link stations known to the subarea node, have identical subarea addresses, but distinct element addresses. By convention, the element address of the PU_T4 or PU_T5 within each subarea is 0. Each NAUna within a subarea node is always known by its network address, na; an LU in a subarea node can be known by more than one network address, if it supports parallel sessions (discussed in a later section, "Parallel Sessions"). Each NAUna within a peripheral node is known not only by its network address, na, but also by another address, na', that is local to (and unique within) its peripheral node. (Note: na' is not the element portion of the network address, nor is it necessarily unique, except within a peripheral node.)

The ability to use short-form, local addressing simplifies the protocols in peripheral nodes. Their insensitivity to the network addresses also greatly simplifies reconfiguration procedures in the network as a whole. For example, NAUs within distinct peripheral nodes may have identical local addresses; only the subarea nodes in the network need to update routing tables when network addresses are changed, added, or deleted. Local addresses need not be affected by these network address changes.

The translation from network addresses to local addresses (as well as other simplifying protocol support) is provided to each peripheral node by the boundary function in the adjacent subarea node, discussed later in this chapter ("Boundary Function Structure").

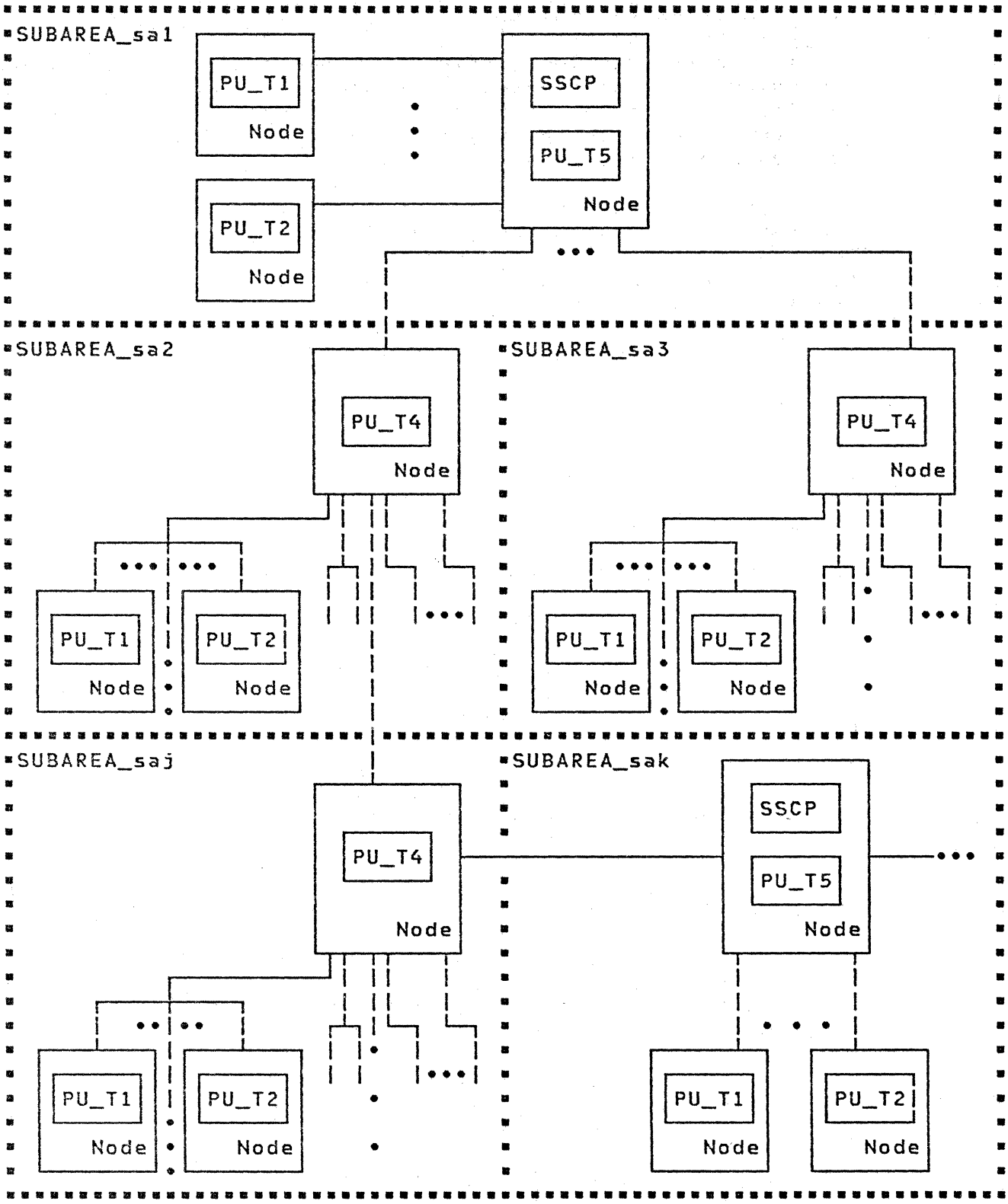


Figure 1-8. Subarea Structure of NTWK.SNA

MESSAGE-UNIT FORMATS AND PARAMETERS

Message units flowing within NTWK.SNA take various forms, depending on the protocol boundary being crossed. A detailed discussion of message unit formats is presented in Chapter 2. For the purposes of this introductory chapter, two important message units are introduced.

An important message unit flowing within NTWK.PC, the innermost nested network in NTWK.SNA, is the path information unit (PIU). Except when segmenting is performed (see Chapters 2 and 3), each PIU (Figure 1-9) consists of three parts:

- A request/response unit (RU)
- A request/response header (RH)
- A transmission header (TH)

The RH-RU combination is called a basic information unit (BIU).

The RU may contain:

- End-user data and/or
- Control information generated by an SNA protocol machine; different control categories and formats exist--Appendix E gives details on these formats.

The RH contains several fields, including:

- The Request/Response indicator, which denotes whether the BIU is a request (RQ) or response (RSP)
- The Response Type indicator, by which responses can be qualified as positive (+RSP) or negative (-RSP)
- An RU Category field indicating the functional category of the request or response; four categories exist: function management data (FMD)--used for end-user data and for network services--data flow control (DFC), session control (SC), and network control (NC); these correspond to structural components to be described in subsequent sections of this chapter, as well as in later chapters.

A TH may assume one of six format types, four types (called FID0, FID1, FID4, and FIDF) in which the destination and origin of the TH are represented as unique network addresses, and two types (called FID2 and FID3) in which the destination and origin are represented by locally unique short-form addresses. Details on the format types and their

relationship to boundary function protocols are presented later in this chapter ("Boundary Function Structure"). Details of the formats of the various THs (and of the RH) are given in Chapter 2, and in summary form in Appendix D.

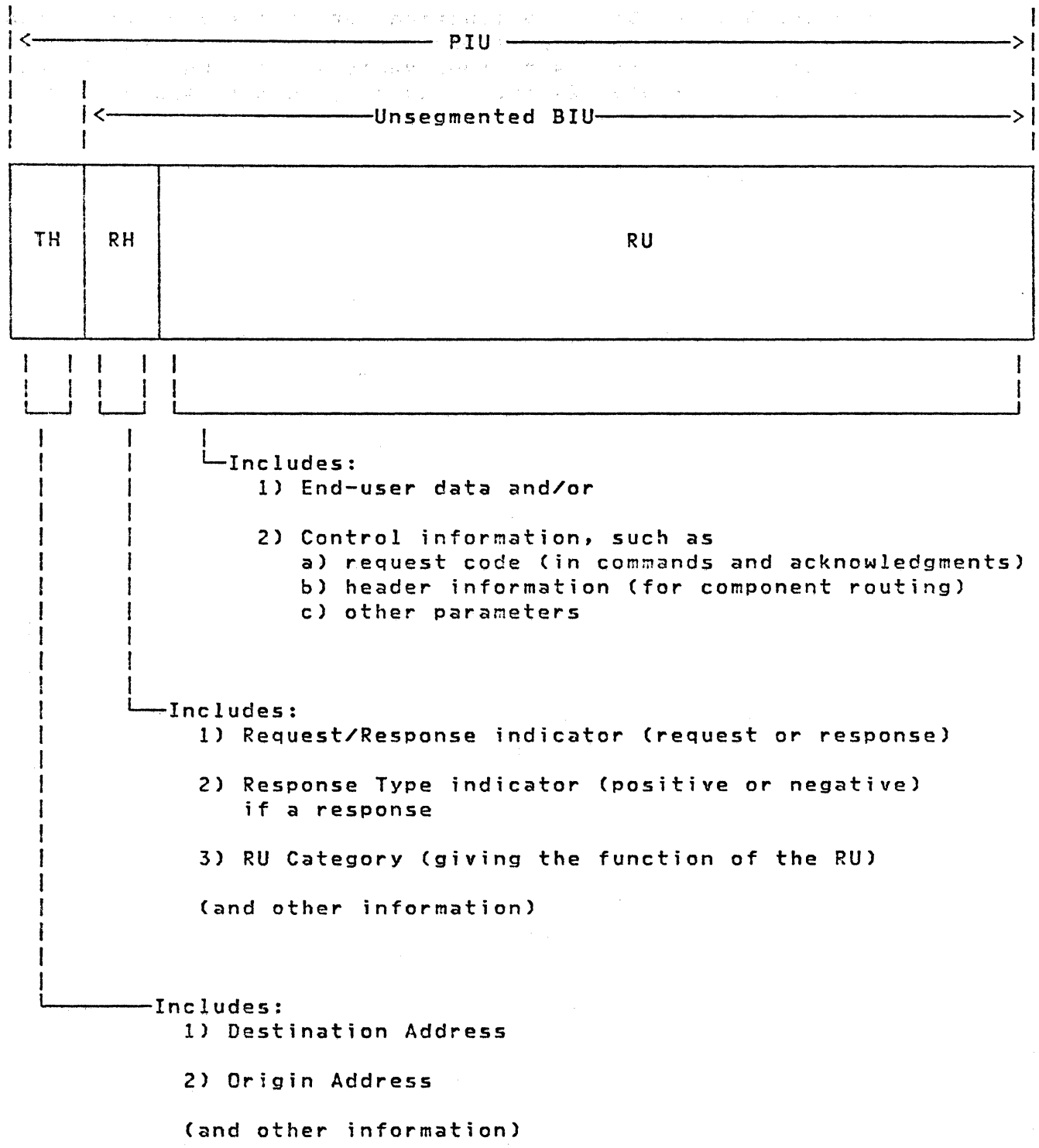


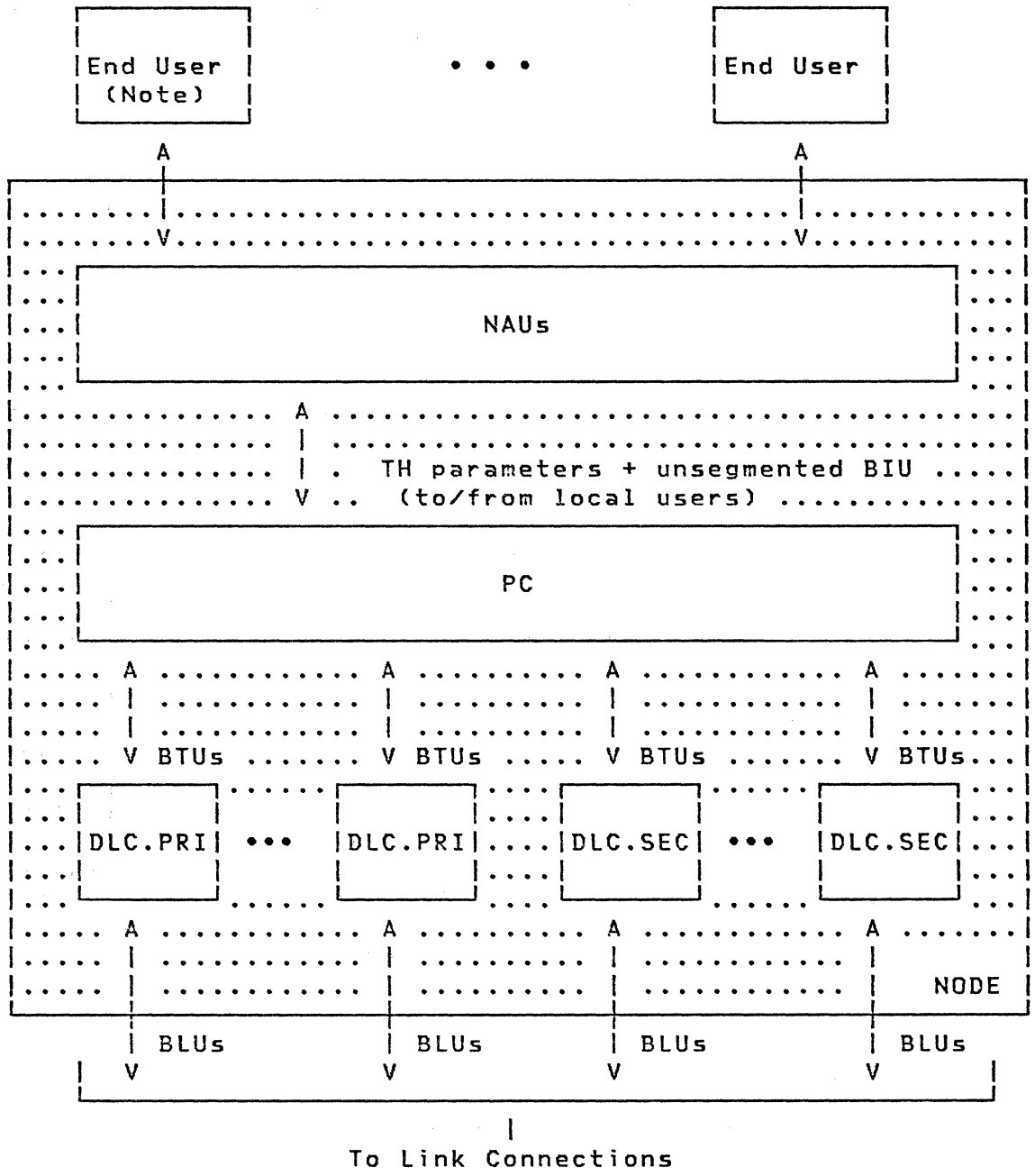
Figure 1-9. Basic Message Format (without segmenting)

PATH CONTROL AND DATA LINK CONTROL WITHIN A NODE

Each node has the PC and DLC structure illustrated in Figure 1-10. This includes a set of primary and secondary DLC protocol machines, which interact with path control (PC). LUs are interposed between PC and the end users associated with the node.

All the DLCs interact with a single PC element (i.e., one per node). The principal purpose of PC is to route PIUs, based on the destination and origin addresses in the TH. PIUs may be received by PC from local NAUs or from the network through DLC. Based on the addresses in the TH of the PIU, PC (1) routes the PIU to a local NAU (e.g., a half-session) or boundary function component (using the origin address as well as the destination address to do this routing) if the destination address is that of a local NAU or boundary function component, or (2) routes the PIU to a specific DLC for transmission out of the node. An important exception concerns session-activation and -deactivation requests and responses. Those received (whether from a local NAU or from DLC) for a local NAU are always sent to the PU.SVC_MGR.CSC_MGR (see the section, "NAU.SVC Structure"). PC also supports blocking of PIUs or segmenting of BIUs (see Chapters 2 and 3), and routing and flow control over the PC logical connections (virtual routes, explicit routes, and transmission groups) described earlier. (See the next section, "NTWK.PC," for additional discussion.)

LUs, above PC within a node, (1) convert BIUs to end-user information and control information (and vice versa), and (2) control the flow of information to and from the end users as directed by the control information.



LEGEND
BIU: Basic Information Unit
BLU: Basic Link Unit (a BTU with added DLC header-trailer information)
BTU: Basic Transmission Unit (one or more PIUs)

Note: End users have a protocol boundary only with LUs.

Figure 1-10. Basic Node Structure, Emphasizing PC and DLC

NTWK.PC

The system consisting of all interconnected PCs and DLCs forms the path control network, or NTWK.PC (see Figure 1-2); the extension of this concept to include boundary function is presented later in this chapter and in Chapter 3. The input/output streams of NTWK.PC consist of streams of TH parameters and associated unsegmented BIUs.

Each node has a PC element and NAUs. The node and DLC configuration of the network, and the PC routing algorithms, combine to provide the following behavior for NTWK.PC:

- An input to a PC element in node-i from a NAU is transmitted and routed by NTWK.PC 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 (destination, origin) identifiers are emitted by NTWK.PC in the order submitted by the origin NAU.

In an earlier section, "NTWK.SNA--Nodes and Their Physical and Logical Interconnections," the path control logical connections for subarea routing--virtual routes, explicit routes, and transmission groups--were discussed. Chapter 3 discusses the sublayers of path control--virtual route control (VRC), explicit route control (ERC), and transmission group control (TGC), which control routing and traffic flow over the logical connections. Chapter 12 discusses the virtual route (VR) manager and explicit route (ER) manager within PUs (no transmission group manager is needed), which control the activation, assignment, testing, status reporting, and deactivation of virtual and explicit routes. (See, also, the later section, "NAU.SVC Structure," for a brief overview.)

A transmission group (TG) includes one or more links between two adjacent subarea nodes, and can be identified by the triple (SA1, SA2, TGN), where:

- SA1 is the address of the subarea at one end of the transmission group.

- SA2 is the address of the subarea at the other end of the transmission group.

- TGN is the number (1-255) assigned to the transmission group.

An explicit route (ER) includes a set of transmission groups connecting subarea nodes. These routes provide connectivity, using a fixed set of transmission groups, from one subarea node to another (not necessarily adjacent) within the network. It is a bidirectional logical connection between subareas and can be identified by the quadruple (SA1, SA2, ERN, RERN), where:

- SA1 is the address of the subarea at one end of the explicit route.
- SA2 is the address of the subarea at the other end of the explicit route.
- ERN is the explicit route number carried in PIUs transmitted from SA1 to SA2.
- RERN is the explicit route number carried in PIUs transmitted from SA2 to SA1 (and is referred to as the reverse explicit route number). The RERN may be a value different from the ERN.

A maximum of 16 explicit route numbers exists for each direction of flow between any two subarea nodes.

PIUs in the network move from subarea node to adjacent subarea node based on routing lists indexed by the destination subarea address and explicit route number carried in the TH. The original source of the PIU has no bearing on the routing of the PIU. The RERN also has no effect on the route taken to the destination node (and is not carried in the transmission header).

A virtual route (VR) logically connects the subareas in which the NAUs participating in a session reside, building high-level flow-control properties onto the connectivity provided by explicit routes. Message-unit integrity is enhanced by sequence numbering within the virtual route. (A route extension is the connection between a peripheral node and the subarea node that is the terminus of the virtual route.) A virtual route is a bidirectional logical connection between subareas, and can be identified by the quadruple (SA1, SA2, VRN, TPF), where:

- SA1 is the address of the subarea at one end of the virtual route.
- SA2 is the address of the subarea at the other end of the virtual route.
- VRN is the number assigned to the virtual route.
- TPF is the transmission priority assigned to the virtual route.

PIUs are transmitted over the explicit route (or set of TGs) underlying a virtual route according to transmission priority. Up to 16 virtual route numbers (VRNs) and 3 transmission priorities (low, medium, and high) can be used between any two subarea nodes, yielding up to 48 virtual routes.

Two uses of virtual routes are worthy of note:

- All SSCP-PU and SSCP-LU sessions for the PUs and LUs in the same subarea use the same virtual route to a given SSCP. The reason for this is discussed in the later section, "Session Outage Notification."

- All sessions involving NAUs in the same subarea are assigned to virtual routes without underlying explicit routes. These sessions connect NAUs in the same subarea node or in a subarea node and an attached peripheral node. Use of virtual route pacing and other flow control is not needed, because the virtual route lies wholly within the subarea node. (A route extension, using boundary function path control, completes the path to a peripheral PU or LU.)

Just as the primary-secondary DLC asymmetries and other DLC details are hidden from PC, so the routing and other concerns of NTWK.PC are not visible at the protocol boundary between NTWK.PC and the NAUs; in particular, NTWK.PC conceals the node interconnections (virtual and explicit routes, transmission groups, links).

SESSIONS AND PU-PU FLOWS

NAU services managers, and therefore the NAUs themselves, interact with each other as pairs, via a protocol machine called a session (see Figure 1-11). The following session pairings are defined:

- Two distinct LUs in a network can be paired to form a session (or more than one session--see the later section, "Parallel Sessions"); for a given session activation, one of them is the primary (and sends the session activation request), while the other is the secondary (and receives the session activation request)--in general, the primary services manager can initiate more protocol features related to session activation and recovery than the paired secondary services manager. (While the session is active, they may have symmetric roles.) LU-LU sessions are used for end-user to end-user communication.

- An SSCP is paired to form a session with each PU in its domain, and one with each LU in its domain; the SSCP is always the primary for the session. SSCP-PU and

SSCP-LU sessions are used for monitoring, controlling, and accessing the processing and communication resources of the network. For example, each SSCP uses SSCP-PU sessions to request activation of links within its domain; LUs use the SSCP-LU sessions to request activation of LU-LU sessions and to receive directory services (e.g., name-to-address translation) from the SSCPs. For the purposes of communication network management (discussed in a later section), the SSCP can provide, via coordinated LU-SSCP and SSCP-PU sessions, LU-PU routing services within a domain.

- Two distinct SSCPs in a network can be paired to form a session; the SSCP sending the successful session activation request is the primary for the duration that the session remains active; the receiver of that request is the secondary. SSCP-SSCP sessions are used for cross-domain services within a network; in particular, they are used for coordinating the activation of cross-domain LU-LU sessions.

Another kind of pairing, using PU-PU flows, exists between PU services managers. These flows are used to:

- Load a peripheral node from its adjacent subarea node (in reaction to a request from an SSCP to the subarea PU).

- Activate and test explicit routes and report their status (e.g., operative, activated, inoperative); information and requests are propagated from subarea PU to subarea PU in either sequential (PU to one adjacent PU) or fan-out (PU to many adjacent PUs) fashion as described in detail in Chapter 12.

- Activate and deactivate virtual routes; each such PU-PU flow connects the PUs in the two subarea nodes where the virtual route terminates.

PU-PU flows do not require session activation requests; PU-to-PU awareness results from Exchange Identification (XID) processing (see Chapter 11 and Appendix E) or is part of system-definition initialization of nodes.

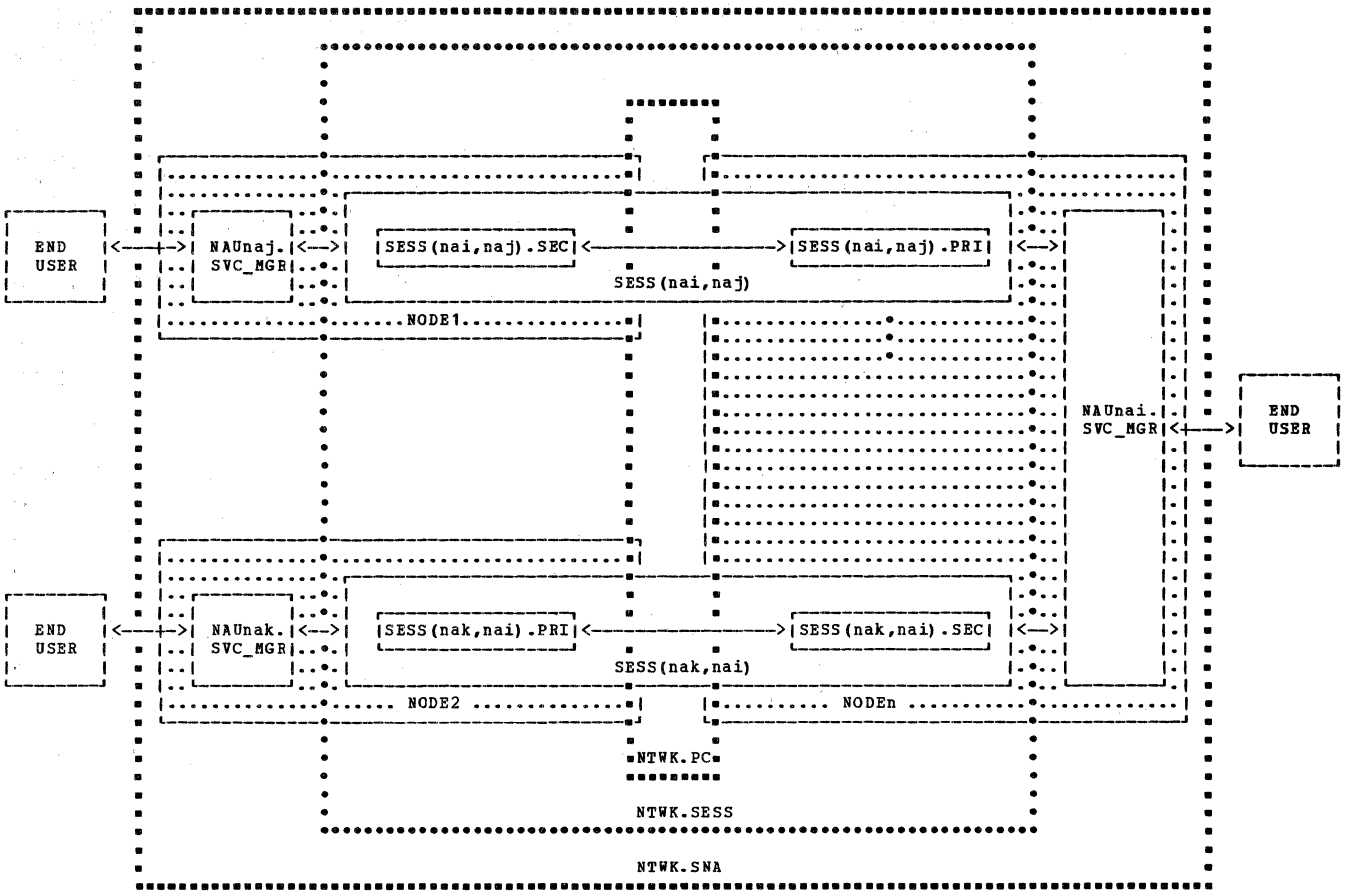


Figure 1-11. Basic Session Structure

Each session (SESS) is qualified by a session identification (SID) specifying network addresses of the NAUs for the session.

The SID associated with the session between NAUnai and NAUnaj is the network address pair (nai,naj). The session between NAUnai and NAUnaj is denoted by SESS(nai,naj).

SIDs are always ordered address pairs and the notation SESS(nai,naj) not only refers to the session between NAUnai and NAUnaj, but by convention also explicitly specifies that NAUnai is the primary and NAUnaj, the secondary for that session.

Each session, SESS(nai,naj), consists of two half-sessions interconnected via the protocols of NTWK.PC. The half-session associated with NAUnai is denoted SESS(nai,naj).nai and that associated with NAUnaj is denoted SESS(nai,naj).naj; the half-session may also be denoted by SESS(nai,naj).PRI for the primary or SESS(nai,naj).SEC for the secondary. Whenever it is not ambiguous, half-sessions are denoted by the short-forms:

- (nai,naj).nai, (nai,naj).PRI, SID.nai, or SID.PRI; and
- (nai,naj).naj, (nai,naj).SEC, SID.naj, or SID.SEC.

The half-session identification, HSID, is the generic term for SID.(nai|naj) or SID.(PRI|SEC). (Frequently, the addresses are not essential to the meaning, and a half-session is designated simply in terms of the types of NAUs involved--for example, (SSCP,LU).PRI.)

At any time, the state of SESS(SID) consists of the state pair:

(state of SID.nai, state of SID.naj).

The only direct communication between paired half-sessions is via NTWK.PC, which provides a signaling path that can exhibit a delay. A principal function of the half-sessions is to synchronize session states in the face of this delay.

PARALLEL SESSIONS

In some cases, two LUs may be paired with each other to form multiple sessions; each such session is then a parallel session between the two LUs. Before two LUs can form parallel sessions, at least one of them must be assigned multiple network addresses.

Each LU has either a single or multiple network addresses. An LU with a single network address can participate in parallel sessions only as a secondary. An LU with multiple network addresses has:

- A single secondary network address, which is used for all secondary half-sessions, including the one with the SSCP.

- Multiple primary network addresses (distinct from the secondary network address), each of which is used for primary half-sessions. Primary network addresses are assigned (for example, via RNAA, discussed in Chapter 7) during session initiation and can be freed (for example, via FNA, discussed also in Chapter 7) when all sessions using them have been terminated. Only LUs in subarea nodes are assigned primary network addresses.

Figure 1-12 shows an example of parallel sessions. LU(nai|naj|nak) and LU(nam|nan|nap) together form a total of four parallel sessions--SESS(nan,nai), SESS(nap,nai), SESS(naj,nam), and SESS(nak,nam). An LU is also capable of having multiple sessions, including parallel sessions, with different LUs; for example, LU(nai|naj|nak) is shown as also having two parallel sessions with LUnaq and a single session with LUnar. Multiple primary half-sessions may make use of any given primary network address assigned to the LU.

LUs distinguish individual parallel sessions by session name (see the discussion of BIND in Chapter 13), which is used during session reactivation processing following an outage. See the section, "Sync Points," for additional discussion.

The distinction between parallel and nonparallel sessions arises primarily during the initiation and termination of sessions. LU-LU session initiation and termination is discussed in Chapter 8). Thus, the term "parallel" is omitted whenever it is not ambiguous to do so.

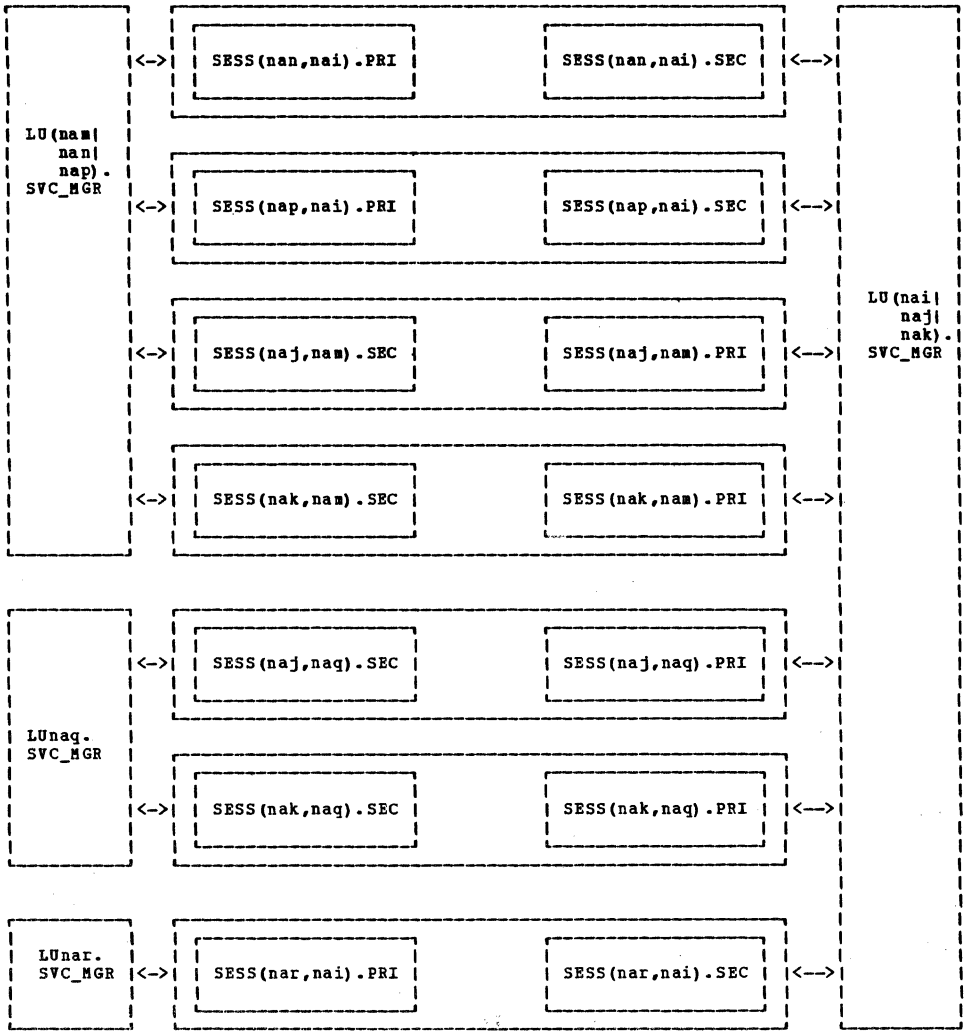


Figure 1-12. LU-LU Parallel Sessions Example

FLOWS WITHIN HALF-SESSIONS

The flow of RUs within a half-session is divided into normal and expedited components in each direction (to and from the paired half-session). In each direction, the normal- and expedited-flow RUs are independently sequence numbered (or identified) and the normal and expedited flows are controlled under separate protocols. Control coupling exists to the extent that commands carried on the expedited flows can change the state of the normal flows--for example, resetting sequence numbering or quiescing traffic. Within half-sessions (and also within boundary function components, discussed later in this chapter), expedited-flow RUs bypass normal-flow queues, thereby passing normal-flow RUs in these queues.

The provision for normal and expedited flows within NTWK.SNA allows various useful session-level flow-control protocols to be imposed on end-user data traffic (which is carried on normal flows) without blocking the passage of crucial control traffic (which is carried on expedited flows). Use of the normal and expedited flows within half-sessions varies by RU category as follows:

- FMD RUs--end-user data and network services requests and responses--are sent only on the normal flow.
- DFC RUs are sent on either the normal flow or the expedited flow, depending on the particular request code.
- SC RUs are sent only on the expedited flow.

A fourth RU category, NC, applies only to PU-PU flows (on which the other three categories, conversely, are never used). PU-PU flows use only the expedited flow and do not involve sessions.

The protocols for handling normal and expedited flows within half-sessions are defined in Chapters 4 and 5.

HALF-SESSION STRUCTURE

Each half-session denoted by the half-session identification, HSID, has the structure shown in Figure 1-13, and consists of:

- Transmission control (HSID.TC)
- Data flow control (HSID.DFC)
- Function management data services (HSID.FMDS)

TC provides basic control of the use of the transmission resources of NTWK.PC by:

- Activating and deactivating the half-session data traffic via session control (SC) requests and responses

- Sequence number checking, session traffic pacing, data enciphering and deciphering, and enforcing maximum RU size of normal-flow traffic passing between TC and PC

DFC half-session protocols include capabilities to:

- Control the concurrency of send and receive operations--one way at a time or both ways concurrently

- Provide request groupings through chaining protocols and transaction delimiting through bracket protocols

- Control the interlocking of requests and responses in accordance with the request and response control modes selected during session activation; this control involves the return order of responses and the number of requests allowed to be awaiting responses on a given flow

- Assign sequence numbers to normal-flow requests

- Correlate requests and their responses

- Interrupt the flow of data in either direction without affecting other control protocols of the session

Some DFC commands are carried in RUs; other DFC commands are carried in RHs and are handled as the message unit and RH parameters pass through the DFC element.

FMDS manages FMD RUs in cooperation with the appropriate NAU services manager. The particular functions provided by FMDS vary by the type of half-session:

- The FMDSs in half-sessions involving an SSCP (i.e., (SSCP,PU|LU).(PRI|SEC) and (SSCP,SSCP').(SSCP|SSCP') half-sessions), because of their network services functions, are more specifically referred to as session network services (SNSs). They provide protocols (in conjunction with the various NAU services managers) through which the SSCPs can monitor and control the processing and communication resources of the network.

- The FMDSs in (LU,LU).(PRI|SEC) half-sessions may provide presentation services protocols for encoding and compression of data, display formatting, and so forth; they are more specifically referred to as session presentation services (SPSs).

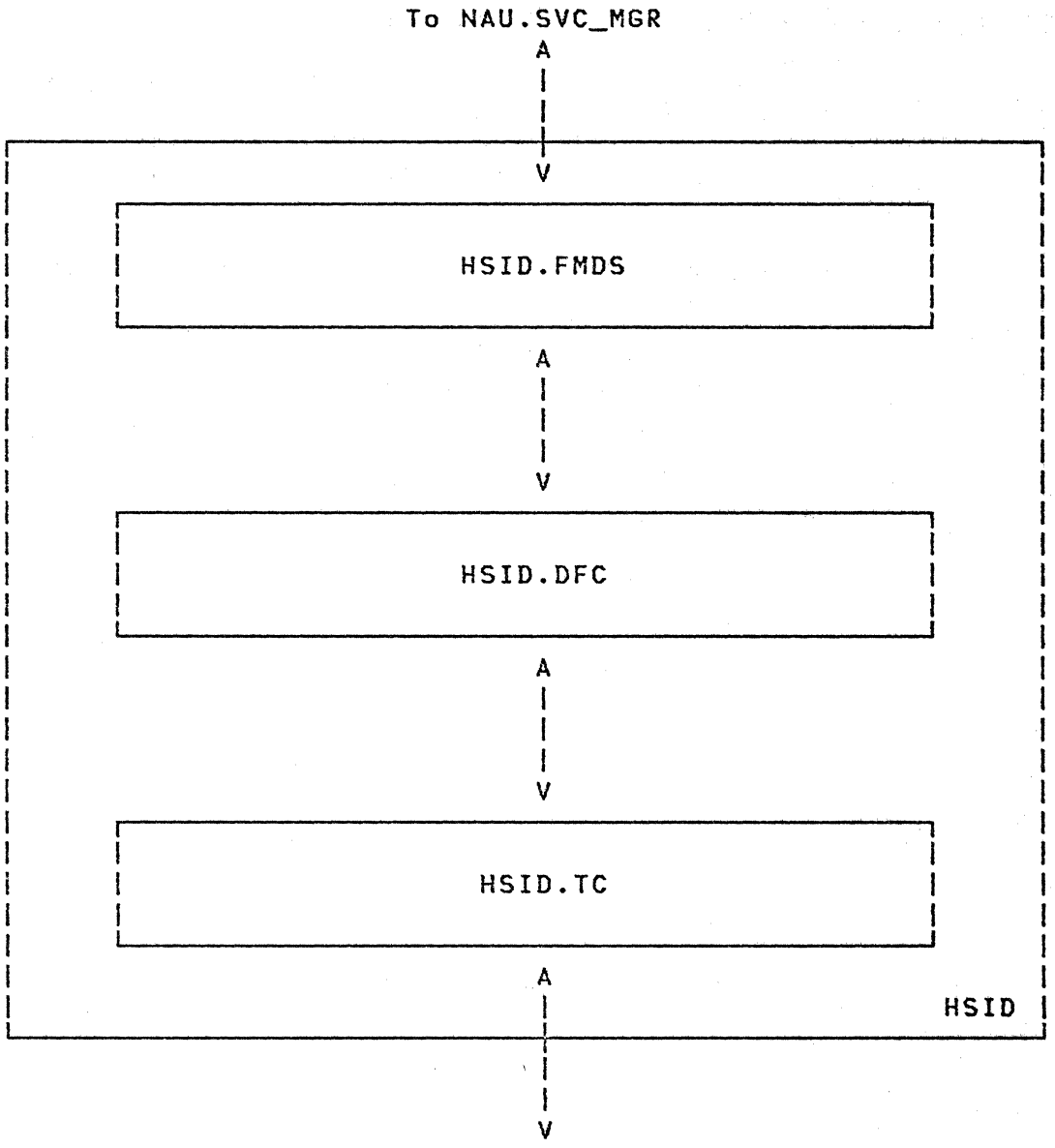


Figure 1-13. Half-Session Structure

NETWORK SERVICES AND THE NAU SERVICES LAYER

NAU services managers control network operation by exchanging network services RUs with one another, using SSCP-based sessions, i.e., SSCP-SSCP, SSCP-PU, and SSCP-LU sessions. Categories of network services RUs (a type of FMD RU) are:

- Configuration services: supported on SSCP-PU sessions in order to activate and deactivate links, to load and dump nodes, to perform dynamic reconfiguration, such as assigning network addresses to local addresses, and, in general, to control resources associated with the physical configuration.

- Session services: supported on SSCP-SSCP and SSCP-LU sessions in order to assist LUs in activating LU-LU sessions; this includes such activities as resolution to network addresses by the SSCPs of network names presented by an LU in its session initiation request, checking of end-user password and access authority, and selection and matching of session parameters.

- Maintenance and management services: supported on SSCP-PU and SSCP-LU sessions in order to perform testing and tracing, and to record statistics on network resources.

- Measurement services: a category of network services set aside for future definition; currently, all collection of measurement data is implementation defined, using LU-LU sessions.

- Network operator services: a category of network services set aside for future definition; currently, all network operator communications with the SSCP are implementation-defined.

For a given NAU, the services manager and the FMDSs (SNSs or SPSs) for its various half-sessions jointly form a NAU services layer (NAU.SVC), as shown in Figure 1-14. The NAU services manager performs a function or arranges for a function (e.g., a DLC function) as requested by a paired services manager. SNS provides routing between the half-session and the appropriate component of the NAU services manager, based on the network services category of a request or response. Depending on the particular network services category, state information relating to network services RU sequencing is maintained by SNS or by the NAU services manager. Chapter 6 discusses this in greater detail. SPS is described in SNA LU-LU Session Types.

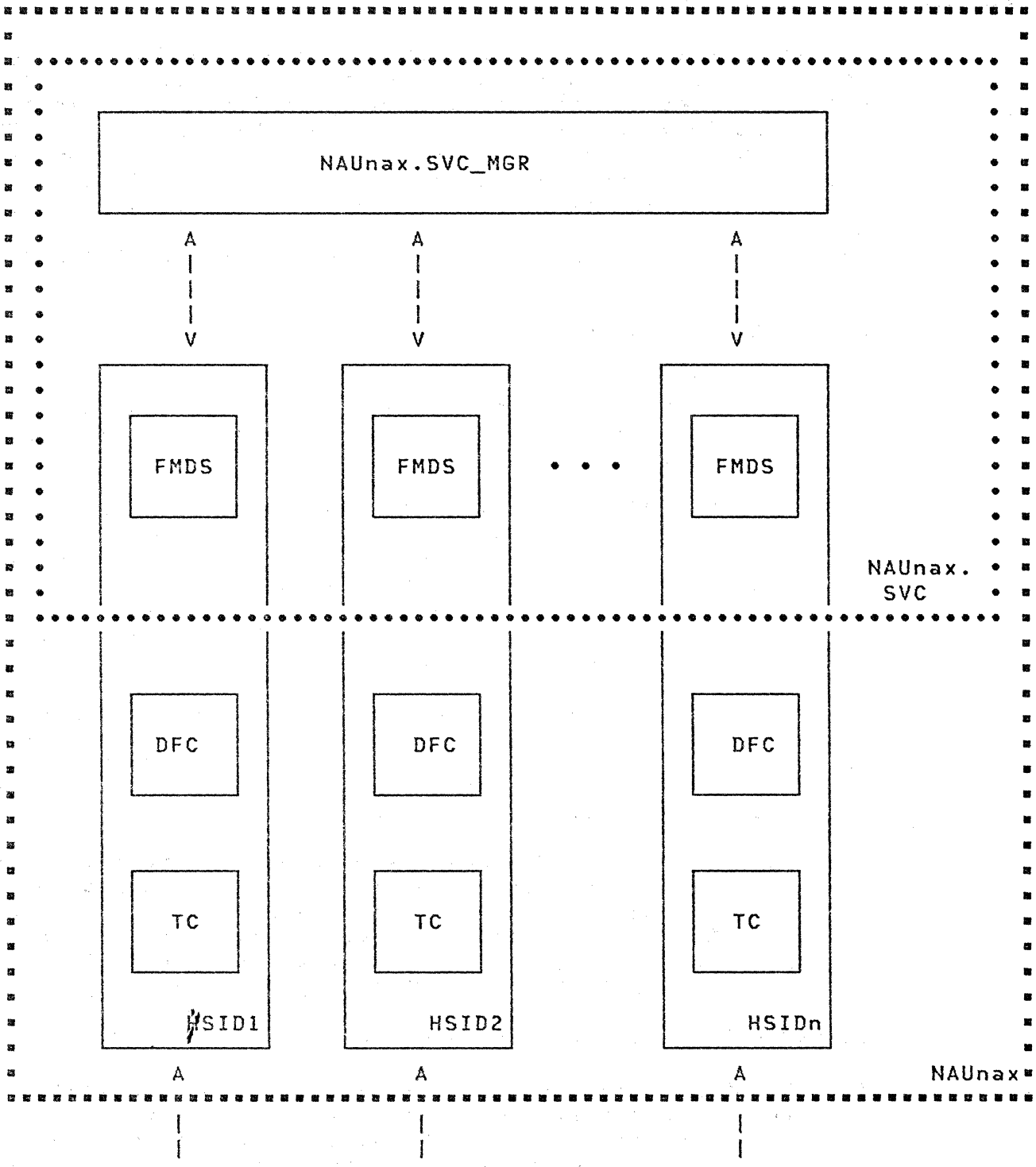


Figure 1-14. NAU Services within a NAU

NAU.SVC STRUCTURE

The structure of a NAU services component varies according to the type of NAU--SSCP, PU, or LU. Each provides a different grouping of services and functions. Figures 1-15, 1-16, and 1-17 show the structures of the SSCP.SVC, LU.SVC, and PU.SVC. An earlier section, "Half-Session Structure," dealt briefly with the FMDS components. Here, highlights of the NAU services manager structures are discussed; details on the various signaling paths (and more comprehensive overviews) are given in Chapters 6 (SSCP and LU) and 10 (PU).

SSCP.SVC_MGR has the following components:

- UPM_TRANSLATION_SVC, an implementation-defined component, routes and translates signals between the SSCP.SVC_MGR and the network operator controlling the SSCP (and its domain).

- SSCP.SVC_MGR.CS provides configuration services support for a domain in cooperation with peer components in the PUs via SSCP-PU sessions. It also sends and receives all session activation requests and responses involving the SSCP via the common session control manager (CSC_MGR) in the local PU. Chapter 7 discusses SSCP.SVC_MGR.CS in greater detail.

- SSCP.SVC_MGR.SS provides session services support for LU-LU sessions in cooperation with peer components in LUs within its domain and other SSCPs via SSCP-LU and SSCP-SSCP sessions. Chapter 8 discusses SSCP.SVC_MGR.SS in greater detail.

- SSCP.SVC_MGR.MN&MA provides management and maintenance services support in cooperation with peer components in PUs and one or more LUs within its domain via SSCP-PU and SSCP-LU sessions. (See the section, "Communication Network Management," later in this chapter.) Chapter 9 discusses SSCP.SVC_MGR.MN&MA in greater detail.

LU.SVC_MGR has the following components:

- LU.SVC_MGR.SS provides session services support for LU-LU sessions, initiating them in response to an end-user request. LU-LU session initiation involves the process beginning with the original LU-LU session initiation request from an end user (requesting a session initiation involving its own LU or two other LUs) and ending with the successful exchange of the LU-LU session activation request and response. The SSCP within each LU's domain cooperates with its LUs and the SSCPs of the others (up to three domains can be involved) via SSCP-LU and SSCP-SSCP sessions to effect

the LU-LU session initiation. Details of the process are given in Chapters 8 and 13. LU.SVC_MGR.SS also exchanges all session activation requests and responses involving its LU with the CSC_MGR manager in its PU. (It provides similar support for LU-LU session termination.) Chapter 8 discusses LU.SVC_MGR.SS in greater detail.

- LU.SVC_MGR.MN&MA provides management and maintenance services support in cooperation with peer components in PUs and the SSCP in its domain as discussed in the later section, "Communication Network Management." Chapter 9 discusses LU.SVC_MGR.MN&MA in greater detail.

- LU.SVC_MGR.PS provides presentation services support for LU-LU sessions as discussed in SNA LU-LU Session Types.

- LU.SVC_MGR.SYNC_PT provides sync point services for coordinating checkpoints of resources anchored in the LU, possibly in conjunction with those located at one or more other LUs connected by sessions to the local LU, and for recovering from failures by reverting to committed checkpoints. See the section, "Sync Points," for additional discussion.

PU.SVC_MGR is the focal point within a node for controlling local resources in response to control point requests and for initializing, and otherwise managing, layers and logical connections represented within the node. PU.SVC_MGR has the following components:

- PU.SVC_MGR.NS provides network services (configuration services, management and maintenance services) support in cooperation with like components in its SSCPs (see the section, "Shared Control") and LUs (see the section, "Communication Network Management"). It responds to PUCP or SSCP requests to activate and deactivate local resources such as links and adjacent link stations. It also converts network services requests received from a control point into appropriate (e.g., NC or DLC) signals to other PU.SVC_MGR components and receives status from them (causing it to send network services RUs to a control point). PU.SVC_MGR.NS receives ACTPUs and sends the responses to ACTPUs via the CSC_MGR manager. Chapter 11 discusses PU.SVC_MGR.NS in detail.

- PU.SVC_MGR.PC_ROUTE_MGR provides services using the PU-PU flows (see the section, "Sessions and PU-PU Flows") and participates (within subarea nodes) in the session-activation process to assign sessions to virtual routes. Subcomponents include a virtual route (VR) manager and an explicit route (ER) manager. The

VR and ER managers control virtual and explicit routes and are layer managers of VRC and TGC. Chapter 12 discusses PU.SVC_MGR.PC_ROUTE_MGR in detail.

- PU.SVC_MGR.CSC_MGR is the layer manager for all half-sessions in the node. It acts as a conduit for all session-activation and -deactivation requests and responses exchanged between other NAU services manager components and path control. It controls the half-session activation and deactivation processes by creating, initializing, and destroying half-session control blocks. It serves the boundary function in the same way (see the section, "Boundary Function Structure"). The CSC manager also provides session outage notification support (see the section, "Session Outage Notification"). Chapter 13 discusses PU.SVC_MGR.CSC_MGR in detail.

- PU.SVC_MGR.LINK_MGRs exist for each link attached to the node. A link manager controls activation, deactivation, and status reporting of an underlying DLC.(PRI|SEC) and link connection. Details of this component are not given in this book.

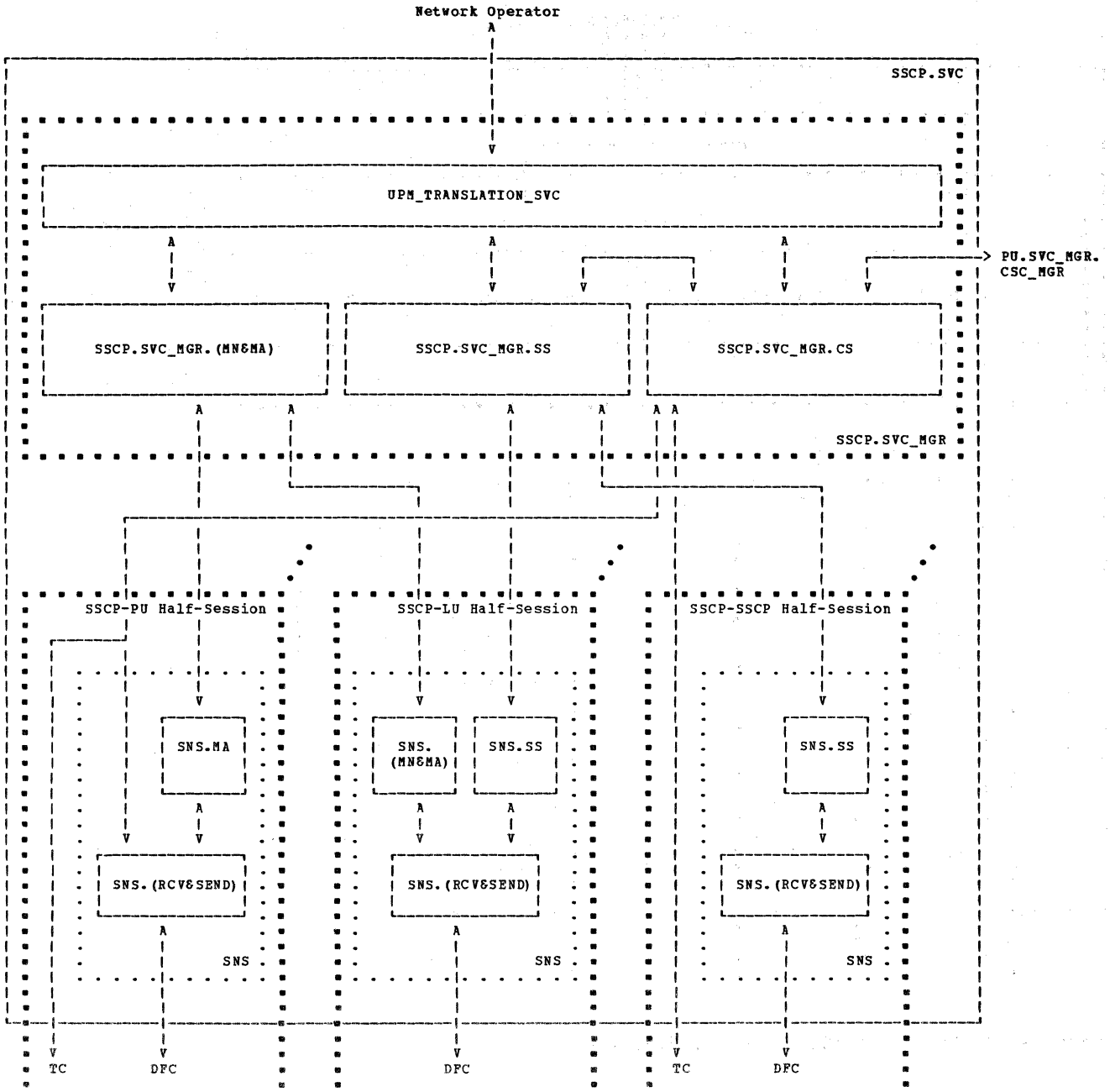


Figure 1-15. Structure of SSCP.SVC

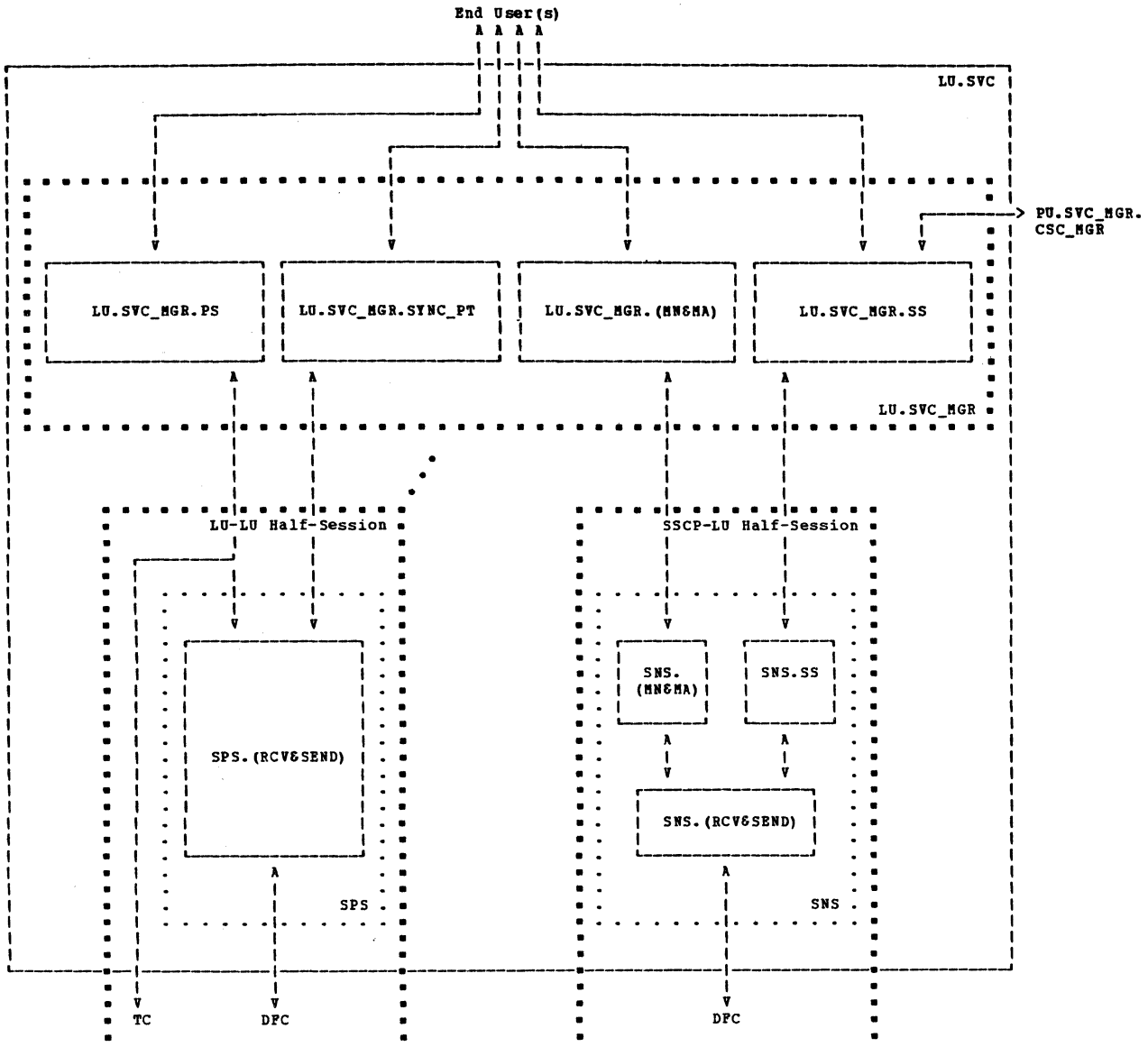


Figure 1-16. Structure of LU.SVC

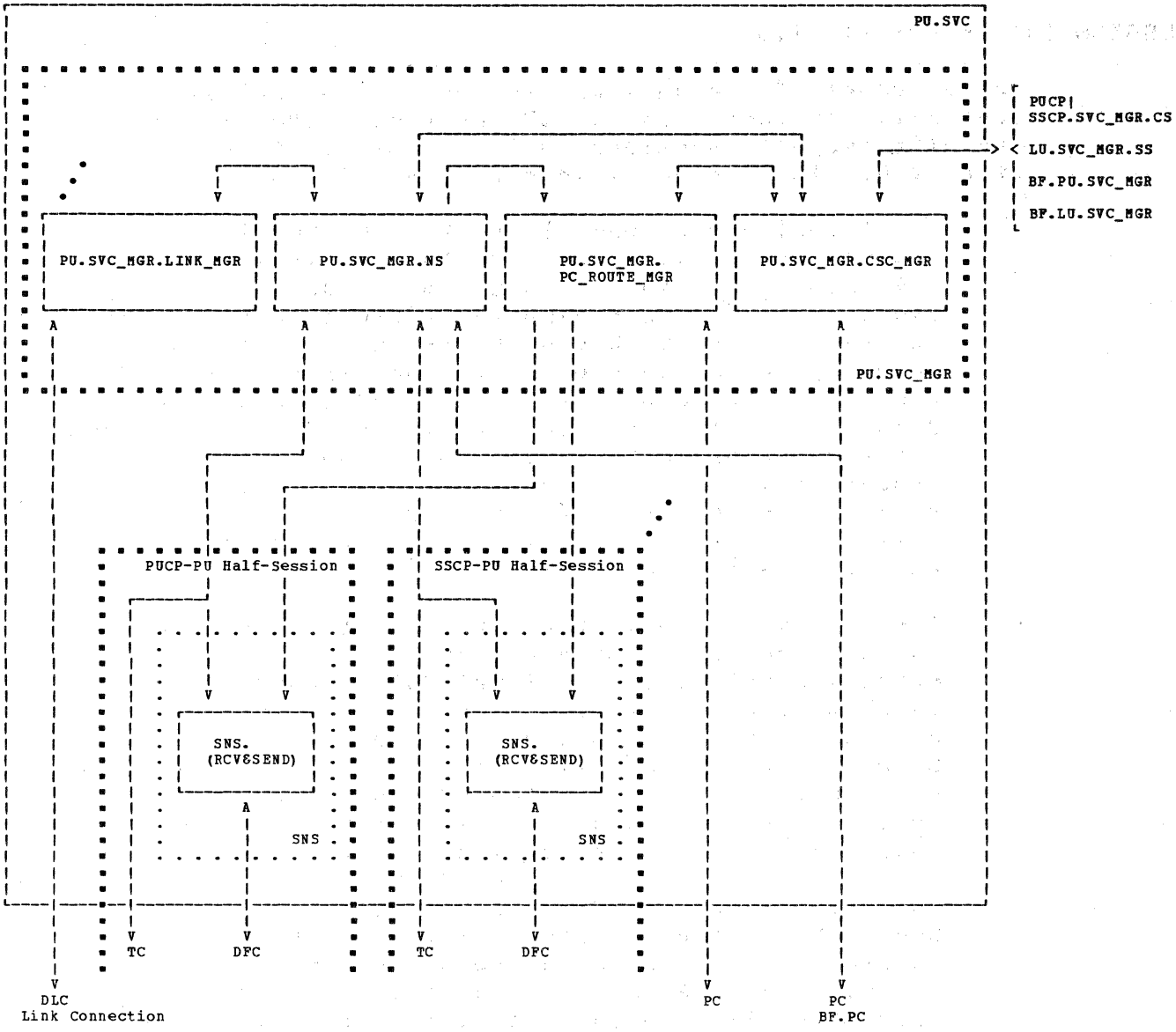


Figure 1-17. Structure of PU.SVC

SESSION OUTAGE NOTIFICATION

For a number of reasons, an active session between two NAUs can fail. In these cases, SNA provides means for notifying the affected half-sessions so that restart processing can be attempted. Two common reasons are as follows:

- A virtual route underlying active sessions has been disrupted as a result of the failure of the last remaining link in a transmission group used by the explicit route underlying the virtual route, or because the virtual route has been forcibly deactivated. The VR managers on both ends of the virtual route inform the local CSC managers, which generate and send session deactivation requests to the affected half-sessions in their subareas, notifying them of the failure. Restart processing, via a network- or terminal-operator reinitiation of the session, can involve assigning a different virtual route in order to bypass the failure and reactivate the sessions.

- A route extension in the path of active sessions fails. The NS manager in the PU on each side of the route extension notifies the local CSC manager, which sends LU-LU session deactivation requests to all affected LU half-sessions on its side of the failure. Within an SSCP, any affected half-sessions are reset as a result of receiving INOP from the PU in the subarea node to which the route extension is attached. (So that INOP reporting for route extension failure can succeed, all SSCP-based half-sessions in PUs and LUs within the same subarea use the same virtual route to a specific SSCP. When the virtual route fails, all sessions for the subarea fail, thereby allowing coordinated reset and reactivation. Enforcement of this constraint is by the SSCP when it resolves the class of service name to a virtual route identifier list.)

There are other cases, where LU-LU sessions have not failed, but hierarchically higher-order (SSCP-based) sessions have, or where active SSCP-SSCP or SSCP-PU sessions are overridden by more recent session activation requests received over a different virtual route. Details of these cases are given in Chapters 8, 11, and 13.

Various reset hierarchies among FSMs exist; that is, groups of related FSMs are reset when another FSM is reset. In general, reset hierarchies are determined by the control blocks in which FSMs are anchored (see Appendix A). Destroying a control block within the meta-implementation resets the FSMs anchored in that control block. Alternatively, explicit reset signals may be sent to individual FSMs to reset them, as shown in the various chapters of this book.

Chapter 13 defines reset hierarchies related to session outage notification; those related to lost-control-point processing and to INOP and deactivation processing are defined in Chapters 11 and 13, respectively.

The next section, "Sync Points," discusses checkpoint and recovery capabilities for LU-LU sessions.

SYNC POINTS

An LU may present to its end user(s) an environment in which various system resources, such as terminals, data bases, and queuing facilities, are all allocated and deallocated within the period that a session is active. Typically, allocation and deallocation are synchronous with the attaching of a transaction processing application program to the LU and its subsequent termination, or detaching from the LU.

This attaching and detaching of a transaction processing application program corresponds to the beginning and ending of a bracket. Where a session connects a terminal operator with a transaction processing subsystem (e.g., a data-base/data-communication application subsystem), a bracket is typically initiated by the terminal operator, who includes the identity (transaction code) of the desired transaction processing program within the first message unit of the bracket. Two transaction processing subsystems also can be connected by one or more sessions, in which case the bracket is initiated from either end.

While a transaction processing program is running, the LU monitors its access to system resources, such as data-base entries or the session itself. As changes are made, such as updating or deleting a data-base entry or sending data to a transaction processing program at the paired half-session, the resource involved is locked for exclusive use and a record is made of the change. When all processing related to a specific unit of work is completed, the change records are erased and the work is thereby committed.

Frequently, transaction processing involves multiple-step actions, wherein all steps must be completed as a unit because only the combined action has meaning. For example, correlated records in a distributed data base must all be updated in synchronism with each other. Should the multiple-step process fail midway through--say because of session failure--the change records could be used to undo the partially completed unit of work.

In SNA, sync point protocol machines are defined within LUs to coordinate the two ends of a session in committing to the completion of a distributed unit of work:

- A special request ("Commit"), encoded as a bit in the request header (RH), is sent by one end to request the other end to commit to completion of a unit of work. The end receiving the request can signal its agreement by positively responding to the Commit request, thereby establishing a new sync point. If a negative response is sent, both ends undo the unit of work and revert to the previous sync point.

- Another request ("Prepare"), encoded as FM header 10 (see SNA LU-LU Session Types) or as a value in the LUSTAT request, allows one end to request the other end to send the Commit request. The sender of the Prepare request need not keep protected resources locked if the session fails before the Commit request is received; it can simply revert to the previous sync point and release the resources. The sender of the Commit request awaiting a response cannot determine whether a session failure occurred before or after the other end committed to the new sync point; this can only be determined at reactivation of the failed session.

- The SET AND TEST SEQUENCE NUMBERS (STSN) request is provided for use at session reactivation--after BIND, but before data traffic is exchanged--in order to determine how a session failure affected the sync points at the two ends. STSN and its response cause the two ends to exchange sequence numbers corresponding to their most recent sync points, thereby synchronizing the two ends. Either they agree already, or one must revert to a previous sync point to reach agreement. Because either end can send Commit, this means zero, one, or two sync points may be in doubt when STSN is exchanged.

LUs provide this optional sync point service via the sync point manager (LU.SVC_MGR.SYNC_PT), shown in Figure 1-16.

All resources allocated to a transaction processing application program attached to an LU are characterized as protected or unprotected. At intervals, the program may signal the sync point manager to commit all the changes to the protected resources, in order to move forward to a new sync point. (The program may be coordinating common sync points for multiple sessions through the sync point manager.) Alternatively, the program may signal the sync point manager to revert to the previous sync point, thereby undoing any changes made to protected resources after that sync point.

Actual implementation of a protected resource can vary widely, depending upon the errors against which protection is desired. In the most stringent case, protection against application program, LU, node, and session failure is

provided; the protected resource might then be a disk file. For this level of protection, a list of states before and after changes to that resource since the last sync point is kept on a non-volatile storage medium (a sync point log). This list is used to restore the state of the protected resources when an abort occurs because of a session failure, an application program request, or an application program error.

One-phase commit, in which the requester sends Commit and the response indicates yes or no, is the basic protocol. Two-phase commit, in which the first sync point manager sends Prepare, requesting the second to send the Commit, is an implementation option. Two-phase commit allows some choice of which LU must hold locks on protected resources pending completion of the sync point--perhaps across a session failure.

Sessions that are protected resources can be resynchronized (using STSN as noted above) after a failed session has been restarted. Since the network addresses of a session can change (being dynamically assigned) during the interval between session failure and session restart, the information needed to support session resynchronization is saved by session name--see the BIND request in Chapter 13. Following resynchronization, restart of application programs that were running at the time of the session failure is an implementation option.

Chapters 2 and 4 and Appendix E discuss the use of the RH, STSN, and LUSTAT for sync points in more detail. Details of the sync point manager are given in SNA LU-LU Session Types.

SHARED CONTROL

Multiple SSCPs can share control of various network resources, either sequentially or concurrently. In addition, the PUCP in a node can share control of some node resources (e.g., the PU, links, adjacent link stations) sequentially or concurrently with multiple SSCPs. This allows local activation of node resources when no SSCP exists in the node. A PU can be shared by allowing more than one control point (CP)--an SSCP or PUCP--to send it the activation request, ACTPU; the PU is a member of each domain whose SSCP can share such control. Once an SSCP has control of the PU in this way, it can also share control of resources associated with the PU--in particular, LUs, which it activates via ACTLUs, and links, which it activates via ACTLINKs. Link stations on a link can be controlled via CONTACTs by each SSCP or PUCP that shares control of the link. SSCPs sharing control of a given PU may vary in their interest in, or awareness of, associated LUs and links; each SSCP may make use of separate resources.

Resources that can be shared have different limits on the number of CPs that can concurrently share them. This concurrency constraint is called the share limit for the resource. LUs have a share limit of 1--only one SSCP at a time can exercise control. (PUCPs do not control LUs.) A peripheral PU, with respect to its boundary function support, also has a share limit of 1--only one SSCP controls it at a time. (However, the PUCP within the peripheral node can share control of the peripheral PU with an SSCP.) For subarea nodes, the share limits of resources other than LUs can be greater than 1, subject to installation-dependent options, thereby allowing the PUCP in the node and multiple SSCPs to concurrently exercise control.

Shared control of network resources can be used for such purposes as:

- Backup of one SSCP by another to increase network availability
- Partitioning control of a network by use, rather than by physical location of resources (e.g., multiple SSCPs can share control of a given link and partition use of the nodes connected via the secondary link stations on that link)
- Time-of-day shifting of control of various network resources

NTWK.TC, NTWK.DFC, AND NTWK.SESS

The nesting of networks within NTWK.SNA can be extended as shown in Figure 1-18, where the intermediate levels between NTWK.PC and NTWK.SNA are introduced.

The TC elements, and the NTWK.PC interconnecting them, form NTWK.TC, which is the second innermost network of the networks nested in NTWK.SNA. Within each session, the TC element pair (one in each half-session), in conjunction with NTWK.PC, provides a connection for passing RUs between the DFC element pair. This connection has the following properties:

- No send-receive coupling exists between the flows in the two directions; this means that NTWK.TC supports concurrent send and receive operations at its protocol boundary with DFC.

- Normal-flow RUs, whether requests or responses, are delivered to a DFC element in the order of submission to NTWK.TC by the paired DFC element, except for responses that carry an RH indicator value specifying that they are to bypass TC queues--these responses can pass queued requests (and responses) delayed by pacing, thereby avoiding session deadlocks (see Chapters 2 and 4).

- The size of the normal-flow RUs and the rate at which normal-flow request RUs can enter NTWK.PC are limited by the TC elements in accordance with parameters chosen at session activation.

- Data traffic (normal-flow FMD and DFC RUs) over the connection can be enabled and disabled through data traffic protocols provided by the session control components within the paired TC elements.

In a later section of this chapter, a protocol machine called the "boundary function," which has a special role in NTWK.TC for supporting peripheral nodes, is discussed.

The DFC elements, and the NTWK.TC interconnecting them, form NTWK.DFC, which is the third innermost network of the networks nested in NTWK.SNA. Within each session, the DFC element pair, in conjunction with NTWK.TC, provides a connection for passing RUs between the FMDS element pair. This connection has the following properties:

- Send-receive coupling of the flows in the two directions is enforced in accordance with session activation parameters.

- Normal-flow RUs, whether requests or responses, are delivered to an FMDS element in the order of submission to NTWK.DFC by the paired FMDS element, except for those responses, mentioned above, that can bypass TC queues.

- Checking and synchronizing of RU groupings (architectural chains and brackets) is supported in accordance with session activation parameters.

- The other DFC functions, discussed in a previous section ("Half-session Structure"), are enforced.

The FMDS elements, and the NTWK.DFC interconnecting them, form NTWK.SESS. NTWK.SESS consists of all the half-sessions in NTWK.SNA, along with NTWK.PC, which interconnects them.

Within each session, the FMDS element pair, in conjunction with NTWK.DFC, provides a connection for passing message units between pairs of services managers. This connection

has all the properties of the connection provided by NTWK.DFC, in addition to properties introduced by FMDS functions:

- A message unit delivered to a services manager may have a format different from that submitted to NTWK.SESS by the paired services manager, in accordance with formatting or presentation services available for the session.

- For an SSCP-based session, the FMDS element pair checks and maintains the current states for controlling and synchronizing the flow of certain network services RUs, e.g., in the session services category. (For other network services categories, e.g., configuration services, control and synchronization of the RUs is performed in the NAU services manager layer.)

The SVC_MGRs and the NTWK.SESS interconnecting them, form NTWK.SNA.

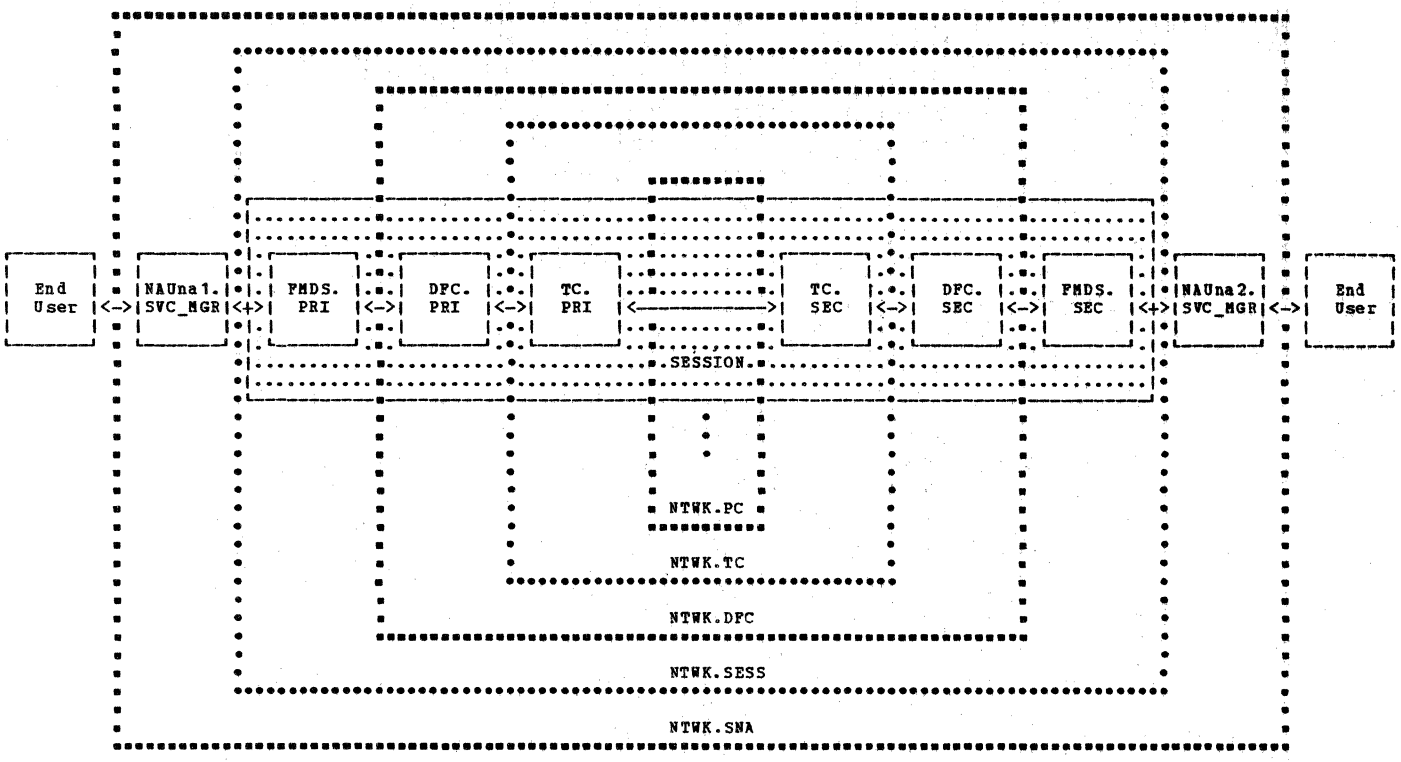


Figure 1-18. Network Layers

BOUNDARY FUNCTION STRUCTURE

Subarea nodes support the attachment of peripheral nodes through a protocol machine known as the boundary function (BF). Each BF consists of BF.PC, and a BF.NODE for each peripheral node attached to the subarea node (see Figure 1-19). Each BF.node establishes a reset hierarchy for a distinct peripheral node, and consists of a BF.PU representing the PU in the peripheral node and as many BF.LUs as there are LUs receiving boundary function support in the peripheral node. Each BF.(PU|LU) includes a BF.(PU|LU).SVC_MGR and as many HSID.BF.TCs as there are half-sessions within the peripheral PU or LU receiving boundary function support.

BF.PC in a subarea node provides path control function for FID2 (to/from a PU_T2 node) and FID3 (to/from a PU_T1 node) traffic into and out of the subarea node. This includes:

- Transformation between FID4 and FID2 or FID3 transmission headers (THs); this involves conversion between NAU network-address pairs and (link station network address, local address(es)) combinations; the relationship between TH format types (FID2, FID3, and FID4) and node types is shown in Figure 1-20.

- Link and adjacent link station routing

- Optional segmenting of message units destined for peripheral nodes into smaller units

BF.PC has a protocol boundary with PU.SVC_MGR.NS for the (PU-PU flow) loading (IPLing) function. This function bypasses the BF.NODE components completely. (See Chapter 11 for details.) BF.PC also has a protocol boundary with PU.SVC_MGR.CSC_MGR, which acts as the conduit (between BF.PC and BF.(PU|LU).SVC_MGR and between PC and BF.(PU|LU).SVC_MGR) for boundary-function related session-activation and -deactivation requests and responses and controls the activation and deactivation of HSID.BF.TCs by creating, initializing, and destroying boundary function session control blocks. PU.SVC_MGR.CSC_MGR also supports session outage notification for boundary-function supported half-sessions. (See the Section, "Session Outage Notification," and Chapter 13 for details.)

Details on BF.PC are provided in Chapter 3.

SIDi.SEC.BF.TC provides support functions for the secondary half-session, SIDi.SEC. This includes:

- Providing session-level sequence number support for type 1 peripheral nodes

- Providing session-level pacing support

An HSID.BF.TC has a protocol boundary with both PC and BF.PC. BF.PC (as part of the route extension) routes message units between HSID.BF.TC and the link to the path control element and the half-session in the peripheral node, while PC (using a virtual route) carries message units between HSID.BF.TC and the paired half-session in a subarea node (either the local one or some other one). Both PC and BF.PC route to HSID.BF.TC using the HSID network address pair.

Details of BF.TC are provided in Chapter 4.

Each BF.(PU|LU).SVC_MGRnai assists in providing BF support for sessions with (PU|LU)nai, which resides in an adjacent peripheral node. Each BF.(PU|LU).SVC_MGR provides cross-session reset coordination (i.e., for sessions associated with the same LU or node that receives boundary function support) and decides whether received session activation parameters are acceptable to the boundary function. It also determines whether the boundary function has sufficient resources to support the session.

The protocol boundary between BF.(PU|LU).SVC_MGR and PU.SVC_MGR.CSC_MGR is described in detail in Chapter 13.

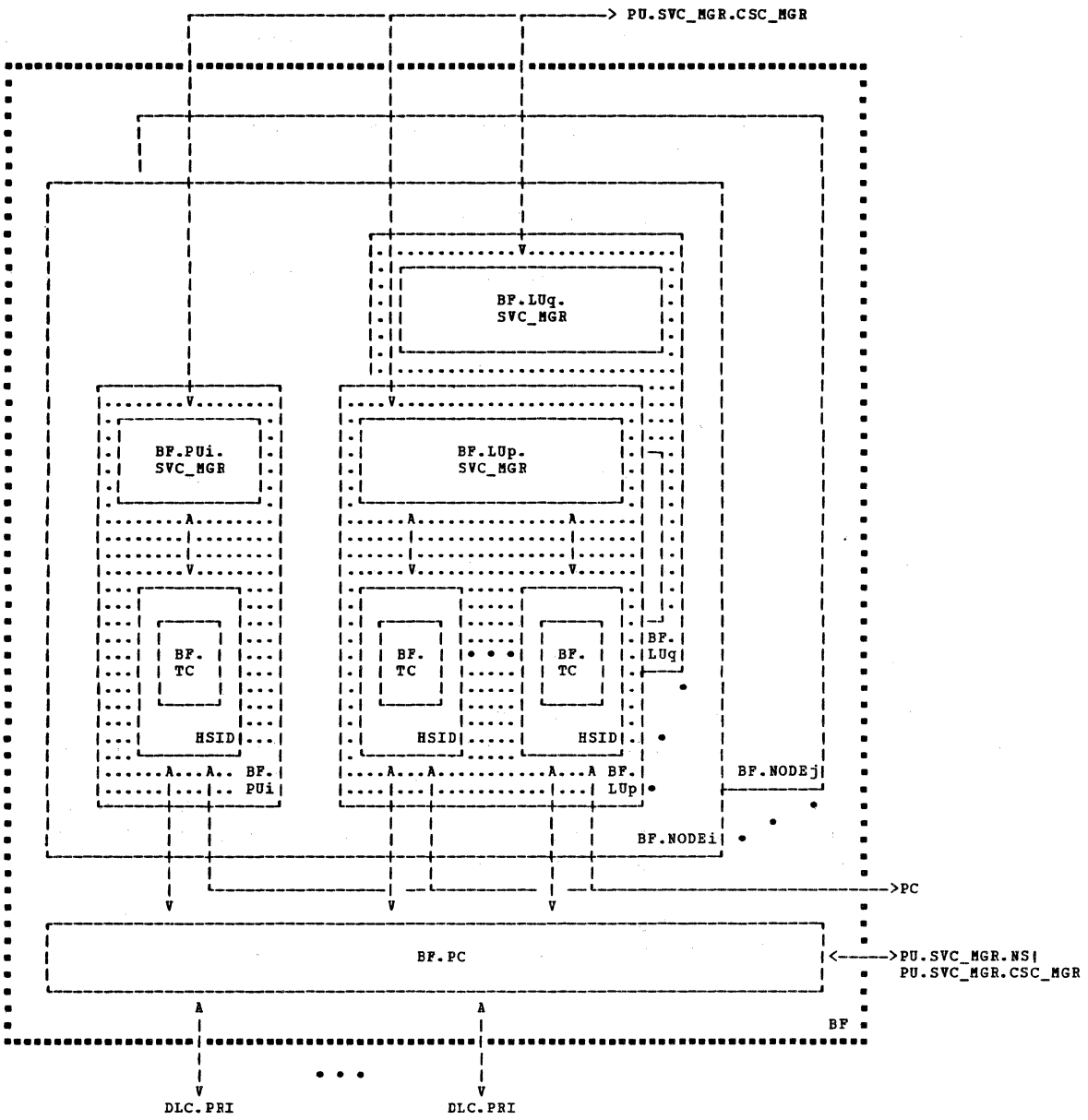


Figure 1-19. Boundary Function Structure

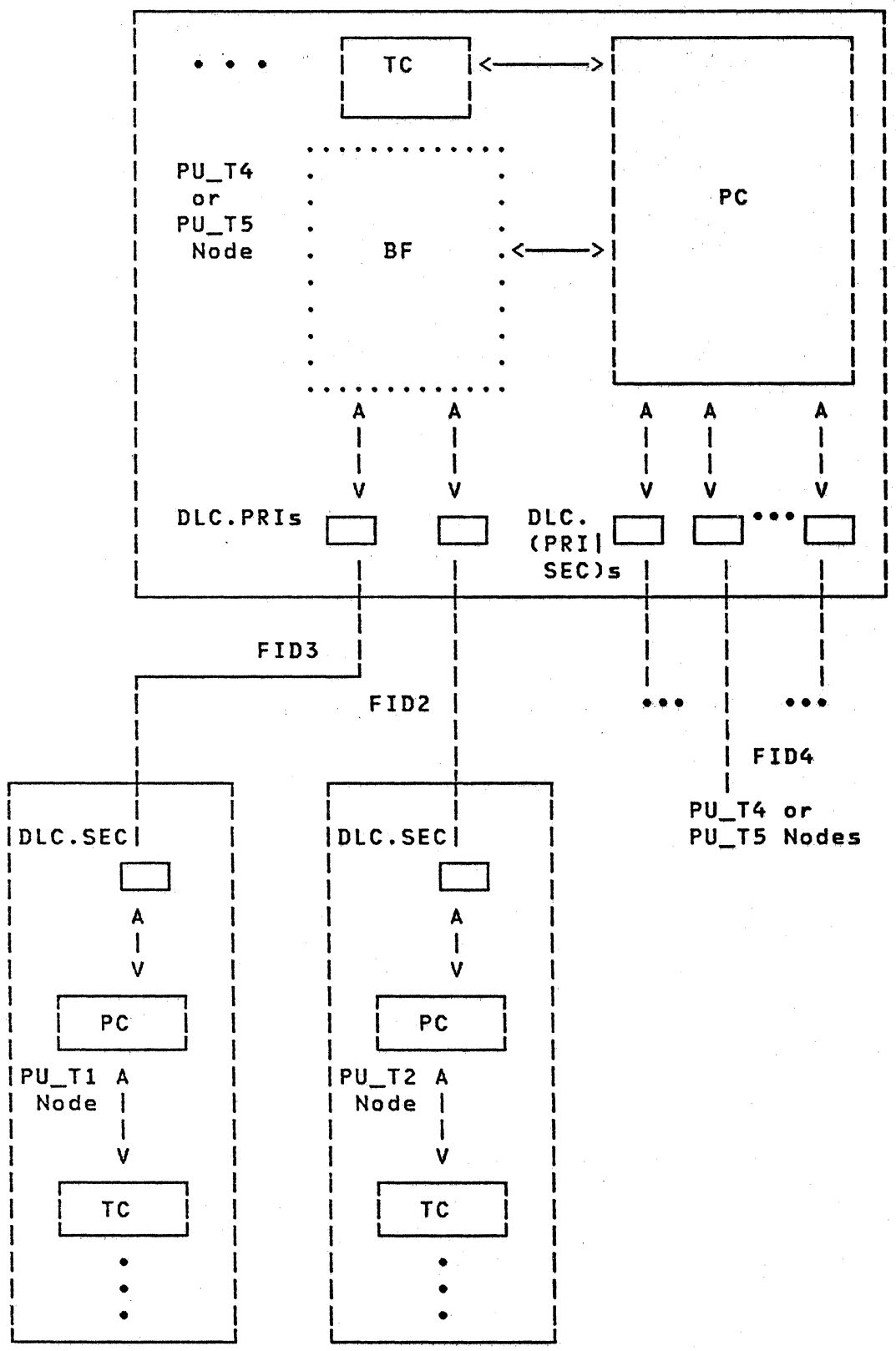


Figure 1-20. DLC/PC/TC/BF Relationships and FID Uses

COMMUNICATION NETWORK MANAGEMENT

Any user-application network requires a means for collecting information about the network for purposes of efficient operation and problem determination, and for display of such information to the installation management when problems exist in the network, or upon request at other times.

The SNA network provides such a means through a facility referred to as communication network management (CNM). A CNM services component may be located at each node of the network and has a protocol boundary with the PU. Typically, one CNM application component exists for each domain and has a protocol boundary with an LU (and thus is a specialized end user). The SSCP in a domain acts as an intermediate router between each CNM application for the domain and all the CNM services components that the application interacts with throughout a domain.

The CNM application and any particular CNM services component associated with its domain are connected together via an underlying LU-SSCP and SSCP-PU session. The CNM application sends to the SSCP (via FORWARD) an embedded request (currently, a maintenance-services category request only) and identifies a particular destination PU by name. The PU name is included in the FORWARD request following the embedded request. The SSCP provides a directory service of translating the name to a network address (just as it does for LU-LU session initiation) and passes the request received in the FORWARD on to the appropriate PU. Upon receipt of a CNM-related (maintenance-services) request from a PU, the SSCP uses the the request code to choose the appropriate LU to receive the request. The SSCP sends the request received from the PU to the LU by embedding the request within a DELIVER request and appending the network names associated with network addresses appearing in the embedded request.

A procedure-related identifier, or PRID, can be generated by the CNM application (or in some cases by the SSCP for its own routing) and included in the CNM header for correlation of requests exchanged via FORWARD and DELIVER. The SSCP uses the PRID to choose the LU to route to upon receipt of a CNM reply from a PU. A PU merely echoes a received PRID in its reply. (Chapter 9 gives more details of PRID usage.)

Because the installation management of a user-application network is not generally concerned with a network only in terms of domains, but is interested in centralized collection, processing, and display of network data, CNM-application to CNM-application communication is supported using cross-domain LU-LU sessions. This allows

data to be collected at one centralized LU (or more than one, if desired), processed, and from there displayed (perhaps using another LU-LU session) at any suitable location.

A number of SNA requests and replies are used to carry CNM-related information, such as for:

- Requesting and returning storage dumps
- Initiating and terminating tests and returning their status and results
- Reporting solicited or unsolicited data on SDLC tests, summary error data and statistics related to a PU or its attached resources, other PU/LU dependent data, engineering-change levels, link-connection subsystem (modem) data, and other maintenance statistics.

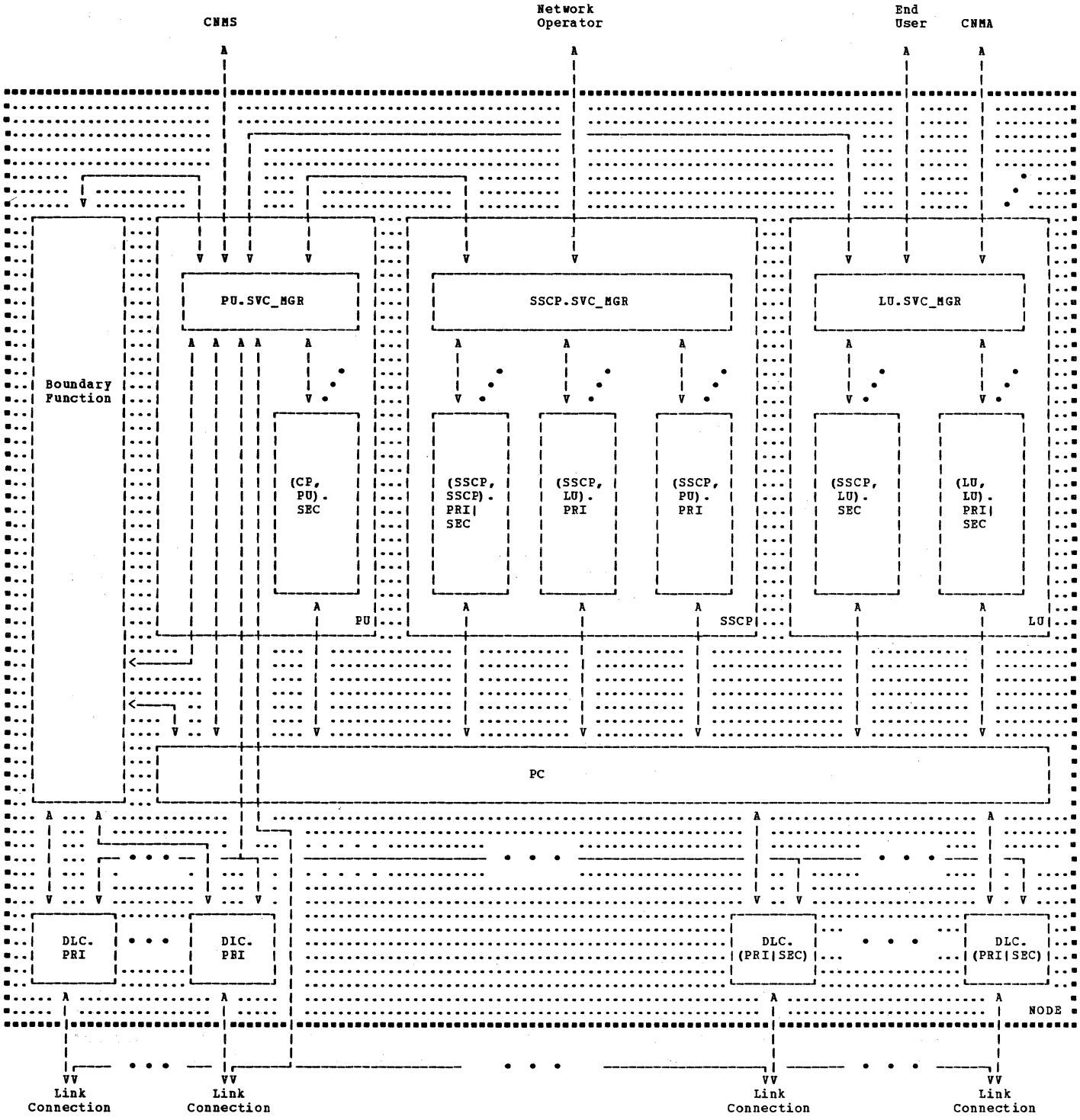
This book discusses the CNM-related routing capabilities in NAUs and the CNM RUs for maintenance and management services (see Chapter 9). Details of CNM services, CNM applications, CNM-application to CNM-application communication, other CNM system operations, and CNM presentation of data are defined in product publications.

STRUCTURAL OVERVIEW OF A NODE

The basic node structure shown in Figure 1-10 can be refined as shown in Figure 1-21. All components illustrated in the figure have been discussed, to varying degrees, in earlier sections of this chapter.

Not all node types have this general structure. For example, an SSCP exists only in a type 5 node; a PUCP (a functional subset of the SSCP) exists in all others. Only subarea nodes to which one or more peripheral nodes are (or can be) attached contain the boundary function.

Every node contains a PU, but LUs (one or more) are optional. PC exists in every node; the number of DLCs and attached link connections varies. (Peripheral nodes, for example, have only one.) The CNM components attach to nodes according to implementation and installation options.



Note: Only a type 5 node contains an SSCP; a type 1, 2, or 4 node contains a PUCP (not shown), which is a subset of an SSCP.

Figure 1-21. Structural Overview of a Node

PROFILES

Some of the session protocols (such as for request and response control modes, brackets, and pacing) are selectable at session activation. Specific combinations of these selectable protocol options are known as profiles. Those profiles that refer to TC options are called transmission services (TS) profiles; those profiles that refer to DFC and FMDS options are called function management (FM) profiles; those profiles that refer to SSCP options for cross-domain support are called CDRM profiles (see Appendix F for details). The TS and FM profiles to be used in any session are specified at the time of session activation via parameters in the appropriate session activation request and response (see ACTCDRM, ACTPU, ACTLU, BIND, and their responses in Appendix E); the CDRM profile is specified at SSCP-SSCP session activation, via a control vector parameter carried in ACTCDRM and +RSP(ACTCDRM).

LAYER AND ELEMENT STRUCTURE

Layering shapes the structure of NTWK.SNA. Layers consist of paired elements having the structure shown in Figure 1-22. Each element has a sending protocol machine (ELEMENT.SEND) from which it sends message units through the inner layers of the network to a receiving protocol machine (ELEMENT.RCV) in the matching element. The sending and receiving protocol machines within an element may be locally coupled.

Most element protocol machines, although differing in specifics, have the generic internal structure shown in Figure 1-23. Each basic element handles two principal flows: one toward the center of the network from the outer layers, the other toward the outer layers from the center of the network. A layer consists of two complementary submachines: ELEMENT.SEND and ELEMENT.RCV. ELEMENT.SEND handles the flow from the outer layers, and ELEMENT.RCV handles the flow toward the outer layers.

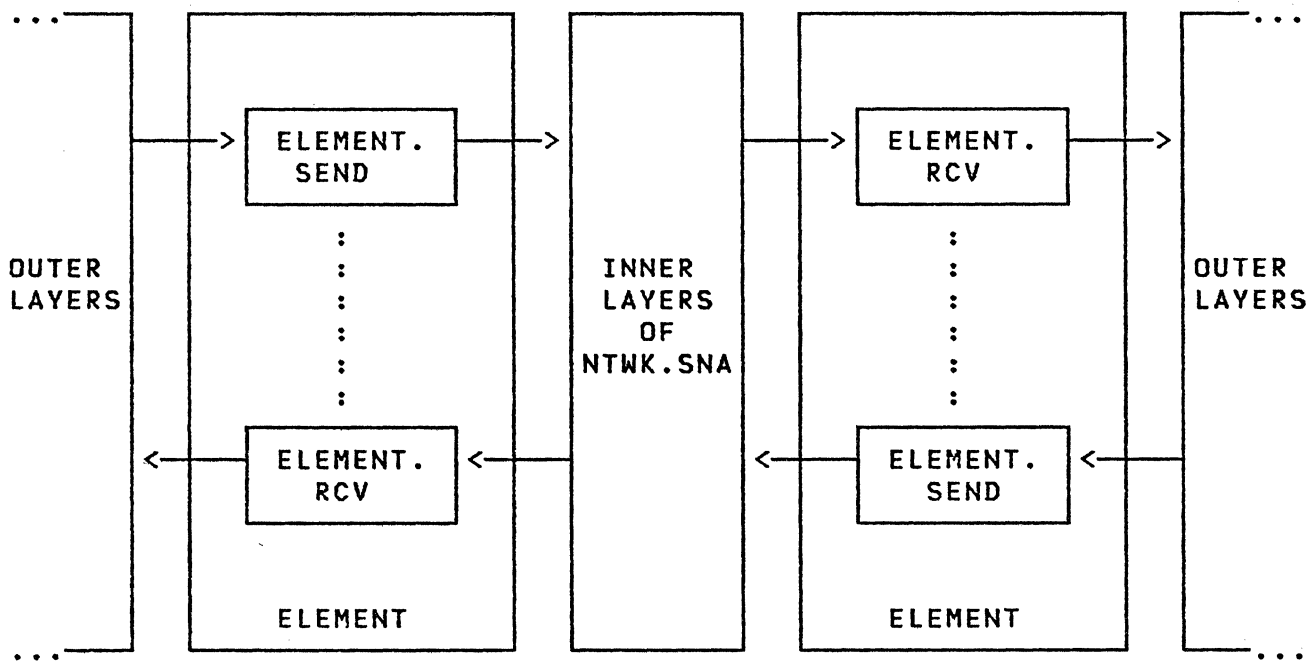


Figure 1-22. Pairing of Elements

ELEMENT.x (x=SEND|RCV) consists of:

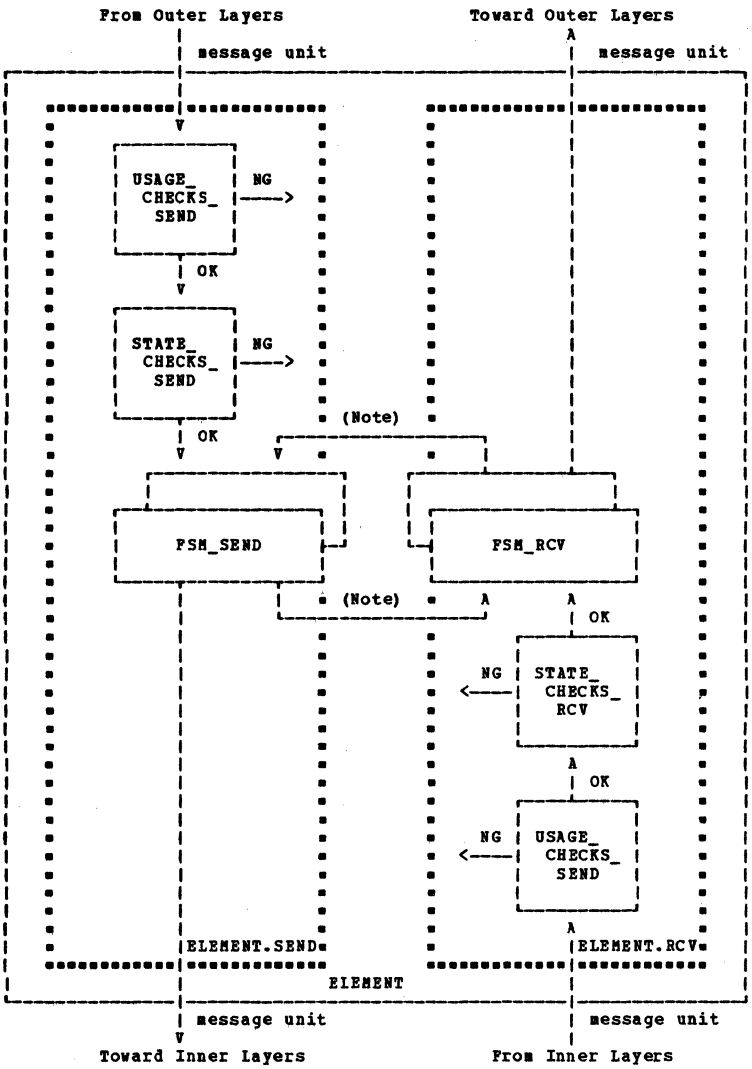
- USAGE_CHECKS_x: Checks message units for valid field usage, parameter values, and other state-independent format checks. If an error is detected, an error message unit may be generated and returned to the originator or sent to the destination. (Alternatively, message-unit fields and parameters may be set to their correct values within ELEMENT.SEND.)

- STATE_CHECKS_x: Checks the validity of message units relative to the current state of the element. If a state-dependent error is detected, an error indication may be generated and returned to the originator or sent to the destination; if no error is detected, the message unit is sent forward.

- FSM_x: Undergoes a state transformation in response to the input message unit being processed and produces an output message unit (not necessarily identical to the input message unit), which is sent forward.

To avoid undergoing state transformations in one layer and then detecting a send-check violation in a lower layer, the send checks for all layers can be made by the top layer. This practice is followed by the meta-implementation described in this book. It avoids rejections by lower layers that would require complex "backing out" of state transformations already made in higher layers.

The FSMs in ELEMENT.SEND and ELEMENT.RCV may be coupled; i.e., they may exchange signals. The strongest type of coupling occurs when both ELEMENT.SEND and ELEMENT.RCV share a single FSM (or set of FSMs).



Note: The FSMs in ELEMENT.SEND and ELEMENT.RCV may be coupled; i.e., they may exchange signals. The strongest type of coupling occurs when both ELEMENT.SEND and ELEMENT.RCV share a single FSM (or set of FSMs).

Figure 1-23. Basic Element Structure

Typical layer elements in NTWK.SNA occur in paired versions, as denoted in Figure 1-22. Message units are passed between element pairs in the following manner.

A message unit entering ELEMENT.SEND from its outer layer is:

1. Partially checked by USAGE_CHECKS_SEND, and, if valid, is

2. Forwarded to STATE_CHECKS_SEND, where it is checked for validity relative to the current send state; if valid, the message unit is

3. Routed to an FSM_SEND, where a state transformation occurs and the message unit (which may be transformed by the FSM) is passed to the inner layers of NTWK.SNA for transmission to ELEMENT.RCV.

The message unit arriving at ELEMENT.RCV from the inner layer is:

4. Checked by USAGE_CHECKS_RCV, and, if valid, is

5. Forwarded to STATE_CHECKS_RCV, where it is checked for validity relative to the current receive state; if valid, the message unit is

6. Routed to an FSM_RCV, where a state transformation occurs and the message unit (which is retransformed, if needed, by the FSM) is passed to its outer layer.

Each of the major composite protocol machines of a half-session, i.e., TC, DFC, and FMDS, involve specific interconnections of several basic elements. Details of these interconnections and of the constituent basic elements are presented in Chapters 4, 5, and 6.

OTHER DEFINITIONS AND NOTATIONAL CONVENTIONS

This section describes some notational conventions widely used in both the figures and the text. For complete details of the FAPL and FSM notation, see Appendix N.

As mentioned in the opening section of this chapter, each protocol machine in the book has a unique name consisting of a sequence of qualifiers. If the protocol machine belongs to a half-session, its name takes the form SID.(PRI|SEC).specific name for LU-LU, SSCP-LU, and SSCP-PU half-sessions, and (SSCP,SSCP').(SSCP|SSCP').specific name for SSCP-SSCP half-sessions.

(SID.PRI.specific name_SEND, SID.SEC.specific name_RCV) and (SID.SEC.specific name_SEND, SID.PRI.specific name_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.

The colon (:) is used (within the main text, but not in FAPL) in expressions of the form FSMx:STATEy, which denotes the statement "FSMx is in STATEy" (where FSMx is an FSM name and STATEy the name of a state in FSMx).

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

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

- Solid arrows indicate data flow using SEND or INSERT logic (see Appendix N).
- Dotted arrows indicate calling relationships.
- Dotted lines indicate data structure access.

A BIU, and hence the resulting PIU (or multiple PIUs, if segmenting is performed), is either a request or a response, depending on the RH coding; these are denoted respectively by RQ and RSP.

RQ(QUAL) denotes an RQ having the property described by QUAL; for example, RQ(CLEAR) denotes a request PIU whose RU is coded "clear," and RQ(Begin Chain) denotes a request PIU whose RH is coded "Begin Chain." A similar convention applies to responses. RSP(CLEAR) denotes a response PIU whose RU is coded "clear." The notation RSP(RQ(QUAL)) has a special meaning: it denotes a response to a request that had the property QUAL. For example, RSP(RQ(BB,EB)) is a response to a request that carried the BB and EB values. Whenever it is not ambiguous, RQ(QUAL) is denoted by the short form, QUAL; e.g., CLEAR implies RQ(CLEAR). No short forms are used for responses.

Abbreviations are used to shorten the length of FSM names, state names, and protocol inputs and outputs. The abbreviations are listed at the back of the book on foldout pages (Appendix T) for easy reference.

This page intentionally left blank

CHAPTER 2. MESSAGE UNITS AND HEADER FORMATS

This chapter describes the formats of the various SNA message units and headers. Additional usage details of the various header fields appear in subsequent chapters. The formats are also shown in figures in Appendix D.

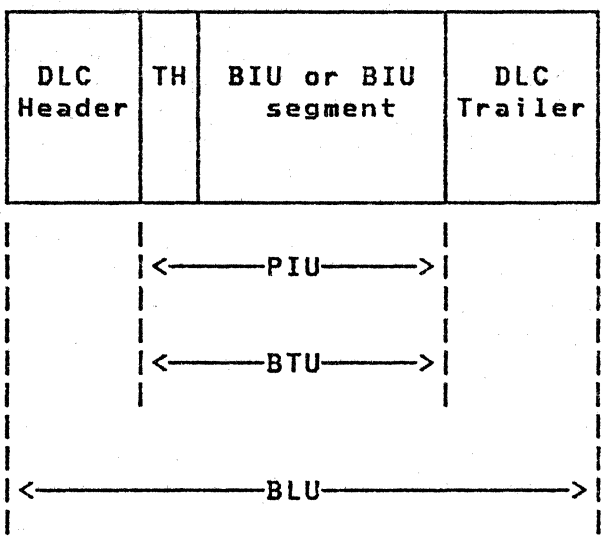
GENERAL MESSAGE UNITS

BASIC LINK UNIT

The basic link unit (BLU) is the basic unit of transmission at the data link and station level; it consists of a DLC header, followed by a basic transmission unit (BTU), followed by a DLC trailer (Figure 2-1). The DLC header and trailer carry DLC control information, and the BTU is the information field transmitted by the DLC element for the DLC users. For example, in synchronous data link control (SDLC), the BLU is one frame:

BLU = Frame = F,A,C [,BTU],FCS,F

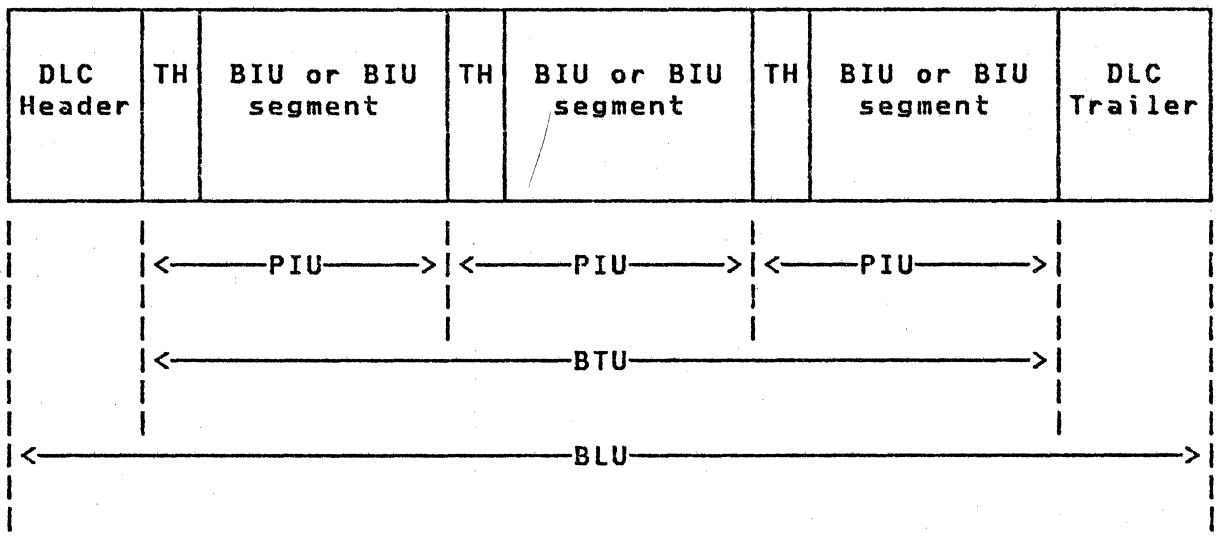
where F = Flag
A = Address
C = Control
BTU = Basic Transmission Unit
FCS = Frame Check Sequence



LEGEND:

DLC	Data Link Control
BIU	Basic Information Unit
BLU	Basic Link Unit
BTU	Basic Transmission Unit
PIU	Path Information Unit
TH	Transmission Header

A. Single PIU



B. Blocked PIUs

Figure 2-1. BIU/PIU/BTU/BLU Relationships

BASIC TRANSMISSION UNIT

The basic transmission unit (BTU) is the fundamental unit passed between path control and data link control. As shown in Figure 2-1, it can consist of one or, if blocking is used, multiple path information units (PIUs).

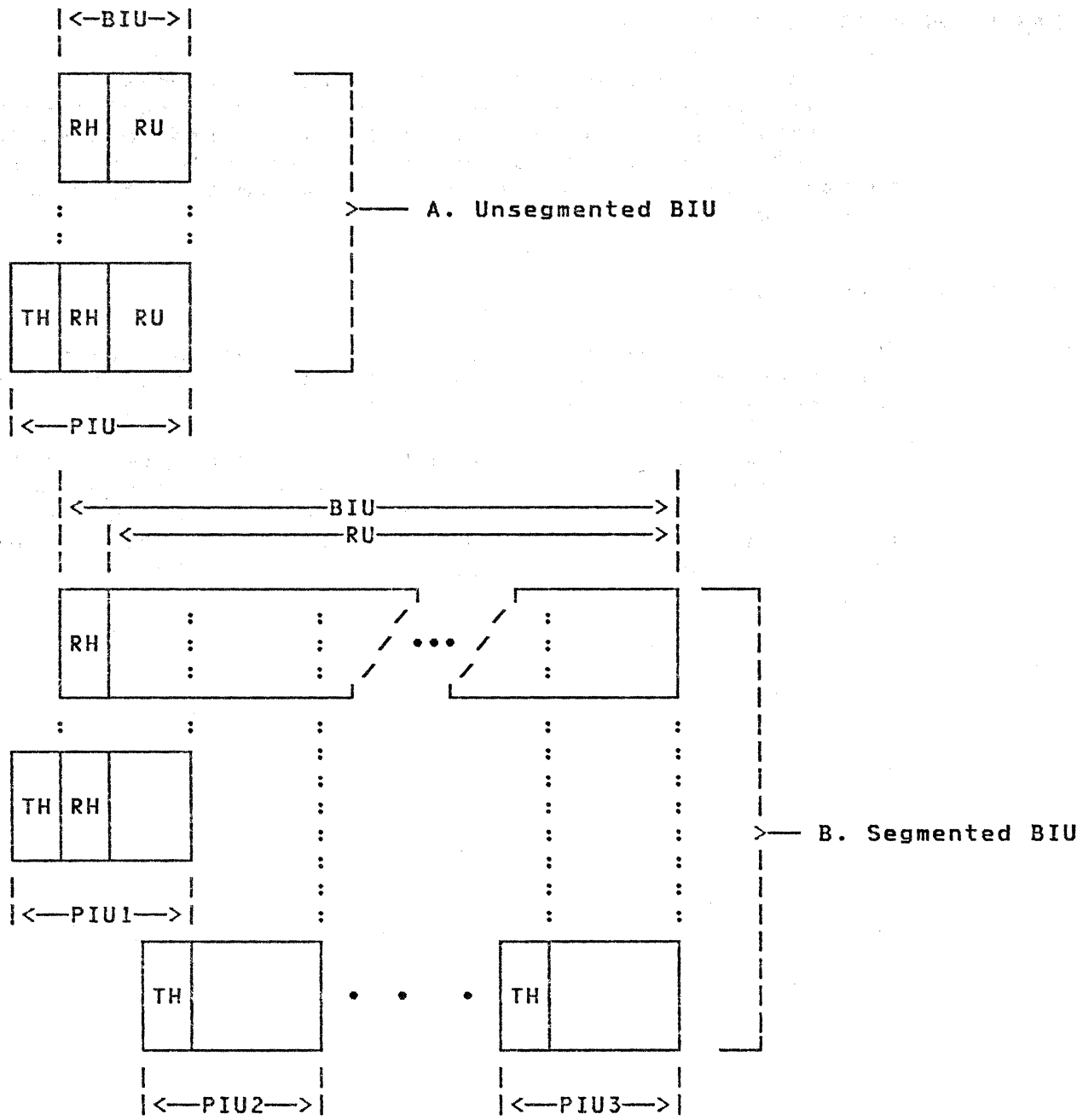
Maximum BTU sizes may be established to accommodate physical buffer-size limitations, buffer utilization considerations at the sending or receiving link station, or the transmission characteristics of the link connection.

PATH INFORMATION UNIT

A path information unit (PIU) consists of a transmission header (TH) alone, or a TH followed by a basic information unit (BIU) or BIU segment. Figure 2-2 shows the BIU/PIU relationship when segmenting is not used (A), and when segmenting is used (B). When segmenting is used and the size of a BIU to be transmitted is such that the resulting PIU (TH + BIU) would be larger than the maximum PIU-size permitted on a virtual route or by an adjacent link station, path control divides the BIU into multiple segments and sends them as multiple PIUs; thus, no PIU (TH + BIU segment) exceeds the maximum PIU-size permitted. See Chapter 3 for details. Note that when a BIU is segmented, only the first segment contains the request header (RH). To avoid truncating the sense data field of certain RUs (to be defined later) when segmenting a BIU, the first BIU segment must be no less than 10 bytes.

When a BIU intended for transmission by a DLC is larger than the message size permitted by the DLC, path control divides the BIU into segments. The maximum size of a segment is the smaller of:

- The maximum buffer size of the sending or receiving link station
- The maximum message size permitted by the characteristics of the link connection



LEGEND:

RH	Request/response Header
RU	Request/response Unit
BIU	Basic Information Unit
PIU	Path Information Unit
TH	Transmission Header

Figure 2-2. PIU/Segmenting Relationships

BASIC INFORMATION UNIT

A basic information unit (BIU) consists of a request/response header (RH) followed by a request/response unit (RU); it is the fundamental unit passed between origin and destination transmission control elements. BIUs are the fundamental units carried by NTWK.PC on the flows between paired half-sessions.

REQUEST/RESPONSE UNIT

The request/response unit (RU) contains user data, acknowledgment of user data, commands for the control of the network, or responses to commands.

The definition of RU formats and RU-related protocols constitutes a major portion of the remainder of the book. Chapters 4 onward, plus Appendix E, provide detailed information on RUs for network control, session control, data flow control, and network services FMD RUs.

EXRS REPLACING REQUESTS

An EXR is generated by a protocol machine when it detects an error in a request that is to be processed by additional protocol machines before being turned into a response. For instance, an EXR is sent by a CPMGR when it detects a sequence number error on a request. The request RU is replaced with sense data (see Appendix G), followed by up to three bytes of the original RU (as described for negative responses in Appendix E under "Response Units--Negative"); also, the Sense Data Included bit in the RH is turned on. EXR is the only request containing sense data; it is identified as an EXR by the value 1 in the SDI bit in a request header. If the request in error is already an EXR, the four bytes of sense data information may be overwritten with the new value, i.e., in the case of a path error. In general, EXRs are not overwritten because the first error detected is usually the most important and is reported in the negative response. EXR results in a negative response to the original request, if allowed; the negative response carries the same sense data as the final EXR.

The boundary function generates EXRs for sequence number errors detected on requests destined for half-sessions in a PU_T1 node. The EXR is also used within a half-session or between half-sessions as a signal generated by request-processing protocol machines and sent onward to indicate that the error denoted by the sense data has been detected for the request.

If an EXR cannot be delivered, the same sense data is returned as if the original request could not be delivered (thereby overriding any different error indicated in the undeliverable EXR).

TH: The Sequence Number field in an EXR is the same as in the request it replaces. It is checked, and the CPMGR sequence number receive count is updated, as for regular (non-EXR) requests. The TH Data Count field is altered to properly record the new BIU size. The Mapping field is set to BBIU, EBIU). All other fields are left as received.

An EXR replaces a complete BIU and is not used to replace one segment of a segmented BIU.

RH: The RH is the same as that of the original request, except that the Sense Data Included indicator is turned on.

RU: Bytes 0-3 contain sense data, in the same format as returned in the negative response. The sense data is followed by the original RU, truncated to no more than three bytes, as described in Appendix E for negative responses.

EXRS REPLACING TOO-LONG PIUS

A special use of EXR applies to both requests (whether segmented or not) and responses: if the length of a PIU received by transmission group control (TGC) from an upper layer exceeds the maximum BTU length allowed on the transmission group, TGC converts the PIU to an EXR having the following characteristics.

TH: Like EXRs replacing requests, EXRs replacing too-long PIUs leave all fields in the TH unchanged except for the MPF--set to (BBIU, EBIU)-- and the DCF--set to 7 here.

RH: The RH is set to X'07B000', no matter what the replaced RH (if any) was.

RU: Bytes 0-3 contain the sense data, X'800A'; no other bytes are included.

HEADER FORMATS

TRANSMISSION HEADER

A transmission header (TH) is the leading, or only, field of every PIU. The first half-byte of any TH is the Format Identifier (FID) field. Six different TH formats, or FID types, are defined: FID0, FID1, FID2, FID3, FID4, and FIDF; they correspond to hexadecimal values 0-4, and F, respectively, in the FID field. All other FID values are reserved.[1]

The remaining fields of the TH vary by FID type, and are described below.

[1] Throughout this book, *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.

FID0 and FID1

These formats are used between adjacent subarea nodes when either or both nodes do not support ER and VR protocols.

FID0 is used for non-SNA device traffic, and FID1 is used for SNA traffic. Except for the FID field value, the TH fields for FID0 and FID1 are identical.

Nodes that support ER and VR protocols provide conversion between FID4 THs and FID0 and FID1 THs. FID4 THs to be sent to nodes not supporting ER and VR protocols are converted to either FID0 or FID1 THs, as determined by the FID4 TH SNA indicator. FID0 and FID1 THs received from nodes not supporting ER and VR protocols are converted to FID4 THs, with the FID4 TH SNA indicator set to -SNA or SNA, respectively.

Except for the translation mentioned above, no FID0 or FID1 protocols are defined in this book.

The FID0|1 TH appears as:

Byte

0	FID0|1--Format Identification MPF--Mapping Field Reserved Bit EFI--Expedited Flow Ind.	Reserved Byte
2	DAF--Destination Address Field	
4	OAF--Origin Address Field	
6	SNF--Sequence Number Field	
8	DCF--Data Count Field	

Byte 0

FID0|1--Format Identification: 0000 for FID0, 0001 for FID1

MPF

The MPF consists of bit 4, the Begin-BIU (BBIU) bit, and bit 5, the End-BIU (EBIU) bit. It specifies whether the information field associated with the TH is a complete or partial BIU, and, if a partial BIU, whether it is the first, a middle, or the last segment.

10 first segment of a BIU = (BBIU, ¬EBIU)
00 middle segment of a BIU = (¬BBIU, ¬EBIU)
01 last segment of a BIU = (¬BBIU, EBIU)
11 whole BIU = (BBIU, EBIU)

Bit 6 of the TH is reserved.

EFI

The EFI is bit 7. It has the following meaning:

0 normal flow
1 expedited flow

The EFI designates whether the PIU belongs to the normal or expedited flow. Normal-flow PIUs are kept in order on a session basis by NTWK.PC; so are expedited-flow PIUs. Expedited flow PIUs can pass normal-flow PIUs flowing in the same direction at queuing points in TC within half-sessions and boundary function half-sessions.

Byte 1

All bits reserved.

Bytes 2 and 3

DAF--Destination Address Field, a two-byte network address denoting the BIU's destination network addressable unit (NAU). The DAF provides the principal routing information needed by NTWK.PC. In a network address the subarea address 0 is reserved; the element address 0 always denotes the PU_T4|5 generating the associated subarea.

Bytes 4 and 5

OAF--Origin Address Field, a two-byte network address denoting the originating NAU. The OAF allows multiple active half-sessions per NAU by distinguishing the origins of all PIUs received by the NAU.

Bytes 6 and 7

SNF--Sequence Number Field, a numerical identifier for the associated BIU. When segmenting, path control puts the same SNF value in each segment derived from the same BIU. For additional details on the use of this field, see Chapter 4, "The Sequencing of Requests and Responses."

Bytes 8 and 9

DCF--Data Count Field, a binary count of the number of bytes in the BIU or BIU segment associated with the transmission header; the count does not include any of the bytes in the transmission header. The DCFs are required in PIUs that are to be blocked, as they convey the PIU length information necessary for proper deblocking.

FID2

FID2 is the format used between a PU_T4 node and an adjacent PU_T2 node or between a PU_T5 node and an adjacent PU_T2 node.

The FID2 TH appears as:

Byte

0	FID2--Format Identification MPF--Mapping Field Reserved Bit EFI--Expedited Flow Ind.	Reserved Byte
2	DAF'--Destination Address	OAF'--Origin Address
4	SNF--Sequence Number Field	

Note: FID2 PIUs cannot be blocked because there is no DCF in the TH format for deblocking.

Byte 0

FID2--Format Identification: 0010

MPF and EFI, described earlier.

Byte 1

All bits reserved.

Byte 2

DAF'--Destination Address Field, a one-byte local address of the destination NAU. Within a specific PU_T2 node, the DAF' identifies the BIU's destination; within a BF.PC, the (LINK, STA, DAF') combination identifies the destination.

Byte 3

OAF'--Origin Address Field, a one-byte local address of the originating NAU. Within a specific PU_T2 node, the OAF' identifies the BIU's origin; within a BF.PC the (LINK, STA, OAF') combination identifies the origin. The BF adjacent to each PU_T2 node translates between each (LINK, STA, DAF', OAF') combination and the equivalent (DAF, OAF) network address pair.

Bytes 4 and 5

SNF--See FID1 description.

Note: The PU_T2 is always assigned the local address value of 0. Therefore, BIUs to the physical unit always have the associated DAF' = 0; BIUs from the physical unit always have the associated OAF' = 0. The OAF' is also 0 for BIUs from the SSCP, and DAF' is 0 for BIUs to the SSCP. A PU_T4|5 adjacent to the PU_T2 node has the local address X'FF'.

FID3

FID3 is the format used between a PU_T4 node and an adjacent PU_T1 node or between a PU_T5 node and an adjacent PU_T1 node.

The FID3 TH appears as:

Byte

0	FID3--Format Identification MPF--Mapping Field Reserved Bit EFI--Expedited Flow Ind.	LSID--Local Session ID

Note: FID3 PIUs cannot be blocked because there is no DCF in the TH format for deblocking.

Byte 0

FID3--Format Identification: 0011

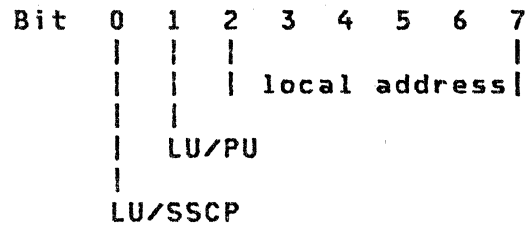
MPF and EFI, described earlier.

Byte 1

LSID--Local Session Identification

In FID3, the DAF and OAF are replaced by a single byte, the LSID, which provides a limited DAF/OAF capability.

The bit configuration of the LSID is:



The LSID consists of three parts: an LU/SSCP indicator (bit 0), an LU/PU indicator (bit 1), and a local address (bits 2-7).

Each PU_T1 node can support up to 64 secondary LUs; each LU is known, local to its PU_T1, by its six-bit local address. The PU_T1 can have an active session only with an SSCP, and each LU can have active sessions only with an SSCP and one other LU. The LSID bit settings for sessions supported by FID3 flows are:

Bit	0	1	2	3	4	5	6	7
Session Type	LU/SSCP	LU/PU	<----- Local Address ---->					
SSCP-PU	0	0	0	0	0	0	0	0
SSCP-LU	0	1	X	X	X	X	X	X
LU-LU	1	1	X	X	X	X	X	X
Reserved	1	0	X	X	X	X	X	X

The BF adjacent to each PU_T1 translates between the (LINK, STA, LSID) combination and the equivalent (DAF, OAF) network address pair. The (LINK, STA, LSID) combination implicitly determines the network address of the secondary LU. The relationship between the (LINK, STA, LSID) combination and the equivalent (DAF, OAF) network address pair is established in the boundary function.

For LU-LU sessions, since each secondary LU can have an active session with only one primary LU at a time, the network address of the secondary LU suffices to identify the session to the adjacent PU_T4 or PU_T5 boundary function, which can then derive the network address of the primary LU.

FID4

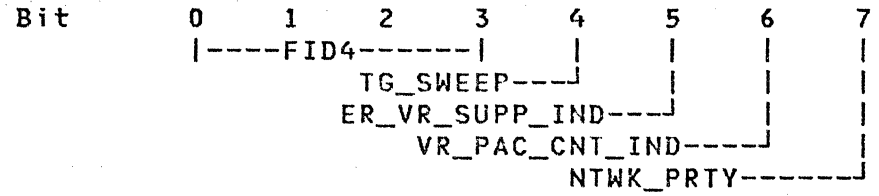
FID4 is the format used between adjacent subarea nodes, provided that both support ER and VR protocols. (FID0|1 is used if either node does not support ER and VR protocols.)

The FID4 TH appears as:

Byte		
0	FID4--Format Identification TG_SWEEP--TG Sweep Indicator ER_VR_SUPP_IND--ER and VR Support Indicator VR_PAC_CNT_IND--VR Pacing Count Indicator NTWK_PRTY--Network Priority	Reserved Byte
2	IERN--Initial Explicit Route Number ERN--Explicit Route Number	VRN--Virtual Route Number Reserved Bits TPF--Transmission Priority Field
4	VR_CWI--Virtual Route Change Window Indicator TG_NONFIFO_IND--TG nonFIFO Indicator VR_SQTI--Virtual Route Sequencing and Type Indicator TG_SNF--Transmission-Group Sequence Number Field	
6	VRPRQ--Virtual Route Pacing Request VRPRS--Virtual Route Pacing Response VR_CWRI--Virtual Route Change Window Reply Indicator VR_RWI--Virtual Route Reset Window Indicator VR_SNF_SEND--Virtual Route Send Sequence Number Field	
8	DSAF--Destination Subarea Address Field	
12	OSAF--Origin Subarea Address Field	
16	Reserved SNAI--SNA Indicator MPF--Mapping Field Reserved EFI--Expedited Flow Indicator	Reserved Byte
18	DEF--Destination Element Field	
20	OEF--Origin Element Field	
22	SNF--Sequence Number Field	
24	DCF--Data Count Field	

Byte 0

The bit configuration is:



FID4--Format Identification: 0100

TG_SWEEP--TG Sweep:

1 This PIU does not overtake any PIU ahead of it in the transmission group.
0 No restriction.

The TG Sweep indicator, when set to 1 in the TH of a PIU, prevents that PIU from getting ahead of other PIUs flowing on the transmission group. Thus, various RUs, such as NC_ER_OP and NC_ER_INOP, can be processed in the order they originate. This is performed in the transmission group control components of NTWK.PC.

ER_VR_SUPP_IND--ER and VR Support Indicator:

1 The explicit route traversed by this PIU includes at least one node that does not support ER and VR protocols.
0 Each node on the explicit route traversed by this PIU supports ER and VR protocols.

This bit is set to the appropriate value when the FID4 TH is originated (and/or when a FID4 TH replaces a FID0|1 TH) to indicate whether some subarea node on the route specified by this FID4 TH does not support ER and VR protocols. The transformation between FID4 and FID1 (or FID0 for non-SNA traffic) takes place in nodes adjacent to the subarea node that does not support ER and VR protocols. The VRN, IERN, and ERN fields must be set to 0 when this bit is set to 1. If this bit is on and the SNAI indicator is on, then FID4 is changed to FID1. Receipt of the first PIU, with this bit set to 1, on an ER results in activation of the ER (ERN = 0) and VR (VRN = 0, TPF = 0).

VR_PAC_CNT_IND--Virtual Route Pacing Count Indicator:

1 Pacing count, on the VR specified in VRID, has reached a value of 0.
0 Pacing count, on the VR specified in VRID, has not reached a value of 0.

This bit is used to initiate implementation specific action to hasten the flow of isolated VRPRSs to the Vk_PAC_CNT sender. It indicates that the VR_PAC_CNT sender cannot send any more PIUs, because its pacing count has reached 0.

NTWK_PRTY--Network Priority:

1 PIU flows at network priority, which is the highest transmission priority.
0 PIU flows at a lower priority, as specified in TPF.

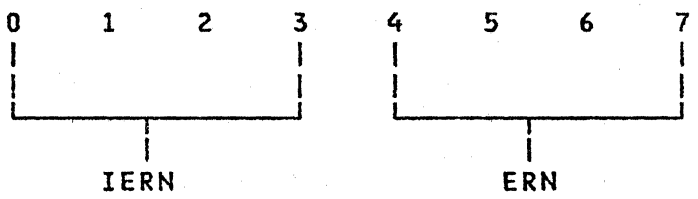
This bit provides a transmission priority higher than those specified by TPF (see VRID, byte 3). It is used to transmit PIUs that must flow ahead of others--for example, to prevent network congestion. Currently, it is used only for isolated VRPRSs.

Byte 1

Reserved

Byte 2

The bit configuration is:



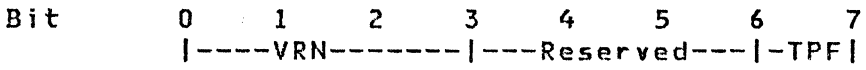
IERN--Initial Explicit Route Number: Currently, this field has the same value as VRN (byte 3).

ERN--Explicit Route Number: The ERN in a TH identifies an explicit route direction of flow (i.e., in the direction the TH is flowing). Two ERNs--one ERN for each direction of flow--together with the two subarea addresses (OSAF, DSAF), specify an explicit route.

Byte 3

VRID--Virtual Route Identifier: This field, along with DSAF and OSAF, identifies a virtual route.

The bit configuration is:



VRN--Virtual Route Number

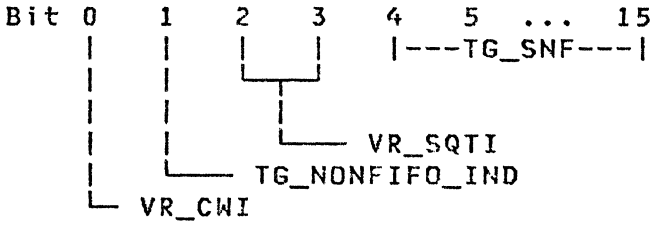
TPF--Transmission Priority Field:

00 low priority
01 medium priority
10 high priority

TPF, if the NTWK_PRTY bit is set to 0, carries the PIU transmission priority to be used by transmission groups on the explicit route; otherwise, this field is ignored by the transmission group, and is simply used to identify the virtual route.

Bytes 4-5

The bit configuration is:



VR_CWI--Virtual Route Change Window Indicator:

1 Decrement window size.
0 Increment window size.

This indicator is used to change the window size of the virtual route by 1, in the direction of flow of this PIU. Any transmission group on the virtual route can turn this bit on if it is congested; each subsequent transmission group on the virtual route leaves it on. The window size mechanism controls the PIU flow on the virtual route. Window size is the amount by which the current virtual route pacing count is incremented when a VRPRS bit set to VR_PAC_RSP is received.

TG_NONFIFO_IND--TG non-FIFO Indicator:

1 TG FIFO is not required.
0 TG FIFO is required.

This indicator identifies whether or not FIFO discipline is to be enforced in transmitting PIUs through the transmission groups to prevent the PIUs getting out of sequence during transmission over the TGs. (A transmission group may have more than one link, providing simultaneous transmission within the group.) When TG FIFO is not indicated, virtual route end-to-end resequencing is coordinated by UPMs.

VR_SQTI--VR Sequence and Type Indicator:

00 nonsequenced, nonsupervisory
01 nonsequenced, supervisory
10 singly sequenced

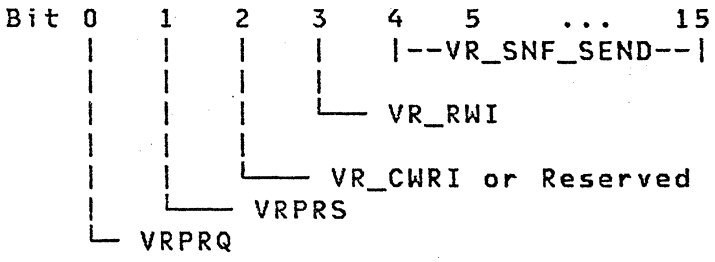
These bits specify the PIU type. All network control (NC) RUs flowing between ER managers or VR managers are coded nonsequenced, nonsupervisory. An isolated VRPRS is coded nonsequenced, supervisory. All other PIUs are coded singly sequenced.

TG_SNF--Transmission-Group Sequence Number Field:

It is used by transmission group protocols to provide TG FIFO when TG_NONFIFO_IND is set to FIFO; otherwise, it is reserved.

Bytes 6-7

The bit configuration is:



VRPRQ--Virtual Route Pacing Request:

1 VR pacing request is sent asking for a VR pacing response.
0 No VR pacing response is requested.

VRPRS--Virtual Route Pacing Response:

1 VR pacing response is sent in response to a VRPRQ bit set to VR_PAC_RQ.
0 No pacing response is sent.

Virtual route pacing provides traffic flow control between the two ends of a VR. In contrast to session pacing, virtual route pacing operates on a group of sessions (on each VR)

and extends only up to the VR endpoints (subarea nodes). The virtual route pacing uses a window size, say k. The sender (endpoint of a VR) can transmit k PIUs for every VRPRS set to VR_PAC_RSP received from the other VR endpoint.

VR_CWRI or Reserved: If VRPRS is set to VR_PAC_RSP, this bit is VR_CWRI; otherwise, it is reserved.

VR_CWRI--Virtual Route Change Window Reply Indicator:

1 Decrement window size by 1 without going under the minimum window size, as specified in NC_ACTVR.
0 Increment window size by 1 without exceeding the maximum window size, as specified in NC_ACTVR.

This bit permits changing the window size by 1 for PIUs received by the sender of this bit.

VR_RWI--Virtual Route Reset Window Indicator:

1 Reset window size to the minimum specified in NC_ACTVR.
0 Do not reset window size.

This bit is set to indicate severe congestion in a node on the virtual route. When a VR endpoint receives this bit set to 1, it reduces the window size to the minimum window size.

VR_SNF_SEND--Virtual Route Send Sequence Number Field:

This number is initialized by a parameter carried in NC_ACTVR. The sender increments this count by 1 for every PIU sent. The VR receiver checks the sequenced arrival of PIUs by examining the VR_SNF_SEND values. This field is reserved except when VR_SQTI is set to SING_SEQ.

Bytes 8-11

DSAF--Destination Subarea Address Field:

A four-byte destination subarea address field.

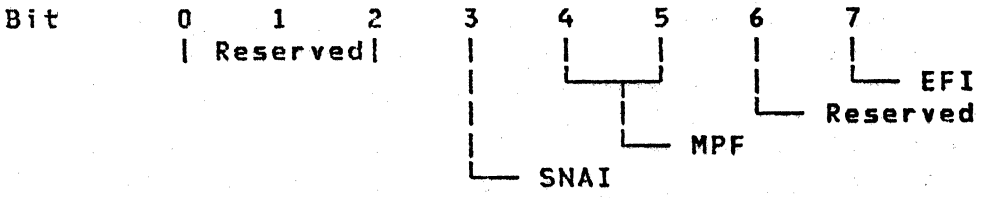
Bytes 12-15

OSAF--Origin Subarea Address Field:

A four-byte origin subarea address field.

Byte 16

The bit configuration is:



SNAI--SNA Indicator:

0 -SNA
1 SNA

This bit is used to identify whether the PIU originated or is destined for an SNA or non-SNA device. If this bit is off, the TH is converted to FID0 in the node supporting the non-SNA device. If this bit is on and the ER_VR_SUPP_IND is set to PRE_ER_VR, the TH is converted to FID1 in the node adjacent to the node not supporting ERs and VRs.

MPF, EFI--As explained earlier

Byte 17

Reserved

Bytes 18-19

DEF--Destination Element Field:

A two-byte destination element address field. The complete network address results from the combination of DSAF and DEF. An element address of 0 denotes the PU_T4|5 controlling the associated subarea.

Bytes 20-21

OEF--Origin Element Field:

A two-byte origin element address field. The complete network address results from the combination of OSAF and OEF.

Bytes 22-23

SNF--Sequence Number Field, as described earlier

Bytes 24-25

DCF--Data Count Field, as described earlier.

FIDF

FIDF is used between the adjacent subarea nodes if both support ER-VR protocols.

The FIDF appears as:

Byte

0	FIDF--Format Identification Reserved Bits (4)	Reserved Byte
2	Command Format	Command Type
4	Command Sequence Number	
6	Reserved Bytes (18)	
24	DCF—Data Count Field	

Byte 0

Bits 0-3: FIDF--Format Identification: 1111

Bits 4-7: Reserved

Byte 1: Reserved

Byte 2

Command Format--X'01' for currently defined format (only value defined).

Byte 3

Command Type--X'01' for currently defined type, to indicate Transmission-Group Sequence-Number-Field Wrap Acknowledgment command (only value defined).

Bytes 4-5

Command Sequence Number--Identifier sequence number for Transmission-Group Sequence-Number-Field Wrap Acknowledgment command. This sequence number is distinct from the Transmission-Group Sequence Number field in the FID4 TH.

Bytes 6-23: Reserved

Bytes 24-25

DCF--Same as described earlier

This page intentionally left blank

REQUEST/RESPONSE HEADER

The request/response header (RH) is a three-byte field; it may be a request header or a response header. The format rules for the RH are the same for all FID types.

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; code points for the various indicator values are presented in Appendix D.

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 corresponding to the four principal function interpreters in each half-session (see Chapter 1): session control (SC), network control (NC), data flow control (DFC), or function management data (FMD).

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, later in this chapter).

For SC, NC, and DFC RUs, this indicator is always set to Format 1.

For (SSCP,SSCP), (SSCP,PU), and (SSCP,LU) sessions, Format 1 indicates that the request RU includes a network services (NS) header (see Chapter 6) 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 zero on positive responses. For LU-LU sessions that do not support FM headers, the meaning of this indicator on requests and positive responses is implementation dependent. In both cases (i.e., regardless of whether FM headers are supported) the Format indicator value on negative responses is implementation dependent. (A BIND session parameter indicates whether FM headers are supported by the session. See Chapter 13 and Appendix E for details on BIND.)

Sense Data Included Indicator (SDI): Indicates that a four-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 must be included on negative responses and on EXCEPTION REQUESTs (see Chapter 4), 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 four-byte format shown in Appendix G.)

Chaining Control: Indicates that a sequence of contiguous transmitted requests is being grouped in a "chain" (see Chapter 5). Two indicators, Begin Chain indicator (BCI) and End Chain indicator (ECI), together denote the relative position of the associated RU within a chain. The one values of these indicators (BCI = 1 and ECI = 1) are referred to as BC and EC, respectively.

(BC, ¬EC) = first RU of chain
(¬BC, ¬EC) = middle RU of chain
(¬BC, EC) = last RU of chain
(BC, EC) = only RU of chain

Responses are always marked "only RU of chain."

Form of Response Requested: In a request header, defines the response protocol to be executed by the request receiver.

There are three bits in a request header that 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. See Chapter 4 for additional details.)

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 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 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 on (three combinations); a response to a request returns the same (DR1I, DR2I) bit combination.

The setting of the DR1I, DR2I, and ERI bits varies by RU category (SC, NC, DFC, FMD). Chapters 4 and 13 define the settings for SC; Chapter 5, DFC; Chapters 7-9, network services FMD; and Chapters 11-12, NC.

In the case of LU-LU sessions, a BIND parameter (see Chapter 13 and Appendix E) specifies the form(s) of response to be requested during the session. For sessions that use sync point protocols (e.g., are activated with TS profile 4), RQD2 asks for the commitment of a unit of work that is shared between the session partners. RQD1 is used to request a response when the current unit of work is not to be committed. The meaning of responses when sync point

protocols are used is given in Figure 2-3. For nonzero session types that do not use sync point protocols, the specific meanings of the DR1I and DR2I bits are defined in SNA LU-LU Session Types. For LU-LU session type 0, the specific meanings of the DR1I and DR2I bits (and distinctions among the three settings) are implementation-dependent.

The (DR1I, DR2I, ERI) = (0, 0, 1) combination is reserved.

Queued Response Indicator (QRI): In a response header for a normal-flow RU, the Queued Response indicator denotes whether the response is to be enqueued in TC queues (Q_PAC, BF.Q_PAC, and Q_TC_TO_DFC): 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.

Response Type: 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.

There are three kinds of positive and negative responses corresponding to the three valid (DR1I, DR2I) combinations allowed on requests. The settings of the DR1I and DR2I bits in a response must equal the settings of the DR1I and DR2I bits of the form-of-response-requested field of the corresponding request header.

Pacing: In a request header, the Pacing Request indicator denotes that the sending CPMGR can accept a Pacing Response indicator.

The Pacing Response indicator in a response header is used to indicate to the receiving CPMGR that additional requests may be sent on the normal flow. The Pacing Response indicator may be on 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 4). Isolated and nonisolated pacing responses are functionally equivalent.

REQUEST	VALID RESPONSE	MEANING OF RESPONSE
RQN =(0,0,0)	None	
RQD1=(1,0,0)	+RSP1=(1,0,0) -RSP1=(1,0,1)	positive response negative response
RQE1=(1,0,1)	-RSP1=(1,0,1)	negative response
RQD2=(0,1,0)	+RSP2=(0,1,0) -RSP2=(0,1,1)	positive sync point response negative sync point response
RQE2=(0,1,1)	-RSP2=(0,1,1)	negative sync point response
RQD3=(1,1,0)	+RSP3=(1,1,0) -RSP3=(1,1,1)	positive sync point response negative sync point response
RQE3=(1,1,1)	-RSP3=(1,1,1)	negative sync point response

Notes:
1. Values displayed in this table are in the order (DR1I,DR2I,ERI) for requests and (DR1I,DR2I,RTI) for responses.
2. Each definite- or exception-response chain (see Chapter 5) has the same setting of (DR1I,DR2I)--either (1,0) or (0,1)--on all requests with ECI = ¬EC. When DR1I = 1 on these requests, the End-Chain request can carry (DR1I,DR2I) = (1,0)|(1,1). When DR2I = 1 on these requests, the End-Chain request can carry only (DR1I,DR2I) = (0,1). ERI is 0 only for definite-response chains and when ECI = EC.

Figure 2-3. Request/Response Combinations For Sessions Using Sync Points

Bracket Control: Used to indicate the beginning or end of a group of exchanged requests and responses called a bracket (see Chapter 5).

Change Direction Control (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 (see Chapter 5).

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 (for example, 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 eight bit binary representation.

PATH CONTROL NETWORK

The path control network (NTWK.PC) provides for the routing and transmission of message units such that the node/link configuration of the network is essentially transparent to the sending component; specifically, NTWK.PC provides for the routing and transmission of:

- PU-PU flow requests and responses between PU services managers
- Session-activation and session-deactivation requests and responses between common session control (CSC) managers
- Requests and responses between paired primary and secondary half-sessions

NTWK.PC is composed of:

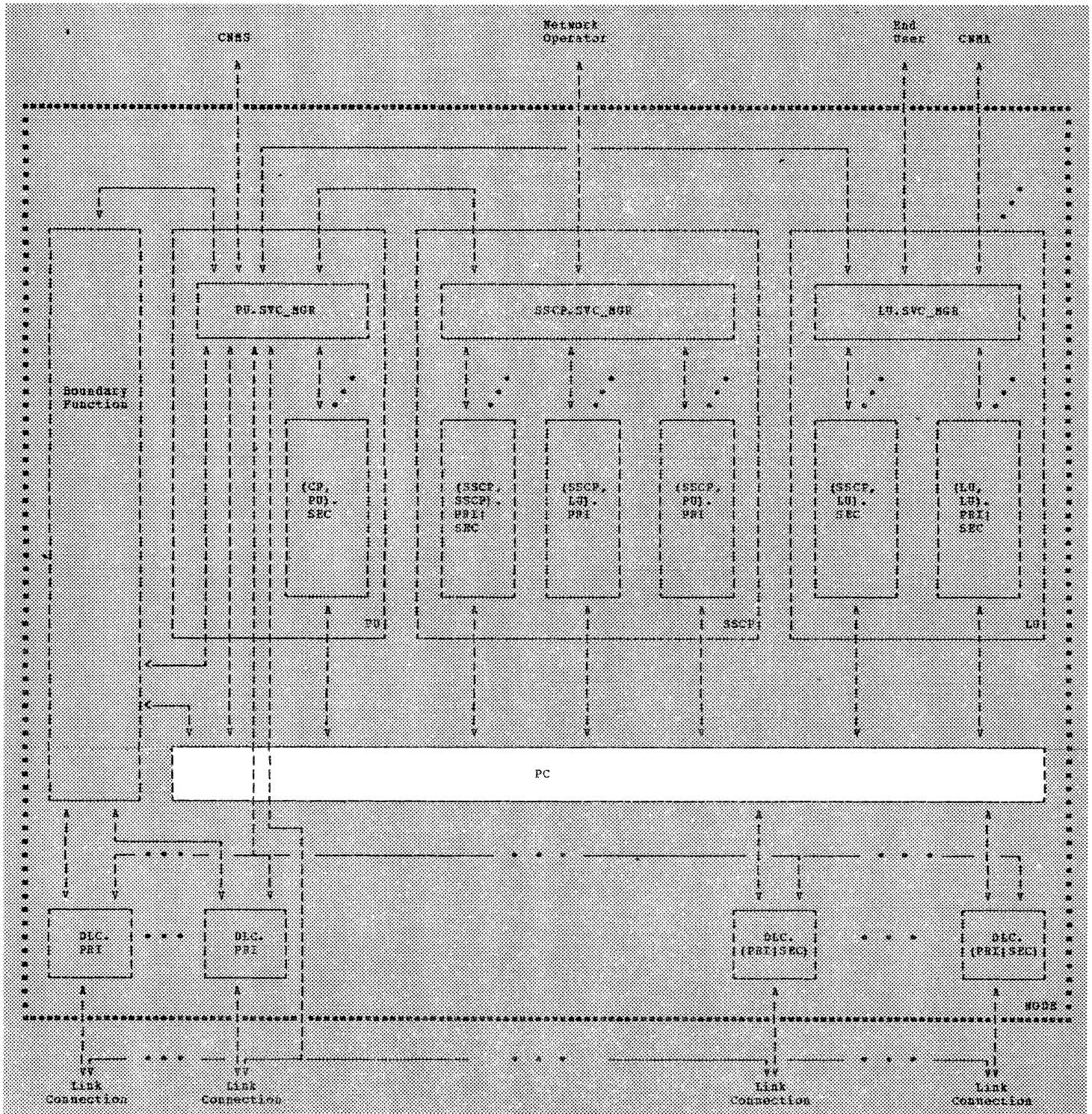
- Subarea routing protocol machines (PC_SA)--applicable to PU_T4 and PU_T5 subarea nodes--that route message units within and between PU_T4 and/or PU_T5 subarea nodes
- Route extension protocol machines that route message units between PU_T4 or PU_T5 subarea nodes providing boundary function support and adjacent PU_T1 or PU_T2 peripheral nodes, consisting of:
  - -- Boundary function path control (BF.PC) protocol machines applicable to PU_T4 or PU_T5 subarea nodes providing boundary function support
  - -- Path control protocol machines applicable to PU_T1 peripheral nodes (PC_T1)
  - -- Path control protocol machines applicable to PU_T2 peripheral nodes (PC_T2)
- A set of data link control (DLC) and link-connection protocol machines that interconnect and effect the transmission of path information units (PIUs) between the path control protocol machines.

NTWK.PC is aided in the routing function by the following components, which process message units en route between PC_SA and BF.PC in PU_T4 or PU_T5 subarea nodes providing boundary function support:

- Boundary function transmission control (BF.TC, described in Chapter 4)
- The boundary function PU services manager (BF.PU.SVC_MGR, not described)
- The boundary function LU services manager (BF.LU.SVC_MGR, not described)
- The common session control manager component of the PU services manager (PU.SVC_MGR.CSC_MGR, described in Chapter 13)

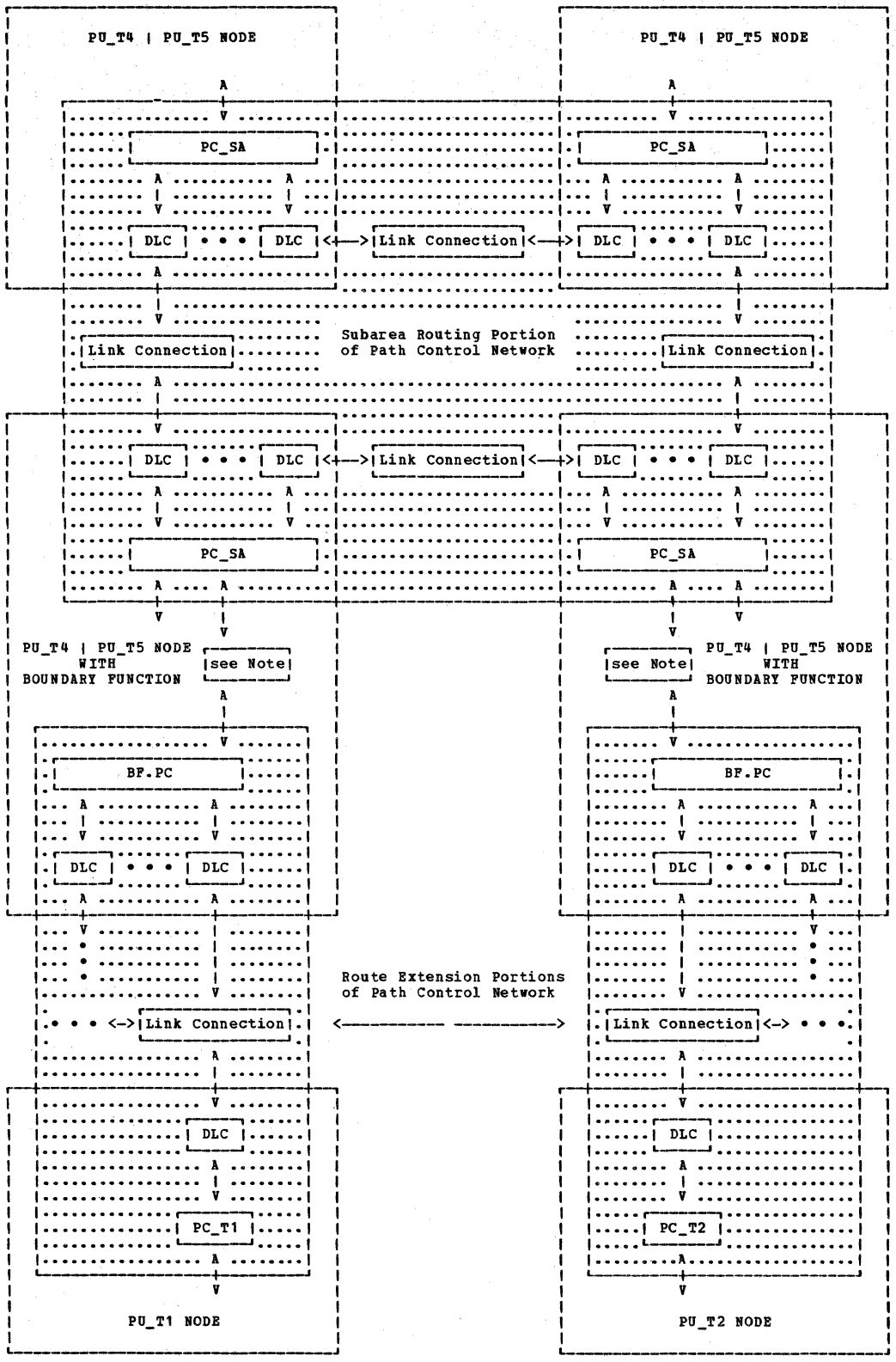
The relationship of path control network components to other components in a node are shown in Figure 3-1 on page 3-3. The structure of the path control network is shown in Figure 3-2 on page 3-4.

PC_SA, BF.PC, PC_T1,. and PC_T2 are described in this chapter.



Note: Only a type 5 node contains an SSCP; a type 1, 2, or 4 node contains a PUCP (not shown), which is a subset of an SSCP.

Figure 3-1. Structural Overview of a Node



Note: BF.TC, BF.PU.SVC_MGR, BF.LU.SVC_MGR, and PU.SVC_MGR.CSC_MGR aid in the routing of message units between PC_SA and BF.PC.

Figure 3-2. Structure of Path Control Network (NTWK.PC)

SEGMENTING OF MESSAGES BY PATH CONTROL NETWORK

The path control network transports only whole message units between sending components (origins) and receiving components (destinations). However, in order to allow more efficient utilization of transmission paths, NTWK.PC provides for:

- Segmenting of message units, and message-unit segments, into smaller message-unit segments prior to transmission
- Transmission of message-unit segments
- Assembly of message-unit segments into a whole message unit prior to delivery to the destination

Segmenting may optionally be performed by PC_SA, BF.PC, PC_T1, or PC_T2. Segment assembly may optionally be performed by PC_SA, PC_T1, or PC_T2. Segment assembly is not performed by BF.PC--this applies to message traffic received from PC_SA and from PC_T1 or PC_T2. It is necessary that the segmenting and segment assembly protocols (defined in this chapter) be coordinated between sending and receiving path control components to achieve correct network behavior.

NTWK.PC uses the Mapping field (MPF) in the path information unit (PIU)--described in Chapter 2--to perform segmenting and segment assembly.

Because message units sent by PC_T1 or PC_T2 may be segmented and BF.PC does not provide segment assembly, PC_SA may receive message-unit segments; these message-unit segments may be further segmented by PC_SA.

The following message units are never segmented by NTWK.PC:

- PU-PU flow requests and responses--network control (NC) RUs
- Session-activation and session-deactivation requests and responses
- Message units sent by PC_SA that are destined for BF.PC
- Message units originated and sent by components of NTWK.PC--transmission group sequence-number-field wrap acknowledgment (TG_SNF_WRAP_ACK) and virtual route pacing response (VRPRS); these message units are described in this chapter

An additional requirement is that a first segment contain at least 10 bytes of message-unit data; hence, message units or first segments less than 11 bytes in length are never segmented.

SUBAREA ROUTING PATH CONTROL

PC_SA provides for the routing of message units within and between subarea nodes.

Each PC_SA protocol machine is composed of:

- Transmission group control (TGC)--which sends PIUs over transmission groups between adjacent subarea nodes
- Explicit route control (ERC)--which provides subarea routing for PIUs over explicit routes
- Virtual route control (VRC)--which provides for the routing of message units within and between subarea nodes over virtual routes.

In prose, the suffix, PC_SA, is generally implied and not explicitly used to denote the components of PC_SA; hence, the components of PC_SA are simply referred to as TGC, ERC, and VRC. In procedures and figures, the suffix, _SA, is generally omitted; the components of PC_SA are denoted as PC.TGC, PC.ERC, and PC.VRC. In general, these conventions are followed in this and other chapters.

FID4 and FIDF PIUs (described in Chapter 2) flow between the PC_SA protocol machines.

The structure of PC_SA is shown in Figure 3-3 on page 3-7. This is followed by a description of each of its components--TGC, ERC, and VRC.

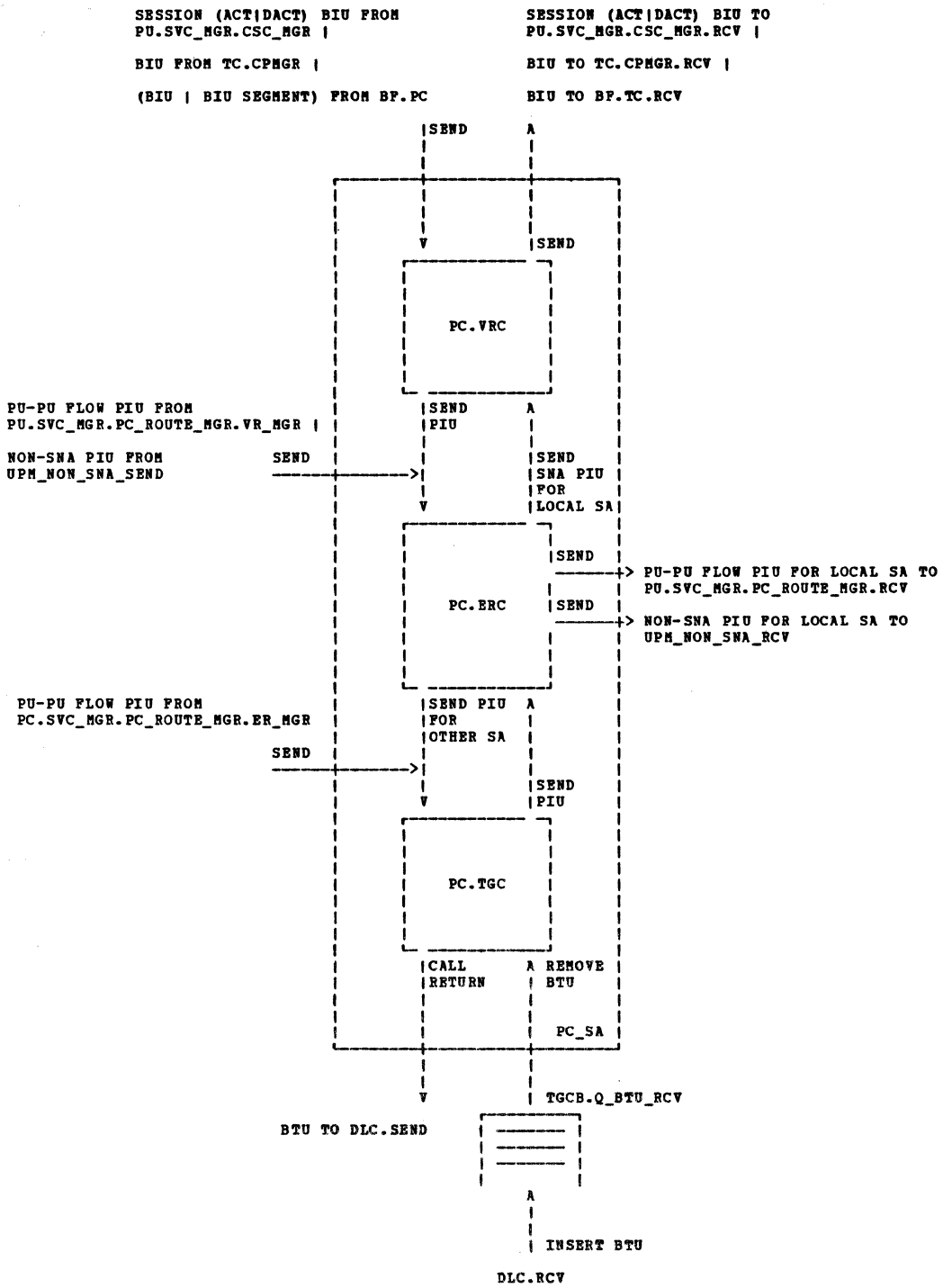


Figure 3-3. Structure of Subarea Routing Path Control (PC_SA) for Subarea Nodes

TRANSMISSION GROUP CONTROL

This section discusses the controls provided by transmission groups for message units sent between adjacent subarea nodes.

A transmission group (TG) is a bidirectional logical connection between two adjacent subarea nodes. A TG can include a single link or multiple links connecting the two adjacent nodes. A TG is defined as either a single-link TG or a multiple-link TG at system-definition time. A single-link TG is a TG that can include only one link; a multiple-link TG is a TG that can include one or more links.

A TG can be denoted by a transmission group number (TGN) and the subarea addresses of the two adjacent nodes. Between two nodes it is possible to have a maximum of 255 TGs by assigning different TGNs (1-255). TGN value 0 has a special meaning during XID Format 2 exchange (see Chapter 11) and is not used to identify a TG. For routing purposes within a subarea node, a TG is identified by a TGN and the subarea address of an adjacent node.

A TG is operational when one or more links that are operational are associated with the TG. A link is operational when the link has been activated and link station contact between the two adjacent nodes has been established over the link.

Within a subarea node, a link is associated with a TG when a contacted subarea node adjacent link station is associated with the TG; hence, a TG is operational when one or more contacted subarea node adjacent link stations are associated with the TG. A TG becomes operational when the first adjacent link station is successfully contacted and associated with the TG. A subarea node adjacent link station can be defined at system-definition time to be associated with a TG when it is contacted, or can be dynamically associated with a TG during XID Format 2 exchange, which is part of the contact process (described in Chapter 11). An operational TG becomes inoperative when the last remaining adjacent link station associated with the TG is disassociated from the TG. An adjacent link station is disassociated from a TG when link-level discontact between the adjacent link stations occurs or when the link becomes inoperative.

The functions associated with sending data over transmission groups are implemented in transmission group control (TGC). TGC is a sublayer in subarea routing path control; it is positioned between explicit route control (ERC) and data link control (DLC). See Figure 3-3 on page 3-7.

TGC sends and receives path information units (PIUs) and basic transmission units (BTUs), and transforms PIUs to BTUs and BTUs to PIUs. The relationships of the PIU and BTU to each other and to the layers of SNA are described in Chapter 2.

TGC effects the transmission of PIUs from ERC or the ER manager (Chapter 12) in one subarea node to ERC in an adjacent subarea node. Within a subarea node, TGC routes PIUs from ERC or the ER manager to DLC as BTUs, and BTUs from DLC to ERC as PIUs. TGC manages and controls:

- Sending and receiving of PIUs over a TG
- Transmission of BTUs over the links associated with a TG, at a level above and distinct from DLC functions

For multiple-link TGs, TGC consolidates multiple physical links into a single logical link.

TGC provides both send and receive functions relative to a TG. For each TG, TGC in one subarea node works in conjunction with TGC in an adjacent subarea node to provide two-way communication between the two nodes. Therefore, for each direction of transmission on a TG, there is a sending TGC in one node and a receiving TGC in the other node. Hence, within a subarea node, TGC consists of both sending and receiving components. The BTU is the message unit passed between TGCs in adjacent subarea nodes.

For each TG in a subarea node, a transmission group control block (TGCB) is used to provide storage for the variables and constants associated with the transmission group. A TGCB represents a TG in a subarea node and is used by TGC to control the functions associated with the TG. The format of the TGCB is shown in Appendix A.

The primary functions associated with TGC are described next. This is followed by an illustration of the structure of TGC and by the detailed TGC procedures and FSMs.

TGC FUNCTIONS

Blocking and Deblocking

The PIU is the message unit passed between TGC and ERC within a subarea node. The BTU is the message unit passed between TGC and DLC within a subarea node and between a sending TGC and a receiving TGC in adjacent subarea nodes. When blocking and deblocking are not performed, a BTU consists of a single PIU. When blocking and deblocking are performed, a BTU can consist of one or more PIUs--the number of PIUs in a BTU being determined by the number and length of the PIUs available to be transmitted at the sending node and the maximum BTU length that is permitted to be transmitted on the TG at the sending node.

TGC provides blocking and deblocking only for single-link TGs; it does not provide blocking and deblocking for multiple-link TGs.

For a TG, a sending TGC and a receiving TGC must provide complementary support relative to blocking and deblocking for a direction of transmission between the two nodes: if a sending TGC performs blocking, the receiving TGC must perform deblocking. Blocking and deblocking support does not have to be the same for the two directions of transmission on a TG: they can be supported in neither direction, both directions, or in one direction and not the other. The maximum BTU length that can be transmitted can also be different for the two directions of transmission.

Whether or not blocking and deblocking are to be performed for a direction of transmission between two adjacent subarea nodes on a TG is defined at system-definition time for each of the nodes. Two attributes relative to blocking and deblocking support are maintained in each TGCB: one for blocking (on send) and one for deblocking (on receive). The maximum BTU length that can be transmitted for a direction of transmission on a TG is the smallest of the maximum BTU lengths that the receiving node is capable of receiving on the links in the TG; it is defined at system-definition time or established during XID Format 2 exchange, and maintained in the TGCB.

When blocking is performed, a sending TGC assembles a BTU to pass to DLC for transmission to the adjacent node such that the BTU consists of the maximum number of currently available PIUs without exceeding the maximum BTU length that can be transmitted. When deblocking is performed, a receiving TGC disassembles a BTU received from DLC into individual PIUs and sends the PIUs one by one to ERC.

BTU Retransmission

In order to expedite the transmission of BTUs to an adjacent subarea node when transmission errors occur on links associated with a multiple-link TG, TGC provides for the retransmission of BTUs over one or more other links associated with the TG.

If a BTU transmission is associated with a multiple-link TG, DLC notifies TGC that the BTU transmission is in error recovery procedure (ERP) mode the first time a transmission error is detected. After this notification, DLC continues to attempt to transmit the BTU to the adjacent link station, as determined by DLC error recovery parameters, until the BTU is successfully transmitted or the link becomes inoperative. DLC does not notify TGC of subsequent transmission errors associated with a BTU transmission, but does notify TGC when a BTU transmission is successfully completed or abandoned.

Each time TGC is notified that a BTU transmission is in ERP mode, it passes the BTU to DLC for transmission over another TG link on which the BTU has not been or is not being transmitted, provided the BTU has not been successfully transmitted over some link before the retransmission assignment is effected.

Therefore, for multiple-link TGs, TGC retransmits a BTU on another link associated with a TG each time the BTU encounters transmission problems on a link, until the BTU is successfully transmitted, or it has been or is being transmitted on all the links associated with the TG.

The link selection algorithm used to determine the next TG link over which a BTU is to be transmitted upon transmission error notification is implemention dependent.

Conversion of a Too-Long PIU to an EXR

When TGC receives a PIU from ERC or the ER manager for transmission to an adjacent subarea node, it checks that the length of the PIU does not exceed the maximum BTU length that is permitted to be transmitted on the TG.

If the length of a PIU does exceed the maximum BTU length, TGC converts the PIU to an EXCEPTION REQUEST (EXR)--discussed in Chapter 2--and sets the sense code to X'800A' to indicate "too-long PIU." This conversion truncates the original PIU and is performed regardless of the attributes of the original PIU, e.g., first, middle, or last BIU segment; request or response RH. The resultant EXR PIU is transmitted in place of the original PIU.

Transmission by Priority and Time of Arrival

When TGC receives a PIU from ERC or the ER manager for transmission to an adjacent subarea node, the PIU has a priority indicated by the Network Priority (NTWK_PRTY) bit and Transmission Priority field (TPF) in the FID4 TH, as follows:

NTWK_PRTY	TPF	Priority	
N_PRTY	Any Value	1	(Highest Priority)
¬N_PRTY	H_PRTY	2	
¬N_PRTY	M_PRTY	3	
¬N_PRTY	L_PRTY	4	(Lowest Priority)

TGC inserts each PIU into a TG transmission priority list (TGCB.PRTY_SEND_PIU_LIST) according to its priority. PIUs with the same priority are inserted in order of arrival. When requested by DLC, TGC builds a BTU for transmission by removing the oldest, highest priority PIU from the list. The result is that PIUs are transmitted on the basis of their priority and time of arrival, with higher priority PIUs transmitted first, and PIUs with the same priority transmitted in the order of their arrival.

When the TG send traffic rate for higher priority PIUs is high, or there are transmission errors on links associated with a TG for a sustained length of time, it is possible for lower priority PIUs to reside in the transmission priority list for an indeterminate length of time, while higher priority PIUs continue to be received and transmitted. An implementation-dependent algorithm may optionally be used to effect the transmission of the lower priority PIUs, based on the length of time they have been in the transmission priority list, so that they are not delayed excessively.

TG Sequencing and Resequencing

When multiple links are associated with a TG, the BTUs transmitted over the TG may be received at the adjacent node out of sequence relative to their order of transmission. This can happen as a result of different length BTUs being transmitted on the links, different transmission speeds on the links, or transmission errors on the links associated with a TG.

In addition, when multiple links are associated with a TG, duplicates of the same BTU may be received at the adjacent node. This can happen because a BTU may be transmitted on more than one link when transmission errors occur.

To prevent PIUs from getting out of sequence and PIU duplication on multiple-link TGs, TGC provides for PIU sequencing at a sending node and PIU resequencing at a receiving node.

TGC performs sequencing and resequencing only if a TG is a multiple-link TG and only for PIUs with the TG Non-FIFO indicator (TG_NONFIFO_IND) in the FID4 TH set to FIFO.

TGC at a sending node sequences a PIU by assigning it a TG sequence number (0-4095) that is carried in the TG Sequence Number field (TG_SNF) of the FID4 TH. When necessary, TGC at a receiving node resequences PIUs by discarding duplicate PIUs and holding nonduplicate, out-of-sequence PIUs in a re-FIFO list (TGCB.REFIFO_PIU_LIST) until they can be sent to ERC in the proper sequence.

The TG sequence numbers flowing in one direction are independent of the TG sequence numbers flowing in the opposite direction; TGC in each node maintains two counters for sequence numbers in the TGCB, one for sending and the other for receiving. The TG sequence numbers for a direction of transmission wrap from 4095 to 0.

TG Sweep

A TG sweep is the suspension of new transmissions over a TG until all previously initiated TG transmissions have been completed. TGC performs a TG sweep only for multiple-link TGs.

TGC performs a TG sweep by not passing any new (non-retransmission) BTUs to DLC for links associated with the TG, until at least one copy of each BTU previously passed to DLC for the TG has been successfully transmitted to the adjacent node and all BTU transmissions for the TG have completed. When a TG sweep is performed, all the TG associated BTUs passed to DLC prior to the sweep will be received by the adjacent node before any TG associated BTUs transmitted subsequent to the sweep.

When a TG sweep is performed before a PIU is passed to DLC, the PIU will not get ahead of any previous PIU. When a TG sweep is performed after a PIU is passed to DLC, no subsequent PIU will get ahead of the PIU.

TG sweep is performed under the following conditions:

- The TG Sweep bit (TG_SWEEP) in the FID4 TH is set to SWEEP.

  Some RUs (e.g., NC_DACTVR) are required not to overtake any previous RU with equal or higher transmission priority when they traverse an explicit or virtual route. TG_SWEEP is set to SWEEP for these PIUs. Before TGC sends a PIU with TG_SWEEP=SWEEP, it performs a TG sweep. Therefore, a PIU with TG_SWEEP=SWEEP cannot get ahead of any previously transmitted PIU as it is transmitted over a TG.

- The TG_NONFIFO_IND in the FID4 TH is set to FIFO and the TG_SNF is set to 0.

  When a TG-sequenced PIU with TG_SNF=4095 is transmitted to an adjacent node, the sending TG sequence number counter is wrapped to 0. The result is that groups of TG-sequenced PIUs with TG_SNF values 0-4095 are transmitted on a TG. Different groups of TG-sequenced PIUs with TG_SNF values 0-4095 are separated to prevent a receiving TGC from receiving PIUs from different groups of TG-sequenced PIUs at the same time.

To separate different groups of TG-sequenced PIUs, TGC performs a TG sweep before a PIU with TG_SNF=0 is transmitted and immediately after it is transmitted. These two TG sweeps force a TG-sequenced PIU with TG_SNF=0 to flow alone on a multiple-link TG and delimits groups of TG-sequenced PIUs at the receiving TGC, as follows.

The sweep performed before TG_SNF=0 is transmitted causes all PIUs in the previous group to be received at the receiving TGC before a new group is started. The sweep performed after TG_SNF=0 is transmitted causes the PIU with TG_SNF=0, and any duplicates of it, to be received at the receiving TGC before any PIUs with TG_SNF>0 in the new group.

Therefore, for a receiving TGC, a PIU with TG_SNF=0 received in sequence indicates the start of a new group of TG-sequenced PIUs--all PIUs, including duplicates, belonging to the previous group have already been received, and no additional PIUs belonging to the new group have yet been transmitted to be confused with PIUs of the previous group. A PIU with TG_SNF=0 received out of sequence is always a duplicate and can be safely ignored and discarded.

TG_SNF Wrap Acknowledgment

For a multiple-link TG, a receiving TGC saves nonduplicate, out-of-sequence, TG-sequenced PIUs in a re-FIFO list (TGCB.REFIFO_PIU_LIST) until the TG-sequenced PIUs with the preceding sequence numbers are received, at which time the in-sequence PIUs in the re-FIFO list are removed from the list and sent to ERC in proper sequence. It is possible for the re-FIFO list to build up and contain a substantial number of PIUs that could simultaneously become available to be sent to ERC on receipt of a single PIU with a "missing" preceding sequence number. In practice, the time required to service the re-FIFO list could be such that PIUs belonging to the next group of TG-sequenced PIUs could be received before all the PIUs of a preceding group have been removed from the re-FIFO list.

In order to avoid the possibility of having PIUs from more than one group of TG-sequenced PIUs reside in the re-FIFO list at the same time, or the need to maintain multiple re-FIFO lists, a receiving TGC sends a Transmission-Group Sequence-Number-Field Wrap Acknowledgment (TG_SNF_WRAP_ACK) to a sending TGC when the receiving TGC has received and passed to ERC all the PIUs in one group of TG-sequenced PIUs and is ready to receive the next group of TG-sequenced PIUs.

A sending TGC suspends transmitting when the last PIU (TG_SNF=4095) in a group of TG-sequenced PIUs is transmitted and does not resume transmitting until it receives the TG_SNF_WRAP_ACK.

The TG_SNF_WRAP_ACK is sent from the receiving TGC to the sending TGC when a TG-sequenced PIU with TG_SNF=4095 is sent to ERC. This acknowledgment serves as a pacing response relative to groups of TG-sequenced PIUs; it informs the sending TGC that the receiving TGC has received and completed processing a group of TG-sequenced PIUs and is ready to receive another group.

The TG_SNF_WRAP_ACK is formatted as a FIDF TH PIU, as described in Chapter 2. Each TG_SNF_WRAP_ACK is sequence numbered to allow a receiving TGC to differentiate between different wrap acknowledgments. The sequence numbers are needed because TG_SNF_WRAP_ACKs, like all other PIUs, are subject to duplication in the case of BTU retransmissions over multiple links. The TG_SNF_WRAP_ACK sequence numbers flowing in one direction are independent of the TG_SNF_WRAP_ACK sequence numbers flowing in the other direction; TGC in each node maintains two counters for wrap acknowledgment sequence numbers in the TGCB, one for sending and the other for receiving.

In order to expedite the resumption of transmissions by TGC in an adjacent node after a group of TG-sequenced PIUs have been transmitted and transmissions are suspended, TGC transmits a pending TG_SNF_WRAP_ACK PIU ahead of any other pending network traffic.

To prevent a deadlock condition on a TG in the event that both ends of the TG reach the end of a group of TG sequenced PIUs at the same time, a pending TG_SNF_WRAP_ACK PIU is transmitted even when the transmission of other network traffic is suspended.

Setting Virtual Route Pacing Control Indicators in FID4 TH

When a subarea node is moderately congested with network traffic and it is communicated relative to a TG, TGC sets an indicator in the TH of each FID4 PIU transmitted on the TG. When a subarea node is severely congested with network traffic and it is communicated relative to a TG, TGC sets an indicator in the TH of each FID4 PIU received on the TG. Different indicators are set for moderate congestion and severe congestion. The indicators are used by virtual route control to control virtual route pacing counts.

When moderate congestion exists, TGC sets the Virtual Route Change Window indicator (VR_CWI) in each FID4 PIU transmitted to DEC_WS (decrement window size). This causes a gradual decrease in the virtual route window size for PIUs flowing in the same direction as the PIU being transmitted by TGC.

When severe congestion exists, TGC sets the Virtual Route Reset Window indicator (VR_RWI) in each FID4 PIU received to RESET_WS (reset window size). This causes the virtual route window size for PIUs flowing in the direction opposite to that of the PIU being received by TGC to be reset to the minimum window size specified in the NC_ACTVR RU.

The determination of when moderate congestion or severe congestion exists within a node, and the communication of these states to a TG, are implementation-dependent.

For a complete description of virtual route pacing and the function of the VR_CWI and VR_RWI bits in the FID4 TH, see the discussion, "Virtual Route Pacing," in the "Virtual Route Control" section of this chapter.

TH Conversion for Pre-ER-VR Subarea Nodes

TGC provides support for communication between a subarea node that supports ER and VR protocols and an adjacent pre-ER-VR subarea node that does not support ER and VR protocols. Whether or not an adjacent subarea node supports ER and VR protocols is defined at system-definition time or is established during XID Format 2 exchange and is maintained in the TGCB.

A TG between a subarea node that supports ER and VR protocols and an adjacent pre-ER-VR subarea node is always a single-link TG. For these TGs, TGC converts PIU transmission headers, as follows.

The TH of each PIU to be transmitted to an adjacent pre-ER-VR subarea node is converted from FID4 to either FID1 or FID0 before the PIU is transmitted. If the SNA indicator in the FID4 TH is set to SNA, the TH is converted to FID1; otherwise, it is converted to FID0.

The TH of each PIU received from an adjacent pre-ER-VR subarea node is converted from either FID0 or FID1 to FID4 before it is sent to ERC. If the TH is FID1, the SNA indicator in the FID4 TH is set to SNA; if the TH is FID0, the SNA indicator in the FID4 TH is set to ¬SNA.

BTU Validity Checking

TGC performs a validity check on each BTU received over a TG before the BTU is accepted by DLC at a subarea node. This check is performed after a BTU has successfully passed DLC-level error checking mechanisms, but before it is accepted and acknowledged by DLC. If a BTU passes the validity check, DLC accepts the BTU and passes it to TGC. If a BTU does not pass the validity check, DLC discards the BTU.

This check, which augments DLC-level error checking mechanisms, serves to reduce the probability that erroneous or invalid data will be propagated. It significantly enhances transmission reliability and provides verification that subsequent processing of a BTU by TGC can be successfully accomplished.

TGC checks the validity of a BTU by checking that the PIU FID and Data Count field values in the BTU are valid, that certain minimum PIU length requirements are met, and that the length of the BTU as reflected by its internal PIU makeup corresponds to the length of the BTU as received by DLC.

STRUCTURE OF TGC

The structure of TGC is shown in Figure 3-4 on page 3-22. This figure illustrates:

- The major procedures, lists, and queues of TGC
- The internal TGC data flows
- The protocol boundaries and data flows between TGC and other node components
- The scheduling of TGC components within separate processes according to the meta-implementation execution model (see Appendix C)

In Figure 3-4, the elements of TGC associated with sending are located on the left and those associated with receiving are located on the right. For the most part, the TG send and receive functions in a node are independent of each other. The exceptions to this independence are when a TG_SNF_WRAP_ACK PIU or a TG-sequenced PIU with TG_SNF=4095 is received. When a TG_SNF_WRAP_ACK is received, the receive function resets the transmissions-suspended state of the send function, and when the last PIU in a group of TG-sequenced PIUs (PIU with TG_SNF=4095) is received and passed to ERC by the receive function, the receive function inserts a TG_SNF_WRAP_ACK into the TG priority send list such that it will be the next PIU to be transmitted to the adjacent node by the send function.

Figure 3-4 also illustrates that, for both the send and receive portions of TGC, different procedures are executed under the control of different schedulers. Procedures dispatched under control of the higher-level scheduler are located at the top and those executed as a result of DLC scheduler dispatching are located at the bottom. The horizontal dotted line passing through the PIU send list, TGCB.PRTY_SEND_PIU_LIST, and the BTU receive queue, TGCB.Q_BTU_RCV, in Figure 3-4, denotes the boundary between the procedures associated with the higher-level scheduler and the procedures associated with a DLC scheduler.

For the send portion of TGC:

- The procedure, PC.TGC.LIST_BY_PRTY, is dispatched as a subthread of a higher-level scheduler thread when a PIU is sent to it.

- The procedure, PC.TGC.SEND, is invoked by a call from DLC.SEND, which is dispatched under control of a DLC scheduler.

- The PIU send list, TGCB.PRTY_SEND_PIU_LIST, located between the TGC send procedures, PC.TGC.LIST_BY_PRTY and PC.TGC.SEND, separates the higher-level and DLC scheduler execution threads and provides linkage for data flow between the procedures.

- The procedure, UPM_PC_TGC_SEND_BTU_MGR, is invoked by a call from DLC.RCV, which is dispatched under control of a DLC scheduler.

For the receive portion of TGC:

- The procedure, PC.TGC.RCV_BTU_CK, is invoked by a call from DLC.RCV, which is dispatched under control of a DLC scheduler.

- The procedure, PC.TGC.DEQ_Q_BTU_RCV, is dispatched as a thread of the higher-level scheduler.

- The procedure, PC.TGC.RCV, is dispatched as a subthread of the higher-level scheduler when a BTU is sent to it from PC.TGC.DEQ_Q_BTU_RCV.

- The BTU receive queue, TGCB.Q_BTU_RCV, located between DLC.RCV and the procedure, PC.TGC.DEQ_Q_BTU_RCV, separates the higher-level and DLC scheduler execution threads and provides linkage for data flow from the DLC scheduler component, DLC.RCV, to the higher-level scheduler procedure, PC.TGC.DEQ_Q_BTU_RCV.

Appendix C discusses the execution model in greater detail.

The detailed TGC procedures and FSMs are presented next. In general, the order of presentation is as follows: (1) the send procedures, in the order they are invoked to process message units flowing from ERC to DLC; (2) the receive procedures, in the order they are invoked to process message units flowing from DLC to ERC; (3) the procedures common to both send and receive; and (4) the FSMs.

The TGCB, which represents a TG in a subarea node, is referenced extensively in TGC; it is described in Appendix A.

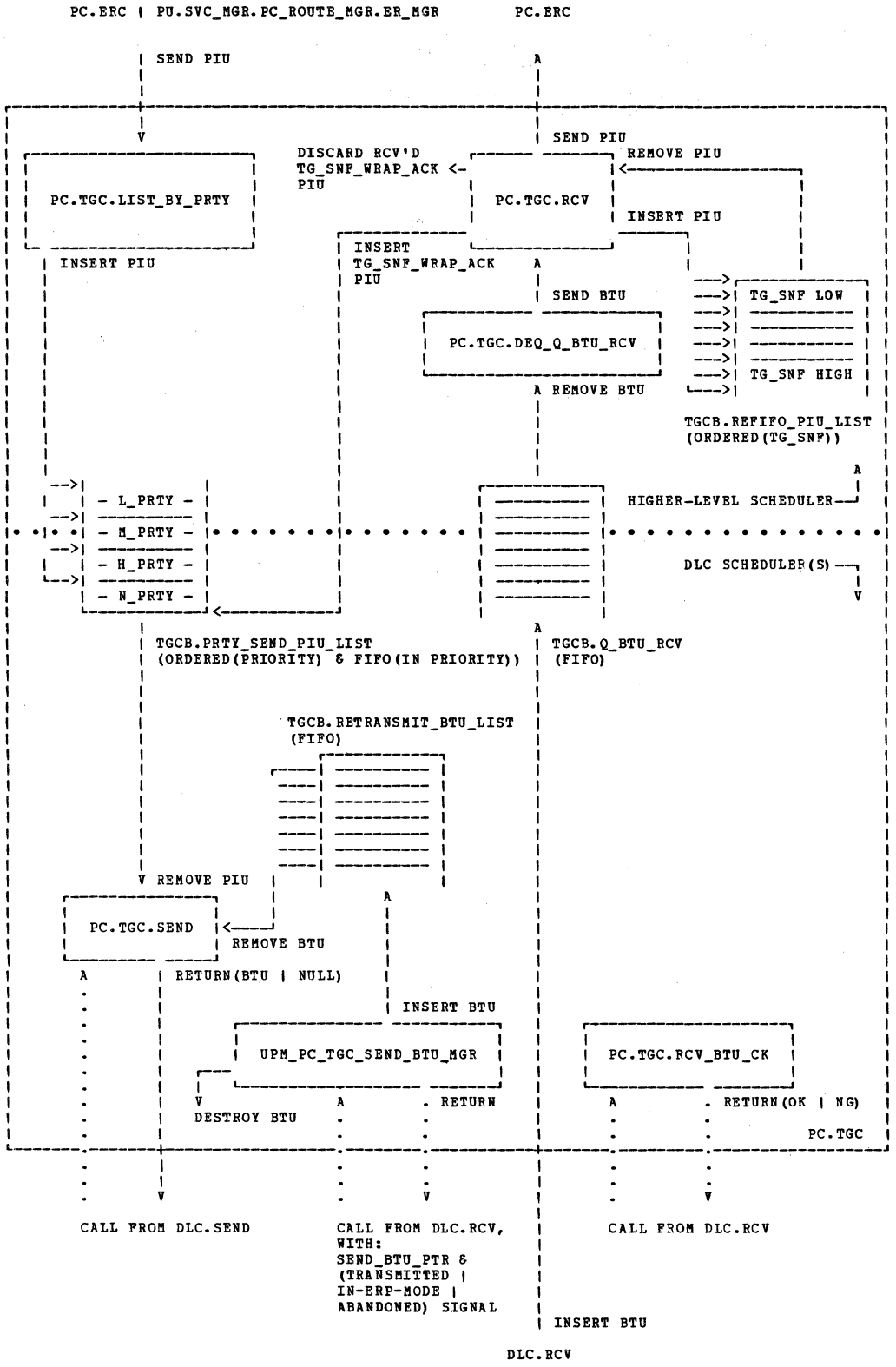


Figure 3-4. Structure of Transmission Group Control (PC.TGC)

```
PC.TGC.LIST_BY_PRTY: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE IS DISPATCHED WHEN A PIU IS SENT TO IT FROM PC.ERC OR
 │                 PU.SVC_MGR.PC_ROUTE_MGR.ER_MGR (CHAPTER 12).
 │
 │                 IF NO ADJACENT LINK STATION IS ASSOCIATED WITH THE TG, THE TG IS NOT
 │                 OPERATIONAL, AND THE PIU IS DISCARDED.
 │
 │                 IF ONE  OR MORE ADJACENT LINK  STATIONS ARE ASSOCIATED WITH  THE TG,
 │                 THE TG IS OPERATIONAL, AND THE PIU IS PROCESSED AS FOLLOWS.
 │
 │                 1. IF TG TRACE IS ACTIVE FOR THE TG, A TRACE OF THE PIU IS PROVIDED.
 │
 │                 2. THE PIU IS ASSIGNED ONE OF FOUR TG SEND PRIORITY VALUES ACCORDING
 │                    TO THE  NTWK_PRTY BIT AND  TPF IN THE  FID4 TH.  THE  ASSIGNED TG
 │                    SEND PRIORITY VALUE IS STORED IN MUCB.TG_SEND_PRTY.
 │
 │                 3. IF  THE  ADJACENT  SUBAREA  NODE  DOES  NOT  SUPPORT  ER  AND  VR
 │                    PROTOCOLS, THE  PIU TH IS CONVERTED  FROM FID4 TO EITHER  FID1 OR
 │                    FID0.
 │
 │                 4. IF THE LENGTH OF  THE PIU IS GREATER THAN THE  MAXIMUM BTU LENGTH
 │                    THAT  IS PERMITTED  TO  BE  TRANSMITTED ON  THE  TG,  THE PIU  IS
 │                    CONVERTED TO AN  EXCEPTION REQUEST (EXR), AND  THEREBY TRUNCATED.
 │                    THIS CONVERSION IS PERFORMED REGARDLESS  OF THE ATTRIBUTES OF THE
 │                    PIU,  E.G.,  FIRST,  MIDDLE,  OR LAST  BIU  SEGMENT;  REQUEST  OR
 │                    RESPONSE.
 │
 │                 5. THE PIU IS INSERTED INTO THE TGCB.PRTY_SEND_PIU_LIST ACCORDING TO
 │                    ITS  ASSIGNED TG  SEND PRIORITY.   PIU'S  WITH THE  SAME TG  SEND
 │                    PRIORITY ARE INSERTED FIFO.
 │
 │      INPUT:     PIU FROM PC.ERC OR  PU.SVC_MGR.PC_ROUTE_MGR.ER_MGR; MU_PTR POINTS TO
 │                 PIU, TGCB_PTR POINTS TO TGCB
 │
 │      OUTPUT:    PIU INSERTED INTO TGCB.PRTY_SEND_PIU_LIST, IF NOT DISCARDED
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 ASSIGN_TG_SEND_PRTY                PAGE 3-24
 │                 CONVERT_FID4_TO_FID1_OR_FID0       PAGE 3-25
 │                 CONVERT_PIU_TO_EXR                 PAGE 3-26
 │                 LENGTH_OF_PIU                      PAGE 3-44
 │                 UPM_TG_TRACE                       PAGE 3-45
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF  ¬EMPTY(TGCB.ASSOC_LSCB_LIST) THEN
    DO;
    .
    . IF TGCB.TG_TRACE = TRACE THEN
    .   CALL UPM_TG_TRACE('SEND');                          /* PAGE 3-45   */
    .
    . CALL ASSIGN_TG_SEND_PRTY;                             /* PAGE 3-24   */
    .
    . IF TGCB.ER_VR_SUPP = PRE_ER_VR THEN
    .   CALL CONVERT_FID4_TO_FID1_OR_FID0;                  /* PAGE 3-25   */
    .
    . IF LENGTH_OF_PIU > TGCB.MAX_SEND_BTU_LENGTH THEN      /* PAGE 3-44   */
    .   CALL CONVERT_PIU_TO_EXR;                            /* PAGE 3-26   */
    .
    . LOCK TGCB.PRTY_SEND_PIU_LIST;
    . .
    . . INSERT MU BY_ASCENDING(MUCB.TG_SEND_PRTY) IN TGCB.PRTY_SEND_PIU_LIST;
    . .
    . UNLOCK;
    .
    END;

  ELSE
    DISCARD MU;

  RETURN;

END PC.TGC.LIST_BY_PRTY;
```

```
ASSIGN_TG_SEND_PRTY: PROCEDURE;
                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE IS CALLED TO ASSIGN A TG SEND PRIORITY TO A PIU.
│
│                 THE PIU IS ASSIGNED ONE OF FOUR TG SEND PRIORITY VALUES BASED ON THE
│                 NTWK_PRTY BIT AND TPF IN THE FID4 TH, AS FOLLOWS:
│
│                   NTWK_PRTY     TPF                        TG SEND PRIORITY
│
│                   N_PRTY (1)    ANY VALUE   (**)   --->    PRTY_1  HIGHEST PRIORITY
│                  ¬N_PRTY (0)    H_PRTY      (10)   --->    PRTY_2
│                  ¬N_PRTY (0)    M_PRTY      (01)   --->    PRTY_3
│                  ¬N_PRTY (0)    L_PRTY      (00)   --->    PRTY_4  LOWEST PRIORITY
│
│                  ¬N_PRTY (0)    RESERVED    (11)   --->    PRTY_2  (SEE NOTE)
│
│                 THE ASSIGNED TG SEND PRIORITY IS  STORED IN MUCB.TG_SEND_PRTY AND IS
│                 SUBSEQUENTLY USED  TO CONTROL THE INSERTION  OF THE PIU INTO  THE TG
│                 PRIORITY SEND PIU LIST (TGCB.PRTY_SEND_PIU_LIST).
│
│      INPUT:     CURRENT PIU; POINTED TO BY MU_PTR
│
│      OUTPUT:    MUCB.TG_SEND_PRTY SET TO ASSIGNED TG SEND PRIORITY
│
│      NOTE:      PIU'S WITH TPF=B'11' ARE DISCARDED BY PC.VRC.RCV.
│
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                 PC.TGC.LIST_BY_PRTY                PAGE 3-23
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF NTWK_PRTY = N_PRTY THEN
    MUCB.TG_SEND_PRTY = PRTY_1;

  ELSE
    SELECT ANYORDER(TPF);
    .
    . WHEN(L_PRTY)
    .   MUCB.TG_SEND_PRTY = PRTY_4;
    .
    . WHEN(M_PRTY)
    .   MUCB.TG_SEND_PRTY = PRTY_3;
    .
    . OTHERWISE
    .   MUCB.TG_SEND_PRTY = PRTY_2;
    .
    END;

  RETURN;

END ASSIGN_TG_SEND_PRTY;
```

```
CONVERT_FID4_TO_FID1_OR_FID0: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  THIS PROCEDURE IS CALLED WHEN A PIU IS TO BE TRANSMITTED TO AN
   │                ADJACENT SUBAREA NODE THAT DOES NOT SUPPORT ER AND VR PROTOCOLS, TO
   │                CONVERT THE PIU TH FROM FID4 TO EITHER FID1 OR FID0. SEE NOTE.
   │
   │                IF THE SNA INDICATOR (SNAI) IN THE FID4 TH IS SET TO SNA, THE PIU TH
   │                IS CONVERTED TO FID1; OTHERWISE, IT IS CONVERTED TO FID0.
   │
   │     INPUT:     CURRENT (FID4) PIU; POINTED TO BY MU_PTR
   │
   │     OUTPUT:    FID1 OR FID0 PIU; POINTED TO BY MU_PTR
   │
   │     NOTE:      THIS PROCEDURE SETS:
   │
   │                • FID1|FID0 TH FIELDS COMMON TO FID4 THAT REQUIRE A CHANGE IN VALUE,
   │                  I.E., FID
   │
   │                • FID1|FID0 TH FIELDS NOT COMMON TO FID4 THAT REQUIRE A VALUE
   │                  DERIVED FROM THE FID4 TH, I.E., DAF, OAF--THESE ARE THE ONLY
   │                  FID1|FID0 FIELDS NOT COMMON TO FID4
   │
   │                THIS PROCEDURE DOES NOT SET THE FID1|FID0 TH FIELDS COMMON TO FID4
   │                THAT DO NOT REQUIRE A CHANGE IN VALUE, I.E., MPF, EFI, SNF, DCF
   │
   │                POSITIONAL CHANGES IN TH FIELDS BETWEEN FID4 AND FID1|FID0--ALL
   │                FIELDS EXCEPT FID--ARE HANDLED BY THE MAP_FROM_CANONICAL PROCEDURE
   │                (SEE APPENDIX B), WHICH IS CALLED BY PC.TGC.SEND USING THE
   │                ADD_PIU_TO_BTU PROCEDURE TO MAP THE PIU FROM CANONICAL FORM TO LINK
   │                FORM BEFORE IT IS TRANSMITTED OUT OF THE NODE.
   │
   │                THE FID1, FID0, AND FID4 TH FORMATS ARE DESCRIBED IN CHAPTER 2.
   │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                PC.TGC.LIST_BY_PRTY                PAGE 3-23
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF SNAI = SNA THEN
    FID = FID1;

  ELSE
    FID = FID0;


  DAF(0:NCB.SUBAREA_LEN - 1) = DSAF((32 - NCB.SUBAREA_LEN):31);

  DAF(NCB.SUBAREA_LEN:15) = DEF(NCB.SUBAREA_LEN:15);


  OAF(0:NCB.SUBAREA_LEN - 1) = OSAF((32 - NCB.SUBAREA_LEN):31);

  OAF(NCB.SUBAREA_LEN:15) = OEF(NCB.SUBAREA_LEN:15);


  RETURN;

END CONVERT_FID4_TO_FID1_OR_FID0;
```

```
CONVERT_PIU_TO_EXR: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE IS CALLED WHEN THE LENGTH  OF A PIU TO BE TRANSMITTED │
   │                 IS  GREATER THAN  THE MAXIMUM  BTU LENGTH  THAT IS  PERMITTED TO  BE │
   │                 TRANSMITTED ON  THE TG, TO CONVERT  THE PIU TO AN  EXCEPTION REQUEST │
   │                 (EXR).                                                   │
   │                                                                          │
   │                 THIS CONVERSION TRUNCATES THE PIU.                       │
   │                                                                          │
   │                 THE SENSE  DATA IS  SET TO  X'800A0000' TO  INDICATE "TOO-LONG  PIU" │
   │                 ERROR.                                                   │
   │                                                                          │
   │      INPUT:     CURRENT PIU; POINTED TO BY MU_PTR                        │
   │                                                                          │
   │      OUTPUT:    INPUT PIU CONVERTED TO EXR; POINTED TO BY MU_PTR         │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 PC.TGC.LIST_BY_PRTY              PAGE 3-23               │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  BBIUI = BBIU;                                          /* TH                 */
  EBIUI = EBIU;
  DCF = 7;

  RRI = RQ;                                              /* FH, X'07B000'      */
  RU_CTGY = FMD;
  FI =  ¬FMH;
  SDI = SD;
  BCI = BC;
  ECI = EC;

  DR1I = DR1;
  DR2I = DR2;
  ERI = ER;
  QRI =  ¬QR;
  PI =  ¬PAC;

  BBI =  ¬BB;
  EBI =  ¬EB;
  CDI =  ¬CD;
  CSI = CODE0;
  EDI =  ¬ED;
  PDI =  ¬PD;

  SNC = X'800A0000';                                     /* SNC = TOO-LONG PIU */

  RETURN;
END CONVERT_PIU_TO_EXR;
```

```
PC.TGC.SEND: PROCEDURE RETURNS(PTR);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE IS  CALLED  BY DLC.SEND  IN  SUBAREA  NODES WHEN  AN │
   │                 OPPORTUNITY EXISTS  TO TRANSMIT ANOTHER BTU  FROM PATH CONTROL  TO A │
   │                 SUBAREA NODE ADJACENT LINK STATION.   THIS PROCEDURE EITHER PASSES A │
   │                 BTU  TO DLC.SEND  TO BE  TRANSMITTED  TO THE  CURRENT ADJACENT  LINK │
   │                 STATION OR INDICATES NO DATA TO SEND.                │
   │                                                                      │
   │                 IF A  BTU IS  NOT TO  BE TRANSMITTED  TO THE  CURRENT ADJACENT  LINK │
   │                 STATION, A NULL POINTER IS RETURNED TO DLC.SEND.     │
   │                                                                      │
   │                 IF A BTU IS TO BE TRANSMITTED  TO THE CURRENT ADJACENT LINK STATION, │
   │                 A POINTER TO THE BTU IS RETURNED TO DLC.SEND.  SEE NOTE.  │
   │                                                                      │
   │      INPUT:     LSCB_PTR POINTS TO LSCB FOR CURRENT ADJACENT LINK STATION │
   │                                                                      │
   │      OUTPUT:    A POINTER TO A SEND_BTU_PIU_VECTOR_LIST--A  LIST OF PIU_VECTORS THAT │
   │                 SPECIFY THE ADDRESSES  AND LENGTHS OF THE PIU'S  THAT CONSTITUTE THE │
   │                 BTU--IS RETURNED  IF A BTU IS  TO BE TRANSMITTED; OTHERWISE,  A NULL │
   │                 POINTER IS RETURNED.                                 │
   │                                                                      │
   │      NOTE:      A  BTU  IS   PASSED  TO  DLC.SEND  BY  RETURNING  A   POINTER  TO  A │
   │                 SEND_BTU_PIU_VECTOR_LIST.  A  SEND_BTU_PIU_VECTOR_LIST IS A  LIST OF │
   │                 PIU_VECTORS.  EACH PIU_VECTOR IN THE LIST CONSISTS OF A POINTER TO A │
   │                 PIU AND A BYTE COUNT INDICATING THE LENGTH OF THE PIU, AND SPECIFIES │
   │                 A PIU THAT  IS INCLUDED IN THE  BTU.  IF BLOCKING IS  NOT PERFORMED, │
   │                 THE  LIST WILL  CONSIST OF  ONE  PIU_VECTOR ENTRY.   IF BLOCKING  IS │
   │                 PERFORMED, THE LIST MAY CONSIST OF ONE OR MORE PIU_VECTOR ENTRIES.  │
   │                                                                      │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
   │                 MULTI_LINK_TG_SEND                PAGE 3-30          │
   │                 SINGLE_LINK_TG_SEND               PAGE 3-29          │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL SEND_BTU_PTR PTR;

  TGCB_PTR = LSCB.TGCBPTR;

  SELECT ANYORDER(TGCB.MULTI_LINK_SUPP);
  .
  . WHEN(¬MULTI_LINK_TG)
  .   SEND_BTU_PTR = SINGLE_LINK_TG_SEND;                  /* PAGE 3-29    */
  .
  . WHEN(MULTI_LINK_TG)
  .   SEND_BTU_PTR = MULTI_LINK_TG_SEND;                   /* PAGE 3-30    */
  .
  END;

  RETURN(SEND_BTU_PTR);

END PC.TGC.SEND;
```

This page intentionally left blank

```
SINGLE_LINK_TG_SEND: PROCEDURE RETURNS(PTR);
                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │     FUNCTION:  THIS PROCEDURE IS INVOKED TO BUILD A SEND BTU FOR A   │
 │                SINGLE-LINK TG.                                       │
 │                                                                      │
 │                IF THE TGCB.PRTY_SEND_PIU_LIST IS EMPTY, THERE ARE NO │
 │                PIU'S AVAILABLE TO BE TRANSMITTED, AND A NULL POINTER │
 │                IS RETURNED.                                          │
 │                                                                      │
 │                IF THE TGCB.PRTY_SEND_PIU_LIST IS NOT EMPTY, A SEND   │
 │                BTU IS BUILT, AND A POINTER TO A SEND_BTU_PIU_VECTOR_  │
 │                LIST IS RETURNED.                                     │
 │                                                                      │
 │                A SEND BTU IS BUILT BY:                               │
 │                                                                      │
 │                1. CREATING A TGCB.SEND_BTU_PIU_VECTOR_LIST           │
 │                2. REMOVING A PIU FROM THE TOP OF THE TGCB.PRTY_SEND_  │
 │                   PIU_LIST                                           │
 │                3. CREATING A PIU_VECTOR THAT CONTAINS A POINTER TO    │
 │                   THE PIU AND A BYTE COUNT INDICATING THE LENGTH OF  │
 │                   THE PIU                                            │
 │                4. ADDING (FIFO) THE PIU_VECTOR TO THE TGCB.SEND_BTU_  │
 │                   PIU_VECTOR_LIST                                    │
 │                5. IF BLOCKING IS NOT PERFORMED, THE TGCB.SEND_BTU_    │
 │                   PIU_VECTOR_LIST IS LIMITED TO ONE PIU_VECTOR ENTRY.│
 │                                                                      │
 │                   IF BLOCKING IS PERFORMED, STEPS 2-4 ABOVE ARE      │
 │                   REPEATED UNTIL EITHER THE TGCB.PRTY_SEND_PIU_LIST  │
 │                   BECOMES EMPTY OR THE ADDITION OF THE NEXT PIU      │
 │                   WOULD CAUSE THE LENGTH OF THE BTU TO EXCEED THE    │
 │                   MAXIMUM BTU LENGTH THAT IS PERMITTED TO BE         │
 │                   TRANSMITTED ON THE TG.                             │
 │                                                                      │
 │                WHEN A FID4 PIU IS REMOVED FROM THE TGCB.PRTY_SEND_   │
 │                PIU_LIST AND INCLUDED IN A SEND BTU (ADJACENT NODE    │
 │                SUPPORTS ER AND VR PROTOCOLS), THE VR_CWI BIT IN THE  │
 │                FID4 TH MAY BE SET TO DEC_WS, DEPENDING ON TRAFFIC    │
 │                CONGESTION IN THE NODE.                               │
 │                                                                      │
 │     INPUT:     TGCB_PTR IS ESTABLISHED; TGCB.PRTY_SEND_PIU_LIST IS   │
 │                SOURCE FOR SEND PIU'S.                                │
 │                                                                      │
 │     OUTPUT:    A POINTER TO A SEND_BTU_PIU_VECTOR_LIST--A LIST OF     │
 │                PIU_VECTORS THAT SPECIFY THE LOCATIONS AND LENGTHS OF  │
 │                THE PIU'S THAT CONSTITUTE THE SEND BTU--IS RETURNED IF │
 │                A BTU IS TO BE TRANSMITTED; OTHERWISE, A NULL POINTER  │
 │                IS RETURNED.                                          │
 │                                                                      │
 │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
 │                PC.TGC.SEND                        PAGE 3-27          │
 │                                                                      │
 │     REFERS TO THE FOLLOWING PROCEDURE(S):                            │
 │                ADD_PIU_TO_BTU                     PAGE 3-33          │
 │                LENGTH_OF_PIU                      PAGE 3-44          │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL SEND_BTU_PIU_VECTOR_LIST_PTR PTR;
  DCL SEND_BTU_LENGTH FIXED(31) BINARY;

  TGCB.SEND_BTU_PIU_VECTOR_LIST = NULL;

  IF TGCB.BLOCKING_SUPP = BLOCKING THEN
    SEND_BTU_LENGTH = 0;

  DO UNTIL(MU_PTR = NULL);
  .
  . MU_PTR = NULL;
  .
  . LOCK TGCB.PRTY_SEND_PIU_LIST;
  . .
  . . IF  ¬EMPTY(TGCB.PRTY_SEND_PIU_LIST) THEN
  . .   DO;
  . .   .
  . .   . MU_PTR = FIRST_ENTRY(TGCB.PRTY_SEND_PIU_LIST);
  . .   .
  . .   . IF TGCB.BLOCKING_SUPP = BLOCKING &
  . .   .      SEND_BTU_LENGTH + LENGTH_OF_PIU > TGCB.MAX_SEND_BTU_LENGTH THEN /* PAGE 3-44  */
  . .   .   MU_PTR = NULL;
  . .   .
  . .   . ELSE
  . .   .   REMOVE MU_PTR->MU FROM TGCB.PRTY_SEND_PIU_LIST;
  . .   .
  . .   END;
  . .
  . UNLOCK;
  .
  . IF MU_PTR ¬= NULL THEN
  .   DO;
  .   .
  .   . IF TGCB.BLOCKING_SUPP = BLOCKING THEN
  .   .   SEND_BTU_LENGTH = SEND_BTU_LENGTH + LENGTH_OF_PIU;     /* PAGE 3-44          */
  .   .
  .   . CALL ADD_PIU_TO_BTU;                                    /* PAGE 3-33          */
  .   .
  .   . IF TGCB.BLOCKING_SUPP =  ¬BLOCKING THEN
  .   .   MU_PTR = NULL;
  .   .
  .   END;
  END;

  SEND_BTU_PIU_VECTOR_LIST_PTR = TGCB.SEND_BTU_PIU_VECTOR_LIST;

  RETURN(SEND_BTU_PIU_VECTOR_LIST_PTR);

END SINGLE_LINK_TG_SEND;
```

```
MULTI_LINK_TG_SEND: PROCEDURE RETURNS(PTR);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  THIS PROCEDURE IS INVOKED TO PROVIDE A SEND BTU FOR A MULTIPLE-LINK │
│                TG; IT EITHER RETURNS A BTU OR INDICATES NO DATA TO SEND.  │
│                                                                          │
│                THIS PROCEDURE RETURNS A NULL POINTER INDICATING NO DATA TO SEND IF │
│                A RETRANSMIT BTU IS NOT SCHEDULED TO BE TRANSMITTED TO THE CURRENT │
│                ADJACENT LINK STATION, AND (IN THE ORDER CHECKED)         │
│                                                                          │
│                1. THE TG IS IN A SWEEP STATE AND ALL PREVIOUS BTU'S PASSED TO │
│                   DLC.SEND FOR THE TG HAVE NOT BEEN SUCCESSFULLY TRANSMITTED, OR │
│                                                                          │
│                2. THE TG IS IN A SUSPEND TG SEND STATE AND A FIDF TG_SNF_WRAP_ACK │
│                   PIU IS NOT AT THE TOP OF THE TGCB.PRTY_SEND_PIU_LIST, OR │
│                                                                          │
│                3. THE TGCB.PRTY_SEND_PIU_LIST IS EMPTY, OR               │
│                                                                          │
│                4. A PIU IS REMOVED FROM THE TGCB.PRTY_SEND_PIU_LIST THAT REQUIRES │
│                   THAT THE TG BE SWEPT BEFORE IT IS TRANSMITTED.         │
│                                                                          │
│                IF MORE THAN ONE ADJACENT LINK STATION IS ASSOCIATED WITH THE TG, A │
│                RETRANSMIT BTU MAY BE SCHEDULED TO BE TRANSMITTED TO THE CURRENT │
│                ADJACENT LINK STATION; IF SO, A POINTER TO A │
│                SEND_BTU_PIU_VECTOR_LIST THAT SPECIFIES THE RETRANSMIT BTU IS │
│                RETURNED.                                                 │
│                                                                          │
│                IF A RETRANSMIT BTU IS NOT SCHEDULED, THE TG IS SWEPT, AND THE TG IS │
│                IN ONE OF THE PRE_SWEEP STATES; A POINTER TO A │
│                TGCB.SEND_BTU_PIU_VECTOR_LIST FOR A BTU BUILT ON A PRIOR CALL (FOR │
│                WHICH THE SWEEP WAS PERFORMED) IS RETURNED.               │
│                                                                          │
│                IF A RETRANSMIT BTU IS NOT SCHEDULED, A PREVIOUSLY BUILT BTU IS NOT │
│                RETURNED, THE TRANSMISSION OF A BTU IS NOT INHIBITED BY TG SWEEP OR │
│                SUSPEND TG SEND REQUIREMENTS, AND THE TGCB.PRTY_SEND_PIU_LIST IS NOT │
│                EMPTY, THEN A BTU IS BUILT BY:                            │
│                                                                          │
│                1. REMOVING A PIU FROM THE TOP OF THE TGCB.PRTY_SEND_PIU_LIST │
│                                                                          │
│                2. CREATING A PIU_VECTOR THAT CONTAINS A POINTER TO THE PIU AND A │
│                   BYTE COUNT INDICATING THE LENGTH OF THE PIU            │
│                                                                          │
│                3. CREATING A TGCB.SEND_BTU_PIU_VECTOR_LIST AND INSERTING THE │
│                   PIU_VECTOR IN THE TGCB.SEND_BTU_PIU_VECTOR_LIST        │
│                                                                          │
│                IF THE PIU DOES NOT REQUIRE THAT THE TG BE SWEPT BEFORE IT IS │
│                TRANSMITTED, A POINTER TO THE TGCB.SEND_BTU_PIU_VECTOR_LIST IS │
│                RETURNED.                                                 │
│                                                                          │
│                THE FOLLOWING FUNCTIONS ARE PERFORMED FOR FID4 PIU'S REMOVED FROM │
│                THE TGCB.PRTY_SEND_PIU_LIST:                              │
│                                                                          │
│                1. THE TG IS SWEPT BEFORE A PIU WITH TG_SWEEP=SWEEP IS TRANSMITTED. │
│                                                                          │
│                2. PIU'S WITH TG_NONFIFO_IND=FIFO ARE ASSIGNED SEQUENTIAL TG_SNF │
│                   VALUES (0-4095). THE TG SEQUENCE NUMBERS WRAP FROM 4095 TO 0. │
│                                                                          │
│                3. THE TG IS SWEPT BEFORE AND AFTER THE TRANSMISSION OF A PIU WITH │
│                   TG_NONFIFO_IND=FIFO AND TG_SNF=0.                      │
│                                                                          │
│                4. WHEN A PIU WITH TG_NONFIFO_IND=FIFO AND TG_SNF=4095 IS │
│                   TRANSMITTED, THE TRANSMISSION OF ADDITIONAL FID4 PIU'S IS │
│                   SUSPENDED, UNTIL A FIDF TG_SNF_WRAP_ACK PIU WITH THE CORRECT │
│                   CMD_SEQ_NUM IS RECEIVED FROM THE ADJACENT NODE.        │
│                                                                          │
│                5. THE VR_CWI BIT IN THE TH MAY BE SET TO DEC_WS, DEPENDING ON │
│                   TRAFFIC CONGESTION IN THE NODE.                        │
│                                                                          │
│     INPUT:     LSCB_PTR AND TGCB_PTR ARE ESTABLISHED, TGCB.SEND_PRTY_PIU_LIST IS │
│                SOURCE FOR SEND PIU'S, AND TGCB.RETRANSMIT_BTU_LIST IS SOURCE FOR │
│                RETRANSMIT BTU'S                                          │
│                                                                          │
│     OUTPUT:    A POINTER TO A SEND_BTU_PIU_VECTOR_LIST--A LIST OF PIU_VECTORS THAT │
│                SPECIFY THE LOCATIONS AND LENGTHS OF THE PIU'S THAT CONSTITUTE THE │
│                BTU--IS RETURNED IF A BTU IS TO BE TRANSMITTED; OTHERWISE, A NULL │
│                POINTER IS RETURNED.                                      │
│                                                                          │
│     NOTE:      SINCE BLOCKING IS NOT SUPPORTED ON MULTIPLE-LINK TG'S, A │
│                SEND_BTU_PIU_VECTOR_LIST FOR A BTU TRANSMITTED ON A MULTIPLE-LINK TG │
│                CONSISTS OF ONLY ONE PIU_VECTOR ENTRY.                    │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                PC.TGC.SEND                          PAGE 3-27            │
│                                                                          │
│     REFERS TO THE FOLLOWING PROCEDURE(S):                                │
│                ADD_PIU_TO_BTU                       PAGE 3-33            │
│                FSM_SUSPEND_TG_SEND                  PAGE 3-46            │
│                FSM_TG_SWEEP                         PAGE 3-46            │
│                UPM_BTU_RETRANSMIT                   PAGE 3-32            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

```
DCL SEND_BTU_PIU_VECTOR_LIST_PTR PTR;

SEND_BTU_PIU_VECTOR_LIST_PTR = UPM_BTU_RETRANSMIT;              /* PAGE 3-32              */

IF SEND_BTU_PIU_VECTOR_LIST_PTR ¬= NULL THEN
  RETURN(SEND_BTU_PIU_VECTOR_LIST_PTR);

IF FSM_TG_SWEEP ¬= RESET & TGCB.OUTSTANDING_BTU_CNT ¬= 0 THEN  /* PAGE 3-46              */
  RETURN(SEND_BTU_PIU_VECTOR_LIST_PTR);

IF FSM_TG_SWEEP = (SWEEP_BIT_PRE_SWEEP | SNF_0_PRE_SWEEP) THEN /* PAGE 3-46              */
  DO;
  . CALL FSM_TG_SWEEP('SWEEP_COMPLETE');                       /* PAGE 3-46              */
  .
  . SEND_BTU_PIU_VECTOR_LIST_PTR = TGCB.SEND_BTU_PIU_VECTOR_LIST;
  .
  . TGCB.OUTSTANDING_BTU_CNT = TGCB.OUTSTANDING_BTU_CNT + 1;
  .
  . RETURN(SEND_BTU_PIU_VECTOR_LIST_PTR);
  END;

IF FSM_TG_SWEEP = SNF_0_POST_SWEEP THEN                        /* PAGE 3-46              */
  CALL FSM_TG_SWEEP('SWEEP_COMPLETE');                         /* PAGE 3-46              */

MU_PTR = NULL;

LOCK TGCB.PRTY_SEND_PIU_LIST;
. IF  ¬EMPTY(TGCB.PRTY_SEND_PIU_LIST) &
.       (FSM_SUSPEND_TG_SEND = RESET |                         /* PAGE 3-46              */
.       FIRST_ENTRY(TGCB.PRTY_SEND_PIU_LIST)->FID = FIDF) THEN
.   REMOVE FIRST(MU) FROM TGCB.PRTY_SEND_PIU_LIST SET(MU_PTR);
UNLOCK;

IF MU_PTR = NULL THEN
  RETURN(SEND_BTU_PIU_VECTOR_LIST_PTR);

TGCB.SEND_BTU_PIU_VECTOR_LIST = NULL;

SELECT ANYORDER(FID);
.
. WHEN(FIDF)
.   DO;
.   . CALL ADD_PIU_TO_BTU;                                     /* PAGE 3-33              */
.   .
.   . SEND_BTU_PIU_VECTOR_LIST_PTR = TGCB.SEND_BTU_PIU_VECTOR_LIST;
.   .
.   . TGCB.OUTSTANDING_BTU_CNT = TGCB.OUTSTANDING_BTU_CNT + 1;
.   .
.   . RETURN(SEND_BTU_PIU_VECTOR_LIST_PTR);
.   END;
.
. WHEN(FID4)
.   DO;
.   . IF TG_NONFIFO_IND = FIFO THEN
.   .   DO;
.   .   . TG_SNF = TGCB.TG_SNF_SEND_CNTR;
.   .   .
.   .   . TGCB.TG_SNF_SEND_CNTR = TGCB.TG_SNF_SEND_CNTR + 1;
.   .   .
.   .   . IF TG_SNF = 0 THEN
.   .   .
.   .   .   IF TGCB.OUTSTANDING_BTU_CNT ¬= 0 THEN
.   .   .     CALL FSM_TG_SWEEP('SNF_0_PRE_SWEEP');            /* PAGE 3-46              */
.   .   .
.   .   .   ELSE
.   .   .     CALL FSM_TG_SWEEP('SNF_0_POST_SWEEP');           /* PAGE 3-46              */
.   .   .
.   .   . ELSE
.   .   .   DO;
.   .   .   . IF TG_SWEEP = SWEEP & TGCB.OUTSTANDING_BTU_CNT ¬= 0 THEN
.   .   .   .   CALL FSM_TG_SWEEP('SWEEP_BIT_PRE_SWEEP');     /* PAGE 3-46              */
.   .   .   .
.   .   .   . IF TG_SNF = 4095 THEN
.   .   .   .   CALL FSM_SUSPEND_TG_SEND('SUSPEND');          /* PAGE 3-46              */
.   .   .   END;
.   .   END;
.   .
.   . ELSE
.   .   IF TG_SWEEP = SWEEP & TGCB.OUTSTANDING_BTU_CNT ¬= 0 THEN
.   .     CALL FSM_TG_SWEEP('SWEEP_BIT_PRE_SWEEP');            /* PAGE 3-46              */
.   .
.   . CALL ADD_PIU_TO_BTU;                                     /* PAGE 3-33              */
.   .
.   . IF FSM_TG_SWEEP = (RESET | SNF_0_POST_SWEEP) THEN        /* PAGE 3-46              */
.   .   DO;
.   .   . SEND_BTU_PIU_VECTOR_LIST_PTR = TGCB.SEND_BTU_PIU_VECTOR_LIST;
.   .   .
.   .   . TGCB.OUTSTANDING_BTU_CNT = TGCB.OUTSTANDING_BTU_CNT + 1;
.   .   END;
.   .
.   . RETURN(SEND_BTU_PIU_VECTOR_LIST_PTR);
.   .
.   END;
  END;

END MULTI_LINK_TG_SEND;
```

```
UPM_BTU_RETRANSMIT: PROCEDURE RETURNS(PTR);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS IMPLEMENTATION-DEPENDENT PROCEDURE IS INVOKED BEFORE ATTEMPTING │
│                 TO BUILD A NEW BTU FOR A MULTIPLE-LINK TG, TO SCHEDULE THE │
│                 TRANSMISSION OF RETRANSMIT BTU'S.                        │
│                                                                          │
│                 IF TGC HAS BEEN NOTIFIED BY DLC THAT A BTU TRANSMISSION IS IN ERP │
│                 MODE (SEE NOTE), IF THE BTU HAS NOT YET BEEN RETRANSMITTED BY TGC AS │
│                 A RESULT OF THE ERP NOTIFICATION, IF THE BTU HAS NOT YET BEEN │
│                 SUCCESSFULLY TRANSMITTED TO THE ADJACENT NODE, AND IF THE BTU IS NOT │
│                 BEING TRANSMITTED TO THE CURRENT ADJACENT LINK STATION, THEN, BASED │
│                 ON AN IMPLEMENTATION-DEPENDENT ALGORITHM, THIS PROCEDURE MAY ELECT │
│                 TO SCHEDULE THE BTU TO BE RETRANSMITTED TO THE CURRENT LINK STATION. │
│                                                                          │
│                 IF THE TGCB.RETRANSMIT_BTU_LIST IS EMPTY, OR IF IT IS NOT EMPTY, BUT │
│                 A RETRANSMIT BTU IS NOT SELECTED TO BE TRANSMITTED TO THE CURRENT │
│                 ADJACENT LINK STATION, A NULL POINTER IS RETURNED.       │
│                                                                          │
│                 IF THE TGCB.RETRANSMIT_BTU_LIST IS NOT EMPTY AND A RETRANSMIT BTU IS │
│                 SELECTED TO BE TRANSMITTED TO THE CURRENT ADJACENT LINK STATION, │
│                 THEN:                                                    │
│                                                                          │
│                 1. THE POINTER TO THE SEND_BTU_PIU_VECTOR_LIST FOR THE RETRANSMIT │
│                    BTU IS REMOVED FROM THE TGCB.RETRANSMIT_BTU_LIST.     │
│                                                                          │
│                 2. IT IS RECORDED THAT THE BTU HAS BEEN PASSED TO DLC.SEND FOR │
│                    TRANSMISSION TO THE CURRENT ADJACENT LINK STATION.    │
│                                                                          │
│                 3. THE TGCB.OUTSTANDING_BTU_CNT IS INCREMENTED BY 1.     │
│                                                                          │
│                 4. A POINTER TO THE SEND_BTU_PIU_VECTOR_LIST FOR THE RETRANSMIT BTU │
│                    IS RETURNED.                                          │
│                                                                          │
│      INPUT:     LSCB_PTR POINTS TO LSCB FOR CURRENT ADJACENT LINK STATION, TGCB_PTR │
│                 POINTS TO TGCB, AND TGCB.RETRANSMIT_BTU_LIST IS SOURCE FOR │
│                 RETRANSMIT BTU'S                                         │
│                                                                          │
│      OUTPUT:    RETURN POINTER POINTS TO SEND_BTU_PIU_VECTOR_LIST FOR RETRANSMIT BTU │
│                 IF A RETRANSMIT BTU IS SELECTED TO BE TRANSMITTED TO THE CURRENT │
│                 ADJACENT LINK STATION; OTHERWISE, A NULL POINTER IS RETURNED │
│                                                                          │
│      NOTE:      FOR BTU TRANSMISSIONS ASSOCIATED WITH MULTIPLE-LINK TG'S, DLC │
│                 NOTIFIES TGC THAT A BTU TRANSMISSION IS IN ERP MODE THE FIRST TIME A │
│                 TRANSMISSION ERROR IS DETECTED.                          │
│                                                                          │
│                 SUBSEQUENT TO THE ERP NOTIFICATION, DLC CONTINUES TO ATTEMPT TO │
│                 TRANSMIT THE BTU TO THE ADJACENT LINK STATION, AS DETERMINED BY DLC │
│                 ERP PARAMETERS, UNTIL THE BTU IS SUCCESSFULLY TRANSMITTED OR THE ERP │
│                 IS TERMINATED--IN WHICH CASE, THE BTU TRANSMISSION IS ABANDONED. │
│                                                                          │
│                 THE ERP MODE NOTIFICATION IS PROVIDED REGARDLESS OF THE VALUES OF │
│                 THE DLC ERP PARAMETERS; EVEN IF NO ERP IS SPECIFIED, DLC PROVIDES │
│                 THE ERP MODE NOTIFICATION TO TGC THE FIRST TIME A TRANSMISSION ERROR │
│                 IS DETECTED.                                             │
│                                                                          │
│                 DLC DOES NOT INFORM TGC OF TRANSMISSION ERRORS ASSOCIATED WITH A BTU │
│                 TRANSMISSION SUBSEQUENT TO THE FIRST ERROR, BUT DOES INFORM TGC WHEN │
│                 A BTU TRANSMISSION HAS BEEN SUCCESSFULLY TRANSMITTED OR ABANDONED. │
│                                                                          │
│                 AN ERP NOTIFICATION INFORMS TGC THAT THE BTU ASSOCIATED WITH THE BTU │
│                 TRANSMISSION IS TO BE RETRANSMITTED TO ANOTHER LINK STATION │
│                 ASSOCIATED WITH THE TG, TO WHICH THE BTU IS NOT BEING TRANSMITTED OR │
│                 TO WHICH THE BTU TRANSMISSION HAS NOT BEEN ABANDONED, PROVIDED ONE │
│                 EXISTS.                                                  │
│                                                                          │
│                 A TRANSMISSION-SUCCESSFUL NOTIFICATION INFORMS TGC THAT A BTU NEED │
│                 NOT BE RETRANSMITTED AND ALLOWS TGC TO MANAGE THE DISPOSITION OF A │
│                 BTU.                                                     │
│                                                                          │
│                 A TRANSMISSION-ABANDONED NOTIFICATION ALLOWS TGC TO MANAGE THE │
│                 DISPOSITION OF A BTU.                                    │
│                                                                          │
│                 SEE UPM_PC_TGC_SEND_BTU_MGR ON PAGE 3-35.                │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 MULTI_LINK_TG_SEND                  PAGE 3-30            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RETRANSMIT_BTU_PTR PTR;

  RETRANSMIT_BTU_PTR = NULL;              /* NORMAL RETURN PTR VALUE      */

  /*  FUNCTION AS DESCRIBED ABOVE */

  RETURN(RETRANSMIT_BTU_PTR);

END UPM_BTU_RETRANSMIT;
```

```
ADD_PIU_TO_BTU: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE IS CALLED TO ADD A PIU TO A SEND BTU; IT PERFORMS THE
 │                 FOLLOWING FUNCTIONS.
 │
 │                 1. IF  THE PIU  IS  THE FIRST  PIU TO  BE  ADDED TO  A  SEND BTU,  A
 │                    TGCB.SEND_BTU_PIU_VECTOR_LIST IS CREATED;  OTHERWISE, ONE ALREADY
 │                    EXISTS.
 │
 │                 2. A PIU_VECTOR (DEFINED IN APPENDIX A) IS CREATED.
 │
 │                 3. THE PIU_VECTOR.PIU_LENGTH IS SET EQUAL TO THE LENGTH OF THE PIU.
 │
 │                 4. IF THE PIU TH IS FID4, THE VR_CWI  BIT IN THE TH IS SET TO DEC_WS
 │                    IF FSM_VR_WINDOW_SIZE INDICATES MODERATE CONGESTION.
 │
 │                 5. THE PIU IS CONVERTED FROM CANONICAL  FORM TO THE FORM REQUIRED TO
 │                    BE SENT OVER A LINK.
 │
 │                 6. THE PIU_VECTOR.PIU_PTR IS SET TO POINT TO THE PIU.
 │
 │                 7. A   POINTER  TO   THE   PIU_VECTOR  IS   ADDED   (FIFO)  TO   THE
 │                    TGCB.SEND_BTU_PIU_VECTOR_LIST.
 │
 │      INPUT:     THE MU_PTR POINTS  TO THE PIU.  IF  TGCB.SEND_BTU_PIU_VECTOR_LIST IS
 │                 NULL, A  TGCB.SEND_BTU_PIU_VECTOR_LIST DOES NOT EXIST;  OTHERWISE, A
 │                 TGCB.SEND_BTU_PIU_VECTOR_LIST ALREADY EXISTS, AND A POINTER TO IT IS
 │                 ESTABLISHED.
 │
 │      OUTPUT:    IF  REQUIRED,   A  TGCB.SEND_BTU_PIU_VECTOR_LIST   IS  CREATED.    A
 │                 PIU_VECTOR IS  CREATED, SET TO INDICATE  THE LOCATION AND  LENGTH OF
 │                 THE PIU, AND  ADDED TO THE TGCB.SEND_BTU_PIU_VECTOR_LIST.   FOR FID4
 │                 PIU'S, THE VR_CWI BIT IN THE TH MAY BE SET TO DEC_WS.
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 MULTI_LINK_TG_SEND                  PAGE 3-30
 │                 SINGLE_LINK_TG_SEND                 PAGE 3-29
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 FSM_VR_WINDOW_SIZE                  PAGE 3-47
 │                 LENGTH_OF_PIU                       PAGE 3-44
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF TGCB.SEND_BTU_PIU_VECTOR_LIST = NULL THEN
    NEWLIST TGCB.SEND_BTU_PIU_VECTOR_LIST ENTRY_NAME(PIU_VECTOR) FIFO;

  CREATE PIU_VECTOR;                    /* PIU_VECTOR IS DEFINED IN APPENDIX A      */

  PIU_VECTOR.PIU_LENGTH = LENGTH_OF_PIU;                    /* PAGE 3-44            */

  IF FID = FID4 &
       FSM_VR_WINDOW_SIZE = MODERATE_CONGESTION THEN        /* PAGE 3-47            */
    VR_CWI = DEC_WS;

  CALL MAP_FROM_CANONICAL;                                  /* APPENDIX B           */

  PIU_VECTOR.PIU_PTR = MU_PTR;

  INSERT PIU_VECTOR LAST IN TGCB.SEND_BTU_PIU_VECTOR_LIST;

  RETURN;

END ADD_PIU_TO_BTU;
```

```
UPM_AGING_ALGORITHM: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  WHEN THE TG SEND  TRAFFIC RATE FOR HIGHER PRIORITY PIU'S  IS HIGH OR │
 │                 WHEN TRANSMISSION  ERROR RATES  ON LINKS  ASSOCIATED WITH  A TG  ARE │
 │                 HIGH,  IT IS  POSSIBLE FOR  LOWER PRIORITY  PIU'S TO  RESIDE IN  THE │
 │                 TGCB.PRTY_SEND_PIU_LIST  FOR INDETERMINATE  LENGTHS  OF TIME,  WHILE │
 │                 HIGHER PRIORITY PIU'S CONTINUE TO BE RECEIVED AND TRANSMITTED.       │
 │                                                                      │
 │                 THE   EXPOSURE  TO   INDETERMINATE  DELAY,   PARTLY  ASCRIBABLE  TO │
 │                 TRANSMISSION PRIORITY, EXISTS ONLY FOR PIU'S WITH NTWK_PRTY=¬N_PRTY, │
 │                 AND IS INVERSELY PROPORTIONAL TO THE PIU TPF VALUE.  │
 │                                                                      │
 │                 THIS OPTIONAL,  IMPLEMENTATION-DEPENDENT AGING ALGORITHM IS  USED TO │
 │                 PROMOTE LOWER PRIORITY PIU'S  WITHIN THE TGCB.PRTY_SEND_PIU_LIST, TO │
 │                 HIGHER PRIORITY, BASED ON  THE LENGTH OF TIME THEY HAVE  BEEN IN THE │
 │                 LIST,  SO  THAT THEY  ARE NOT  DELAYED FOR  INDETERMINATE LENGTHS  OF │
 │                 TIME, BECAUSE OF TRANSMISSION PRIORITY, IN THE EVENT OF HIGH TRAFFIC │
 │                 OR ERROR RATE CONDITIONS.                            │
 │                                                                      │
 │                 THE  INVOCATION  OF  THIS   PROCEDURE  IS  IMPLEMENTATION-DEPENDENT; │
 │                 THEREFORE, THIS  PROCEDURE IS NOT  INVOKED WITHIN  THE ARCHITECTURAL │
 │                 DESCRIPTION.                                         │
 │                                                                      │
 │      NOTE:      THE NTWK_PRTY BIT AND THE TPF IN A FID4 TH ARE NOT CHANGED.  │
 │                                                                      │
 │                 THE CHRONOLOGICAL INTEGRITY OF  THE TGCB.PRTY_SEND_PIU_LIST RELATIVE │
 │                 TO EQUAL OR HIGHER PRIORITIES IS  MAINTAINED; THE PROMOTION OF LOWER │
 │                 PRIORITY PIU'S TO HIGHER  PRIORITY IS DONE IN A MANNER  SUCH THAT NO │
 │                 PIU IS PLACED AHEAD OF AN "OLDER" PIU OF EQUAL OR HIGHER PRIORITY.   │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  /* FUNCTION AS DESCRIBED ABOVE */

  RETURN;

END UPM_AGING_ALGORITHM;
```

```
UPM_PC_TGC_SEND_BTU_MGR: PROCEDURE(SEND_BTU_PTR,BTU_TRANSMISSION_STATUS);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  THIS IMPLEMENTATION-DEPENDENT PROCEDURE IS CALLED BY DLC.RCV IN │
   │                SUBAREA NODES WHEN A BTU TRANSMISSION ASSOCIATED WITH A │
   │                MULTIPLE-LINK TG IS SUCCESSFULLY TRANSMITTED, ENTERS ERP MODE, OR IS │
   │                ABANDONED. THIS PROCEDURE PERFORMS FUNCTIONS REQUIRED BY TGC WHEN │
   │                THESE EVENTS OCCUR.                                   │
   │                                                                      │
   │                THIS PROCEDURE PERFORMS FUNCTIONS ASSOCIATED WITH THE REQUIREMENT │
   │                THAT TGC RETRANSMIT A BTU OVER ANOTHER LINK IN A MULTIPLE-LINK TG │
   │                EACH TIME THE TRANSMISSION OF THE BTU OVER A LINK IS IMPEDED BY │
   │                TRANSMISSION PROBLEMS, PROVIDED THERE IS ANOTHER OPERATIONAL LINK IN │
   │                THE TG OVER WHICH THE BTU IS NOT BEING OR HAS NOT BEEN TRANSMITTED │
   │                AND THAT THE BTU IS NOT SUCCESSFULLY TRANSMITTED ON SOME LINK BEFORE │
   │                THE RETRANSMISSION CAN BE EFFECTED.                   │
   │                                                                      │
   │                A CONSEQUENCE OF THE BTU RETRANSMIT FUNCTION IS THAT BTU'S │
   │                TRANSMITTED OVER MULTIPLE-LINK TG'S CAN BE PASSED TO MORE THAN ONE │
   │                LINK. THIS IMPOSES THE REQUIREMENT THAT TGC MANAGE THE FINAL │
   │                DISPOSITION OF BTU'S TRANSMITTED OVER MULTIPLE_LINK TG'S, I.E., TO │
   │                DESTROY A BTU WHEN IT IS NO LONGER NEEDED--THE BTU HAS BEEN EITHER │
   │                SUCCESSFULLY TRANSMITTED OR ABANDONED ON ALL THE LINKS TO WHICH IT │
   │                HAS BEEN PASSED. THIS PROCEDURE PERFORMS THIS FUNCTION. │
   │                                                                      │
   │                AN ADDITIONAL REQUIREMENT ASSOCIATED WITH BTU TRANSMISSIONS OVER │
   │                MULTIPLE-LINK TG'S IS BTU TRANSMISSION ACCOUNTING FOR TG SWEEP │
   │                CONTROL PURPOSES, I.E., TO DECREMENT THE TGCB.OUTSTANDING_BTU_CNT BY │
   │                1 WHEN A BTU TRANSMISSION HAS COMPLETED--A BTU PASSED TO A LINK HAS │
   │                BEEN EITHER SUCCESSFULLY TRANSMITTED OR ABANDONED. THIS PROCEDURE │
   │                PERFORMS THIS FUNCTION.                               │
   │                                                                      │
   │                THIS PROCEDURE PERFORMS THE FOLLOWING FUNCTIONS.      │
   │                                                                      │
   │                FOR A "TRANSMITTED" CALL:                             │
   │                                                                      │
   │                1. THE TGCB.OUTSTANDING_BTU_CNT IS DECREMENTED BY 1.  │
   │                                                                      │
   │                2. IF THE BTU IS IN THE TGCB.RETRANSMIT_BTU_LIST, IT IS REMOVED FROM │
   │                   THE LIST.                                          │
   │                                                                      │
   │                3. IF THE BTU IS NOT BEING TRANSMITTED TO ANY OTHER ADJACENT LINK │
   │                   STATION, IT IS DESTROYED; OTHERWISE, IT IS RECORDED THAT THE BTU │
   │                   HAS BEEN SUCCESSFULLY TRANSMITTED AND THAT THE TRANSMISSION TO │
   │                   THE CURRENT LINK STATION HAS COMPLETED.            │
   │                                                                      │
   │                FOR AN "ERP_NOTIFICATION" CALL:                       │
   │                                                                      │
   │                   IF THE BTU HAS ALREADY BEEN SUCCESSFULLY TRANSMITTED, OR IF THERE │
   │                   IS NOT ANOTHER ADJACENT LINK STATION ASSOCIATED WITH THE TG TO │
   │                   WHICH THE BTU HAS NOT BEEN OR IS NOT BEING TRANSMITTED, NO ACTION │
   │                   IS TAKEN; OTHERWISE, THE BTU IS INSERTED IN THE │
   │                   TGCB.RETRANSMIT_BTU_LIST.                          │
   │                                                                      │
   │                FOR AN "ABANDONED" CALL:                              │
   │                                                                      │
   │                1. THE TGCB.OUTSTANDING_BTU_CNT IS DECREMENTED BY 1.  │
   │                                                                      │
   │                2. IT IS RECORDED THAT THE TRANSMISSION OF THE BTU TO THE CURRENT │
   │                   ADJACENT LINK STATION HAS COMPLETED.               │
   │                                                                      │
   │                3. IF THE BTU IS NOT IN THE TGCB.RETRANSMIT_BTU_LIST AND IS NOT │
   │                   BEING TRANSMITTED TO ANY OTHER ADJACENT LINK STATION, IT IS │
   │                   DESTROYED.                                         │
   │                                                                      │
   │     INPUT:     THE LSCB_PTR IS ESTABLISHED, THE SEND BTU ASSOCIATED WITH THE BTU │
   │                TRANSMISSION IS POINTED TO BY THE SEND_BTU_PTR CALL PARAMETER, AND │
   │                THE CURRENT STATUS OF THE BTU TRANSMISSION IS IDENTIFIED IN THE │
   │                BTU_TRANSMISSION_STATUS CALL PARAMETER--"TRANSMITTED," │
   │                "ERP_NOTIFICATION," OR "ABANDONED"                    │
   │                                                                      │
   │     OUTPUT:    SEE DESCRIPTION ABOVE                                 │
   │                                                                      │
   │     NOTE:      THE SEND_BTU_PTR IS A POINTER TO A SEND_BTU_PIU_VECTOR_LIST. │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL SEND_BTU_PTR PTR;
  DCL BTU_TRANSMISSION_STATUS CHAR(16);

  /* FUNCTION AS DESCRIBED ABOVE */

  RETURN;

END UPM_PC_TGC_SEND_BTU_MGR;
```

```
PC.TGC.RCV_BTU_CK: PROCEDURE RETURNS(BIT(1));
                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE IS CALLED BY DLC.RCV IN  A SUBAREA NODE WHEN A BLU IS │
│                 RECEIVED FROM A CONTACTED SUBAREA NODE,  TO PERFORM A VALIDITY CHECK │
│                 ON THE BTU PORTION OF THE BLU.                           │
│                                                                          │
│                 THIS  PROCEDURE IS  CALLED  AFTER THE  BLU  HAS SUCCESSFULLY  PASSED │
│                 DLC-LEVEL ERROR CHECKING, BUT BEFORE THE BLU IS ACKNOWLEDGED.  │
│                                                                          │
│                 THIS PC-LEVEL CHECK AUGMENTS DLC-LEVEL ERROR CHECKING MECHANISMS AND │
│                 SERVES  TO  ENHANCE  TRANSMISSION  RELIABILITY.  IN  ADDITION,  IT │
│                 PROVIDES VERIFICATION THAT  SUBSEQUENT PROCESSING OF THE  BTU CAN BE │
│                 SUCCESSFULLY  ACCOMPLISHED, AND  NOT RESULT  IN  THE PROPAGATION  OF │
│                 INVALID DATA.                                            │
│                                                                          │
│                 THE  PRIMARY OBJECTIVE  OF  THIS PROCEDURE  IS  TO  VERIFY THAT  THE │
│                 OVERALL STRUCTURE OF A RECEIVED BTU IS VALID.            │
│                                                                          │
│                 THIS PROCEDURE CHECKS:                                   │
│                                                                          │
│                 1. THAT THE PIU FID VALUES CONTAINED IN A BTU ARE VALID FOR THE TG │
│                                                                          │
│                 2. THAT CERTAIN PIU'S MEET MINIMUM LENGTH REQUIREMENTS   │
│                                                                          │
│                 3. THAT THE LENGTH OF A BTU AS  SPECIFIED BY ITS INTERNAL PIU MAKEUP │
│                    CORRESPONDS TO THE LENGTH OF THE BTU RECEIVED BY DLC  │
│                                                                          │
│                 BECAUSE  OF THE  LOGIC REQUIRED  TO  PERFORM THIS  CHECK, PIU'S  ARE │
│                 CHECKED TO BE  VALID RELATIVE TO FID  AND DCF VALUES AND  THE LENGTH │
│                 AND PLACEMENT OF APPENDED BIU FIELDS, IF ANY.            │
│                                                                          │
│                 THE BTU  DATA ANALYZED BY THIS  PROCEDURE CONSISTS OF PIU'S  IN LINK │
│                 FORM (AS  OPPOSED TO  CANONICAL FORM).  IF THE  SENDING TGC  IN THE │
│                 ADJACENT  NODE  SUPPORTS BLOCKING  (ON  SEND)  FOR  THE TG  AND  THE │
│                 RECEIVING TGC IN THIS NODE SUPPORTS  DEBLOCKING (ON RECEIVE) FOR THE │
│                 TG, THE BTU MAY CONSIST OF MULTIPLE PIU'S.               │
│                                                                          │
│      INPUT:     LSCB_PTR AND BTU_PTR ARE ESTABLISHED,  AND BTUCB.BTU_LENGTH IS EQUAL │
│                 TO THE LENGTH OF THE BTU RECEIVED BY DLC                 │
│                                                                          │
│      OUTPUT:    OK RETURN CODE IF BTU LOGICALLY VALID; OTHERWISE, NG RETURN CODE │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RC BIT(1);

  DCL PIU_TH_LENGTH FIXED(7) BINARY;

  DCL CURRENT_PIU_LENGTH FIXED(15) BINARY;

  DCL REMAINING_BTU_LENGTH FIXED(31) BINARY;

  DCL PIU_FID_PTR PTR;

  DCL PIU_FID BIT(4) BASED(PIU_FID_PTR);

  DCL PIU_MPF_PTR PTR;

  DCL 1 PIU_MPF_BYTE UNALIGNED BASED(PIU_MPF_PTR),

        2 PIU_FID_OR_3_RSVD_BITS_AND_SNAI BIT(4),

        2 PIU_BBIUI BIT(1),

        2 PIU_EBIUI BIT(1),

        2 RESERVED BIT(1),

        2 PIU_EFI BIT(1);

  DCL PIU_DCF_PTR PTR;

  DCL PIU_DCF FIXED(15) BASED(PIU_DCF_PTR);
```

```
TGCB_PTR = LSCB.TGCBPTR;

PIU_FID_PTR = ADDR(BTU_DATA);

REMAINING_BTU_LENGTH = BTUCB.BTU_LENGTH;

RC = OK;

DO UNTIL(RC = NG | REMAINING_BTU_LENGTH = 0);
.
. IF TGCB.ER_VR_SUPP =  ¬PRE_ER_VR THEN
.
.   IF (TGCB.MULTI_LINK_SUPP =  ¬MULTI_LINK_TG & PIU_FID ¬= FID4) |
.        (TGCB.MULTI_LINK_SUPP = MULTI_LINK_TG & PIU_FID ¬= (FID4 | FIDF)) THEN
.     RC = NG;
.
.   ELSE
.     DO;
.     .
.     . PIU_TH_LENGTH = 26;
.     .
.     . PIU_MPF_PTR = PTR_ADD(PIU_FID_PTR,16);                  /* APPENDIX B              */
.     .
.     END;
.
. ELSE
.   IF PIU_FID ¬= (FID0 | FID1) THEN
.     RC = NG;
.
.   ELSE
.     DO;
.     .
.     . PIU_TH_LENGTH = 10;
.     .
.     . PIU_MPF_PTR = PIU_FID_PTR;
.     .
.     END;
.
. IF REMAINING_BTU_LENGTH < PIU_TH_LENGTH THEN
.   RC = NG;
.
. IF RC = OK THEN
.   DO;
.   .
.   . PIU_DCF_PTR = PTR_ADD(PIU_FID_PTR,PIU_TH_LENGTH - 2);    /* APPENDIX B              */
.   .
.   . CURRENT_PIU_LENGTH = PIU_TH_LENGTH + PIU_DCF;
.   .
.   . IF CURRENT_PIU_LENGTH > REMAINING_BTU_LENGTH THEN
.   .   RC = NG;
.   .
.   . ELSE
.   .   DO;
.   .   .
.   .   . SELECT ANYORDER(TGCB.ER_VR_SUPP);
.   .   . .
.   .   . . WHEN(¬PRE_ER_VR)
.   .   . .   IF FID = FID4 &
.   .   . .        PIU_BBIUI = BBIU & PIU_EBIUI =  ¬EBIU & CURRENT_PIU_LENGTH < 36 THEN
.   .   . .     RC = NG;
.   .   . .
.   .   . . WHEN(PRE_ER_VR)
.   .   . .   IF (PIU_BBIUI = BBIU & PIU_EBIUI = EBIU & CURRENT_PIU_LENGTH < 13) |
.   .   . .        (PIU_BBIUI = BBIU & PIU_EBIUI =  ¬EBIU & CURRENT_PIU_LENGTH < 20) THEN
.   .   . .     RC = NG;
.   .   . .
.   .   . END;
.   .   .
.   .   . IF RC = OK THEN
.   .   .   DO;
.   .   .   .
.   .   .   . REMAINING_BTU_LENGTH = REMAINING_BTU_LENGTH - CURRENT_PIU_LENGTH;
.   .   .   .
.   .   .   . IF TGCB.DEBLOCKING_SUPP = DEBLOCKING THEN
.   .   .   .   PIU_FID_PTR = PTR_ADD(PIU_FID_PTR,CURRENT_PIU_LENGTH);  /* APPENDIX B      */
.   .   .   .
.   .   .   . ELSE
.   .   .   .   IF REMAINING_BTU_LENGTH ¬= 0 THEN
.   .   .   .     RC = NG;
.   .   .   .
.   .   .   END;
.   .   END;
.   END;
END;

RETURN(RC);

END PC.TGC.RCV_BTU_CK;
```

This page intentionally left blank

```
PC.TGC.DEQ_Q_BTU_RCV: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE IS DISPATCHED WHEN AN OPEN_QUEUE SIGNAL IS SENT TO IT
   │                 FROM THE HIGHER-LEVEL SCHEDULER.
   │
   │                 THE BTU LOCATED AT THE TOP OF THE TG BTU RECEIVE QUEUE
   │                 (TGCB.Q_BTU_RCV) IS REMOVED FROM THE QUEUE AND SENT TO PC.TGC.RCV.
   │
   │      INPUT:     AN OPEN_QUEUE SIGNAL FROM HIGHER-LEVEL SCHEDULER, WITH TGCB_PTR
   │                 ESTABLISHED
   │
   │      OUTPUT:    A BTU IS SENT TO PC.TGC.RCV; THIS IS DONE BY SENDING A "BTU" SIGNAL
   │                 TO PC.TGC.RCV WITH THE PARM_PTR POINTING TO THE BTU.
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  LOCK TGCB.Q_BTU_RCV;
  .
  .   REMOVE FIRST(BTU) FROM TGCB.Q_BTU_RCV SET(BTU_PTR);
  .
  UNLOCK;

  SEND 'BTU' TO PC.TGC.RCV USING(PARM_PTR=BTU_PTR);          /* PAGE 3-40   */

  RETURN;

END PC.TGC.DEQ_Q_BTU_RCV;
```

```
PC.TGC.RCV: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS  PROCEDURE  IS  DISPATCHED  WHEN  A BTU  IS  SENT  TO  IT  FROM  │
│                 PC.TGC.DEQ_Q_BTU_RCV.                                    │
│                                                                          │
│                 IF DEBLOCKING IS  SUPPORTED AND REQUIRED, THE BTU  IS DEBLOCKED INTO  │
│                 INDIVIDUAL PIU'S (SEE NOTE 1).  THE PIU'S ARE PROCESSED AS FOLLOWS.  │
│                                                                          │
│                 IF TG TRACE IS ACTIVE FOR THE TG, A TRACE OF EACH PIU IS PROVIDED.  │
│                                                                          │
│                 IF THE TG IS A SINGLE-LINK TG AND THE ADJACENT SUBAREA NODE SUPPORTS  │
│                 ER AND VR PROTOCOLS, THE PIU'S ARE SENT TO PC.ERC.       │
│                                                                          │
│                 IF THE TG IS A SINGLE-LINK TG AND THE ADJACENT SUBAREA NODE DOES NOT  │
│                 SUPPORT ER  AND VR  PROTOCOLS, THE PIU'S  ARE CONVERTED  FROM EITHER  │
│                 FID1 OR FID0 TO FID4 AND SENT TO PC.ERC.                 │
│                                                                          │
│                 IF THE TG  IS A MULTIPLE-LINK TG (ADJACENT SUBAREA  NODE SUPPORTS ER  │
│                 AND VR PROTOCOLS), THEN:                                 │
│                                                                          │
│                 1. FID4 PIU'S WITH TG_NONFIFO_IND=NON_FIFO ARE SENT TO PC.ERC.  │
│                                                                          │
│                 2. FID4  PIU'S  WITH  TG_NONFIFO_IND=FIFO  ARE  SENT  TO  PC.ERC  IN  │
│                    SEQUENCE AND WITHOUT  DUPLICATION; THIS IS DONE  BY: CHECKING THE  │
│                    PIU  TG_SNF  VALUES  AGAINST  THE  TGCB.TG_SNF_RCV_CNTR,  SENDING  │
│                    IN-SEQUENCE  PIU'S TO  PC.ERC,  DISCARDING  DUPLICATE PIU'S,  AND  │
│                    HOLDING    NONDUPLICATE,    OUT-OF-SEQUENCE    PIU'S    IN    THE  │
│                    TGCB.REFIFO_PIU_LIST  UNTIL  THEY  CAN  BE  SENT  TO  PC.ERC  IN  │
│                    SEQUENCE.   IN CONJUNCTION  WITH THIS  PROCESSING,  WHEN ALL  THE  │
│                    PIU'S IN  A GROUP  OF TG-SEQUENCED  PIU'S (TG_SNF'S  0-4095) HAVE  │
│                    BEEN  RECEIVED AND  PASSED TO  ERC,  THE TRANSMISSION  OF A  FIDF  │
│                    TG_SNF_WRAP_ACK PIU TO THE ADJACENT SUBAREA NODE IS INITIATED.  │
│                                                                          │
│                 3. FIDF   TG_SNF_WRAP_ACK   PIU'S   RECEIVED   IN   SEQUENCE,   WHEN  │
│                    FSM_SUSPEND_TG_SEND IS IN  A SUSPEND STATE, RESULT IN  A RESET TO  │
│                    FSM_SUSPEND_TG_SEND.  ALL FIDF PIU'S ARE DISCARDED.  │
│                                                                          │
│                 FOR  FID4  PIU'S, THE  VR_RWI  IN  THE  TH  IS SET  TO  RESET_WS  IF  │
│                 FSM_VR_WINDOW_SIZE INDICATES SEVERE_CONGESTION.          │
│                                                                          │
│                 WHEN  ALL THE  PIU'S  IN  A BTU  HAVE  BEEN  PROCESSED, THE  BTU  IS  │
│                 DISCARDED.                                               │
│                                                                          │
│      INPUT:     A "BTU" SIGNAL  FROM PC.TGC.DEQ_Q_BTU_RCV WITH PARM_PTR  POINTING TO  │
│                 BTU, AND LSCB_PTR AND TGCB_PTR ESTABLISHED               │
│                                                                          │
│      OUTPUT:    RECEIVED  PIU'S  SENT TO  PC.ERC  (SEE  NOTE  2) OR  DISCARDED,  FOR  │
│                 MULTIPLE-LINK TG'S A FIDF TG_SNF-WRAP_ACK PIU MAY BE INSERTED AT TOP  │
│                 OF TGCB.PRTY_SEND_PIU_LIST                               │
│                                                                          │
│      NOTES:     1. A RECEIVED BTU IS  PASSED FROM DLC.RCV TO PC.TGC AS  A BTU ENTITY  │
│                    CONSISTING OF  A BTU CONTROL BLOCK  AND THE BTU ITSELF.  THE BTU  │
│                    CONSIST OF ONE  OR, IF BLOCKING IS PERFORMED BY  THE SENDING TGC,  │
│                    ONE  OR  MORE PIU'S  IN  LINK  FORM.   THIS PROCEDURE  CALLS  THE  │
│                    MAP_TO_CANONICAL PROCEDURE (SEE APPENDIX B)  TO MAP EACH PIU INTO  │
│                    A CANONICAL PIU;  THIS CONVERTS THE PIU INTO  CANONICAL FORM (FOR  │
│                    ARCHITECTURAL  DESCRIPTION  PURPOSES)  AND ESTABLISHES  IT  AS  A  │
│                    SEPARATE ENTITY.                                      │
│                                                                          │
│                 2. FOR  MULTIPLE-LINK   TG'S,  NONDUPLICATE,   OUT-OF-SEQUENCE  FID4  │
│                    TG-FIFO PIU'S  ARE TEMPORARILY  HELD IN  THE TGCB.REFIFO_PIU_LIST  │
│                    UNTIL THEY CAN BE PASSED TO PC.ERC IN SEQUENCE.       │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 CONVERT_FID1_OR_FID0_TO_FID4      PAGE 3-43              │
│                 FSM_SUSPEND_TG_SEND               PAGE 3-46              │
│                 FSM_VR_WINDOW_SIZE                PAGE 3-47              │
│                 LENGTH_OF_PIU                     PAGE 3-44              │
│                 LOG_ERROR_AND_DISCARD_PIU         PAGE 3-101             │
│                 SEND_TG_SNF_WRAP_ACK              PAGE 3-42              │
│                 UPM_TG_TRACE                      PAGE 3-45              │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

DCL LINK_PIU_PTR PTR;
DCL PIU_LENGTH FIXED(15) BINARY;
DCL PROCESSED_BYTE_CNT FIXED(31) BINARY;
DCL REFIFO_PIU_PTR PTR;

BTU_PTR = PARM_PTR;
LINK_PIU_PTR = ADDR(BTU_DATA);
PROCESSED_BYTE_CNT = 0;

DO UNTIL(PROCESSED_BYTE_CNT = BTUCB.BTU_LENGTH);
.
. CREATE MU;                          /* MU IS DEFINED IN APPENDIX C      */
.
. CALL MAP_TO_CANONICAL(LINK_PIU_PTR,MU_PTR);          /* APPENDIX B      */
.
. PIU_LENGTH = LENGTH_OF_PIU;                          /* PAGE 3-44       */
.
. IF TGCB.TG_TRACE = TRACE THEN
.   CALL UPM_TG_TRACE('RCV');                          /* PAGE 3-45       */
```

```
. IF TGCB.ER_VR_SUPP = PRE_ER_VR THEN
.   DO;
.   . CALL CONVERT_FID1_OR_FID0_TO_FID4;                    /* PAGE 3-43          */
.   .
.   . SEND MU TO PC.ERC;                                    /* PAGE 3-50          */
.   END;
.
. ELSE
.   DO;
.   . IF FID = FID4 & FSM_VR_WINDOW_SIZE = SEVERE_CONGESTION THEN /* PAGE 3-47     */
.   .   VR_RWI = RESET_WS;
.   .
.   . IF TGCB.MULTI_LINK_SUPP =  ¬MULTI_LINK_TG THEN
.   .   SEND MU TO PC.ERC;                                  /* PAGE 3-50          */
.   .
.   . ELSE
.   .   IF FID = FID4 THEN
.   .
.   .     IF TG_NONFIFO_IND = NON_FIFO THEN
.   .       SEND MU TO PC.ERC;                              /* PAGE 3-50          */
.   .
.   .     ELSE
.   .       SELECT ANYORDER;
.   .       .
.   .       . WHEN(TG_SNF = TGCB.TG_SNF_RCV_CNTR)
.   .       .   DO;
.   .       .   . SEND MU TO PC.ERC;                        /* PAGE 3-50          */
.   .       .   .
.   .       .   . IF TG_SNF = 4095 THEN
.   .       .   .   CALL SEND_TG_SNF_WRAP_ACK;              /* PAGE 3-42          */
.   .       .   .
.   .       .   . TGCB.TG_SNF_RCV_CNTR = TGCB.TG_SNF_RCV_CNTR + 1;
.   .       .   .
.   .       .   . DO WHILE( ¬EMPTY(TGCB.REFIFO_PIU_LIST) &
.   .       .   .       FIRST_ENTRY(TGCB.REFIFO_PIU_LIST)->TG_SNF = TGCB.TG_SNF_RCV_CNTR);
.   .       .   . . REMOVE FIRST(MU) FROM TGCB.REFIFO_PIU_LIST SET(MU_PTR);
.   .       .   . .
.   .       .   . . SEND MU TO PC.ERC;                      /* PAGE 3-50          */
.   .       .   . .
.   .       .   . . IF TG_SNF = 4095 THEN
.   .       .   . .   CALL SEND_TG_SNF_WRAP_ACK;            /* PAGE 3-42          */
.   .       .   . .
.   .       .   . . TGCB.TG_SNF_RCV_CNTR = TGCB.TG_SNF_RCV_CNTR + 1;
.   .       .   . END;
.   .       .   END;
.   .       .
.   .       . WHEN(TG_SNF > TGCB.TG_SNF_RCV_CNTR)
.   .       .   DO;
.   .       .   . IF TGCB.TG_SNF_RCV_CNTR = 0 THEN
.   .       .   .   DISCARD MU;
.   .       .   .
.   .       .   . ELSE
.   .       .   .   DO;
.   .       .   .   . FIND REFIFO_PIU_PTR->MU IN TGCB.REFIFO_PIU_LIST
.   .       .   .   .       WHERE(TG_SNF = REFIFO_PIU_PTR->TG_SNF);
.   .       .   .   .
.   .       .   .   . IF REFIFO_PIU_PTR = NULL THEN
.   .       .   .   .   INSERT MU BY_ASCENDING(TG_SNF) IN TGCB.REFIFO_PIU_LIST;
.   .       .   .   .
.   .       .   .   . ELSE
.   .       .   .   .   DISCARD MU;
.   .       .   .   END;
.   .       .   END;
.   .       .
.   .       . WHEN(TG_SNF < TGCB.TG_SNF_RCV_CNTR)
.   .       .   DISCARD MU;
.   .       END;
.   .
.   .   ELSE                                                /* FIDF               */
.   .     IF CMD_FORMAT = TG_CMD & CMD_TYPE = TG_SNF_WRAP_ACK & DCF = 0 THEN
.   .       DO;
.   .       . IF FSM_SUSPEND_TG_SEND = SUSPEND &            /* PAGE 3-46          */
.   .       .      CMD_SEQ_NUM = TGCB.TG_SNF_WRAP_ACK_RCV_CNTR THEN
.   .       .   DO;
.   .       .   . CALL FSM_SUSPEND_TG_SEND('RESET');        /* PAGE 3-46          */
.   .       .   .
.   .       .   . TGCB.TG_SNF_WRAP_ACK_RCV_CNTR = TGCB.TG_SNF_WRAP_ACK_RCV_CNTR + 1;
.   .       .   END;
.   .       .
.   .       . DISCARD MU;
.   .       END;
.   .
.   .     ELSE
.   .       CALL LOG_ERROR_AND_DISCARD_PIU('INVALID FIDF PIU'); /* PAGE 3-101     */
.   END;
.
. LINK_PIU_PTR = PTR_ADD(LINK_PIU_PTR,PIU_LENGTH);          /* APPENDIX B         */
.
. PROCESSED_BYTE_CNT = PROCESSED_BYTE_CNT + PIU_LENGTH;
END;

DISCARD BTU;

RETURN;

END PC.TGC.RCV;
```

```
SEND_TG_SNF_WRAP_ACK: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS  PROCEDURE  IS  CALLED  WHEN  ALL  THE  PIU'S  IN  A  GROUP  OF  │
│                 TG-SEQUENCED PIU'S  (TG_SNF VALUES  0-4095) HAVE  BEEN RECEIVED  AND  │
│                 PASSED   TO  ERC,   TO   INITIATE  THE   TRANSMISSION   OF  A  FIDF  │
│                 TG_SNF_WRAP_ACK PIU TO THE ADJACENT SUBAREA NODE.                    │
│                                                                          │
│                 THIS PROCEDURE:                                          │
│                                                                          │
│                 1. BUILDS A FIDF TG_SNF_WRAP_ACK PIU  WITH THE CMD_SEQ_NUM FIELD SET │
│                    EQUAL TO THE TGCB.SNF_WRAP_ACK_SEND_CNTR              │
│                                                                          │
│                 2. ASSIGNS THE PIU A MUCB.TG_SEND_PRTY VALUE OF PRTY_1; THIS IS DONE │
│                    SO  THAT THE  PIU WILL  BE COMPATIBLE  WITH AND  NOT DISRUPT  THE  │
│                    OPERATION  OF  THE  TGCB.PRTY_SEND_PIU_LIST, WHICH  IS  KEYED  BY  │
│                    MUCB.TG_SEND_PRTY                                     │
│                                                                          │
│                 3. INSERTS THE PIU AT THE TOP OF THE TGCB.PRTY_SEND_PIU_LIST         │
│                                                                          │
│                 4. INCREMENTS THE TGCB.SNF_WRAP_ACK_SEND_CNTR BY 1       │
│                                                                          │
│                 THIS FIDF  TG_SNF_WRAP_ACK PIU WILL BE  THE NEXT PIU  TRANSMITTED TO │
│                 THE ADJACENT NODE FROM THE TGCB.PRTY_SEND_PIU_LIST.      │
│                                                                          │
│      INPUT:     NONE                                                     │
│                                                                          │
│      OUTPUT:    FIDF TG_SNF_WRAP_ACK PIU INSERTED  AT TOP OF TGCB.PRTY_SEND_PIU_LIST  │
│                 AND TGCB.TG_WRAP_ACK_SEND_CNTR INCREMENTED BY 1          │
│                                                                          │
│      NOTE:      ALL FIELDS IN THE MU (DEFINED IN APPENDIX C) ARE SET TO 0 WHEN IT IS  │
│                 CREATED.                                                 │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 PC.TGC.RCV                          PAGE 3-40            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL PIU_PTR_SAVE PTR;

  PIU_PTR_SAVE = MU_PTR;

  CREATE MU;                                                /* NOTE      */

  FID = FIDF;

  CMD_FORMAT = TG_CMD;

  CMD_TYPE = TG_SNF_WRAP_ACK;

  CMD_SEQ_NUM = TGCB.TG_SNF_WRAP_ACK_SEND_CNTR;

  DCF = 0;

  MUCB.TG_SEND_PRTY = PRTY_1;

  LOCK TGCB.PRTY_SEND_PIU_LIST;
  .
  . INSERT MU FIRST IN TGCB.PRTY_SEND_PIU_LIST;
  .
  UNLOCK;

  TGCB.TG_SNF_WRAP_ACK_SEND_CNTR = TGCB.TG_SNF_WRAP_ACK_SEND_CNTR + 1;

  MU_PTR = PIU_PTR_SAVE;

  RETURN;

END SEND_TG_SNF_WRAP_ACK;
```

```
CONVERT_FID1_OR_FID0_TO_FID4: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  THIS PROCEDURE  IS CALLED WHEN  A PIU  IS RECEIVED FROM  AN ADJACENT │
│                SUBAREA NODE THAT  DOES NOT SUPPORT ER AND VR  PROTOCOLS, TO CONVERT │
│                THE PIU TH FROM FID1 OR FID0 TO FID4.  SEE NOTE.          │
│                                                                          │
│                IF THE PIU  TH IS FID1, THE SNA  INDICATOR (SNAI) IN THE  FID4 TH IS │
│                SET TO SNA.                                               │
│                                                                          │
│                IF THE PIU  TH IS FID0, THE SNA  INDICATOR (SNAI) IN THE  FID4 TH IS │
│                SET TO ¬SNA.                                              │
│                                                                          │
│     INPUT:     CURRENT (FID1 OR FID0) PIU; POINTED TO BY MU_PTR          │
│                                                                          │
│     OUTPUT:    FID4 PIU; POINTED TO BY MU_PTR                            │
│                                                                          │
│     NOTE:      THIS PROCEDURE SETS:-                                     │
│                                                                          │
│                • FID4 TH FIELDS COMMON TO FID1|FID0 THAT REQUIRE A CHANGE IN VALUE, │
│                  I.E., FID                                               │
│                                                                          │
│                • FID4 TH FIELDS NOT COMMON TO FID1|FID0 THAT REQUIRE EITHER A VALUE │
│                  DERIVED FROM THE  FID1|FID0 TH, I.E., DSAF, OSAF,  SNAI, DEF, OEF; │
│                  OR A ¬0 VALUE, I.E., ER_VR_SUPP_IND, TG_NONFIFO_IND,    │
│                                                                          │
│                THIS PROCEDURE DOES NOT SET:                              │
│                                                                          │
│                • FID4 TH FIELDS COMMON TO FID1|FID0 THAT DO NOT REQUIRE A CHANGE IN │
│                  VALUE, I.E., MPF, EFI, SNF, DCF                         │
│                                                                          │
│                • FID4 TH FIELDS  NOT COMMON TO FID1|FID0  THAT ARE TO BE  SET TO 0, │
│                  I.E., TG_SWEEP,  VR_PAC_CNT_IND, NTWK_PRTY,  IERN, ERN,  VRN, TPF, │
│                  VR_CWI, VR_SQTI, TG_SNF, VRPRQ, VRPRS, VR_CWI, VR_RWI, VR_SNF │
│                                                                          │
│                ALL THE TH  FIELDS NOT EXPLICITLY SET WERE  SET CORRECTLY--EITHER TO │
│                THE FID1|FID0  VALUE OR TO  0, AS REQUIRED--BY  THE MAP_TO_CANONICAL │
│                PROCEDURE (SEE  APPENDIX B), WHICH WAS  CALLED BY PC.TGC.RCV  TO MAP │
│                THE PIU FROM LINK FORM TO CANONICAL FORM.                 │
│                                                                          │
│                POSITIONAL  CHANGES IN  TH FIELDS  BETWEEN  FID1|FID0 AND  FID4--ALL │
│                FIELDS EXCEPT  FID--ARE HANDLED BY THE  MAP_FROM_CANONICAL PROCEDURE │
│                (SEE  APPENDIX  B),  WHICH  IS  CALLED  BY  PC.TGC.SEND  USING  THE │
│                ADD_PIU_TO_BTU PROCEDURE TO MAP THE PIU  FROM CANONICAL FORM TO LINK │
│                FORM, IF THE  PIU IS SUBSEQUENTLY TRANSMITTED  OUT OF THE NODE  TO A │
│                SUBAREA NODE THAT SUPPORTS ER AND VR PROTOCOLS.           │
│                                                                          │
│                THE FID1, FID0, AND FID4 TH FORMATS ARE DESCRIBED IN CHAPTER 2. │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                PC.TGC.RCV                          PAGE 3-40             │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  FID = FID4;

  ER_VR_SUPP_IND = PRE_ER_VR;


  TG_NONFIFO_IND = FIFO;


  DSAF(32 - NCB.SUBAREA_LEN:31) = DAF(0:NCB.SUBAREA_LEN - 1);


  OSAF(32 - NCB.SUBAREA_LEN:31) = OAF(0:NCB.SUBAREA_LEN - 1);


  IF FID = FID1 THEN
    SNAI = SNA;

  ELSE
    SNAI =  ¬SNA;


  DEF(NCB.SUBAREA_LEN:15) = DAF(NCB.SUBAREA_LEN:15);


  OEF(NCB.SUBAREA_LEN:15) = OAF(NCB.SUBAREA_LEN:15);


  RETURN;

END CONVERT_FID1_OR_FID0_TO_FID4;
```

```
LENGTH_OF_PIU: PROCEDURE RETURNS(FIXED BINARY(15));

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE IS  INVOKED BY A FUNCTION REFERENCE  TO CALCULATE THE
 │                 LENGTH OF A FID0, FID1, FID4, OR FIDF PIU.           │
 │                                                                      │
 │                 THE PIU  LENGTH IS CALCULATED BY  ADDING TOGETHER THE LENGTH  OF THE
 │                 PIU TH AND THE PIU DCF VALUE.                        │
 │                                                                      │
 │                 FOR FID4 AND FIDF PIU'S, THE TH LENGTH IS 26.        │
 │                                                                      │
 │                 FOR FID1 AND FID0 PIU'S, THE TH LENGTH IS 10.        │
 │                                                                      │
 │                 THE CALCULATED PIU LENGTH IS RETURNED TO THE FUNCTION REFERENCE.
 │                                                                      │
 │      INPUT:     CURRENT PIU; POINTED TO BY MU_PTR                    │
 │                                                                      │
 │      OUTPUT:    RETURN PARAMETER IS SET EQUAL TO CALCULATED LENGTH OF CURRENT PIU
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 ADD_PIU_TO_BTU                    PAGE 3-33          │
 │                 PC.TGC.LIST_BY_PRTY               PAGE 3-23          │
 │                 PC.TGC.RCV                        PAGE 3-40          │
 │                 SINGLE_LINK_TG_SEND               PAGE 3-29          │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL CALCULATED_PIU_LENGTH FIXED(15) BINARY;

  SELECT ANYORDER(FID);
  .
  . WHEN(FID4 | FIDF)
  .   CALCULATED_PIU_LENGTH = DCF + 26;
  .
  . WHEN(FID1 | FID0)
  .   CALCULATED_PIU_LENGTH = DCF + 10;
  .
  END;

  RETURN(CALCULATED_PIU_LENGTH);

END LENGTH_OF_PIU;
```

```
UPM_TG_TRACE: PROCEDURE(DIRECTION);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  THIS IMPLEMENTATION-DEPENDENT PROCEDURE IS CALLED WHEN TGCB.TG_TRACE
   │                IS  ACTIVE  AND  A  PIU  IS  EITHER  INSERTED  INTO  THE
   │                TGCB.PRTY_SEND_PIU_LIST  OR PROCESSED  BY PC.TGC.RCV,  TO PROVIDE  A
   │                TRACE OF THE SEND AND RECEIVED PIU TRAFFIC OVER A TG.
   │
   │                THIS  PROCEDURE  IS  CALLED  WITH  A  "SEND"  PARAMETER  BY
   │                PC.TGC.LIST_BY_PRTY  WHEN  TGCB.TG_TRACE  IS ACTIVE  AND  A  PIU  IS
   │                INSERTED INTO THE TGCB.PRTY_SEND_PIU_LIST.
   │
   │                THIS PROCEDURE IS  CALLED WITH A "RCV" PARAMETER  BY PC.TGC.RCV WHEN
   │                TGCB.TG_TRACE IS ACTIVE AND A PIU IS RECEIVED FROM DLC.RCV.
   │
   │                TYPICALLY, THE TRACE  IS PRESENTED AS SEND AND  RECEIVED PIU'S.  FOR
   │                SEND PIU'S, THE TG IS IDENTIFIED.  FOR RECEIVED PIU'S, THE TG, LINK,
   │                AND ADJACENT LINK STATION ASSOCIATED WITH EACH PIU ARE IDENTIFIED.
   │
   │     INPUT:     EITHER "SEND" OR  "RCV" CALL PARAMETER WITH MU_PTR  POINTING PIU AND
   │                TGCB.PTR ESTABLISHED; LSCB_PTR IS ESTABLISHED FOR "RCV" CALL.
   │
   │     OUTPUT:    TRACE OF SEND AND RECEIVED PIU'S FOR TG
   │
   │     NOTE:      THE SEND TRACE IS A TRACE OF PIU'S RECEIVED BY THE SEND COMPONENT OF
   │                TGC FOR THE TG,  TO BE TRANSMITTED OVER THE TG.   IT SHOWS THE ORDER
   │                IN WHICH THE  PIU'S ARRIVE AT THE SEND  SIDE OF THE TG;  IT DOES NOT
   │                SHOW THE ORDER IN WHICH THE PIU'S ARE TRANSMITTED.  THE TRANSMISSION
   │                OF   FIDF  TG_SNF_WRAP_ACK   PIU'S  AND   PIU'S   INCLUDED  IN   BTU
   │                RETRANSMISSIONS ARE NOT INCLUDED IN THE SEND TRACE.
   │
   │                THE  RECEIVE  TRACE  IS  A  TRACE  OF  ALL  PIU'S RECEIVED  OVER  THE
   │                TG--PIU'S  INCLUDED  IN  BTU'S  THAT ARE  REJECTED  BECAUSE  OF  THE
   │                PC.TGC.RCV_BTU_CK ARE  NOT CONSIDERED TO BE  RECEIVED BY THE  TG AND
   │                NOT INCLUDED IN  THE RECEIVE TRACE.  THE RECEIVE  TRACE REFLECTS THE
   │                ORDER  IN   WHICH  THE   PIU'S  ARE   RECEIVED  AND   INCLUDES  FIDF
   │                TG_SNF_WRAP_ACK PIU'S AND DUPLICATE PIU'S THAT MAY RESULT BECAUSE OF
   │                BTU RETRANSMISSION.
   │
   │                FOR MULTIPLE-LINK  TG'S, ALL THE  PIU'S RECEIVED BY  PC.TGC.RCV FROM
   │                DLC.RCV AND SHOWN IN A TG  RECEIVE TRACE ARE NOT NECESSARILY PASSED,
   │                OR PASSED IN  ORDER OF ARRIVAL, TO ERC.  THIS  IS BECAUSE PC.TGC.RCV
   │                DISCARDS  FIDF PIU'S  AND  DUPLICATE  TG-SEQUENCED FID4  PIU'S,  AND
   │                ENFORCES THE SEQUENTIAL DELIVERY OF TG-SEQUENCED FID4 PIU'S TO ERC.
   │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                PC.TGC.LIST_BY_PRTY                PAGE 3-23
   │                PC.TGC.RCV                         PAGE 3-40
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL DIRECTION CHAR(4);

  /* FUNCTION AS DESCRIBED ABOVE */

  RETURN;

END UPM_TG_TRACE;
```

```
FSM_TG_SWEEP: FSM_DEFINITION CONTEXT(TGCB);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS FSM IS USED TO CONTROL THE TG SWEEP FUNCTION FOR MULTIPLE-LINK  │
   │                 TG'S.                                                │
   │                                                                      │
   │                 A TG IS IN A SWEPT STATE WHEN NO BTU TRANSMISSIONS ARE IN PROGRESS  │
   │                 FOR THE TG--THE TGCB.OUTSTANDING_BTU_CNT=0. ACCORDINGLY, A TG SWEEP  │
   │                 COMPLETES WHEN THE TG IS IN A SWEEP STATE AND THE    │
   │                 TGCB.OUTSTANDING_BTU_CNT BECOMES 0.                  │
   │                                                                      │
   │                 THE SWEEP_BIT_PRE_SWEEP STATE IS SET BEFORE A FID4 PIU WITH  │
   │                 TG_SWEEP=SWEEP IS TRANSMITTED, IF THE TG IS NOT IN A SWEPT STATE.  │
   │                                                                      │
   │                 THE SNF_0_PRE_SWEEP STATE IS SET BEFORE A FID4 PIU WITH  │
   │                 TG_NONFIFO_IND=FIFO AND TG_SNF=0 IS TRANSMITTED, IF THE TG IS NOT IN  │
   │                 A SWEPT STATE.                                       │
   │                                                                      │
   │                 THE SNF_0_POST_SWEEP STATE IS SET WHEN A FID4 PIU WITH  │
   │                 TG_NONFIFO_IND=FIFO AND TG_SNF=0 IS TRANSMITTED; THIS STATE MAY BE  │
   │                 SET DIRECTLY FROM THE RESET STATE OR RESULT FROM A "SWEEP_COMPLETE"  │
   │                 INPUT SIGNAL WITH THE FSM IN THE SNF_0_PRE_SWEEP STATE.  │
   │                                                                      │
   │                 "SWEEP_COMPLETE" IS SIGNALED TO THE FSM WHEN IT IS IN ANY OF THE  │
   │                 SWEEP STATES AND A TG SWEEP COMPLETES.               │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 MULTI_LINK_TG_SEND                PAGE 3-30          │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES---> / INPUTS	RESET 01	SWEEP_BIT_PRE_SWEEP 02	SNF_0_PRE_SWEEP 03	SNF_0_POST_SWEEP 04
'SWEEP_BIT_PRE_SWEEP'	2	/	/	/
'SNF_0_PRE_SWEEP'	3	/	/	/
'SNF_0_POST_SWEEP'	4	/	/	/
'SWEEP_COMPLETE'	/	1	4	1
'RESET'	-	1	1	1

```
END FSM_TG_SWEEP;


FSM_SUSPEND_TG_SEND: FSM_DEFINITION CONTEXT(TGCB);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS FSM INDICATES WHEN THE TRANSMISSION OF FID4 PIU'S OVER A  │
   │                 MULTIPLE-LINK TG IS SUSPENDED, PENDING RECEIPT OF A PROPER FIDF  │
   │                 TG_SNF_WRAP_ACK PIU FROM THE ADJACENT NODE.          │
   │                                                                      │
   │                 THIS FSM IS USED TO SUSPEND ONLY THE TRANSMISSION OF FID4 PIU'S THAT  │
   │                 ARE IN THE TGCB.PRTY_SEND_PIU_LIST--FID4 PIU'S THAT HAVE NOT BEEN  │
   │                 PREVIOUSLY TRANSMITTED; IT IS NOT USED TO PREVENT THE RETRANSMISSION  │
   │                 OF FID4 PIU'S THAT MAY OCCUR AS A RESULT OF TGC BTU RETRANSMISSION  │
   │                 OR THE TRANSMISSION OF A FIDF TG_SNF_WRAP_ACK PIU.   │
   │                                                                      │
   │                 THE FSM IS SET TO THE SUSPEND STATE WHEN THE LAST PIU IN A GROUP OF  │
   │                 TG-SEQUENCED PIU'S IS TRANSMITTED; THAT IS, WHEN A FID4 PIU WITH  │
   │                 TG_NONFIFO_IND=FIFO AND TG_SNF=4095 IS TRANSMITTED.  │
   │                                                                      │
   │                 THE FSM IS RESET WHEN IT IS IN THE SUSPEND STATE AND A FIDF  │
   │                 TG_SNF_WRAP_ACK PIU WITH THE CMD_SEQ_NUM FIELD EQUAL TO THE  │
   │                 TGCB.TG_SNF_WRAP_ACK_RCV_CNTR IS RECEIVED FROM THE ADJACENT NODE.  │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 MULTI_LINK_TG_SEND                PAGE 3-30          │
   │                 PC.TGC.RCV                        PAGE 3-40          │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES---> / INPUTS	RESET 01	SUSPEND 02
'SUSPEND'	2	/
'RESET'	-	1

```
END FSM_SUSPEND_TG_SEND;
```

```
FSM_VR_WINDOW_SIZE: FSM_DEFINITION CONTEXT(TGCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  THIS FSM IS USED BY TGC TO DETERMINE WHETHER TO:          │
│                                                                          │
│                1. SET THE VIRTUAL ROUTE CHANGE  WINDOW INDICATOR (VR_CWI) TO DEC_WS
│                   (DECREMENT  WINDOW SIZE)  IN  EACH FID4  PIU  TRANSMITTED OVER  A
│                   TG--THIS IS DONE WHEN MODERATE DATA TRAFFIC CONGESTION EXISTS, TO
│                   CAUSE A  GRADUAL DECREASE  OF THE VIRTUAL  ROUTE WINDOW  SIZE FOR
│                   PIUS FLOWING IN  THE SAME DIRECTION AS THE  PIU BEING TRANSMITTED
│                   BY TGC.                                                │
│                                                                          │
│                2. SET THE VIRTUAL ROUTE RESET WINDOW INDICATOR (VR_RWI) TO RESET_WS
│                   (RESET WINDOW SIZE) IN EACH FID4  PIU RECEIVED OVER A TG--THIS IS
│                   DONE WHEN  SEVERE DATA  TRAFFIC CONGESTION  EXISTS, TO  RESET THE
│                   VIRTUAL ROUTE  WINDOW SIZE TO THE  MINIMUM SIZE SPECIFIED  IN THE
│                   NC_ACTVR RU FOR  PIU'S FLOWING IN THE DIRECTION  OPPOSITE TO THAT
│                   OF THE PIU BEING RECEIVED BY TGC.                      │
│                                                                          │
│                THE DETERMINATION OF  WHEN MODERATE CONGESTION OR  SEVERE CONGESTION
│                EXISTS AND THE  SETTING OF THESE STATES FOR A  TG ARE IMPLEMENTATION
│                DEPENDENT.                                                │
│                                                                          │
│                THE    INPUT    SIGNALS    TO    THIS    FSM    ARE    SENT    BY    AN
│                IMPLEMENTATION-DEPENDENT UPM.                             │
│                                                                          │
│                THE VIRTUAL  ROUTE WINDOW SIZE  AND THE  FUNCTION OF THE  VR_CWI AND
│                VR_RWI  BITS IN  THE  FID4 TH  ARE DESCRIBED  IN  THE DISCUSSION  ON
│                "VIRTUAL ROUTE  PACING" IN  THE "VIRTUAL  ROUTE CONTROL"  SECTION OF
│                THIS CHAPTER.                                             │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                ADD_PIU_TO_BTU                    PAGE 3-33               │
│                PC.TGC.RCV                        PAGE 3-40               │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES-----> / INPUTS	RESET 01	MODERATE_CONGESTION 02	SEVERE_CONGESTION 03
'MODERATE_CONGESTION'	2	-	2
'SEVERE_CONGESTION'	3	3	-
'RESET'	-	1	1

```
END FSM_VR_WINDOW_SIZE;
```

EXPLICIT ROUTE CONTROL

This section discusses the routing controls provided to send message units between subareas over explicit routes.

An explicit route (ER) is a bidirectional logical connection between two subareas, and can be denoted by the quadruple, (SA1, SA2, ERN, RERN), where:

- SA1 and SA2 are the subarea addresses of the two subarea nodes at the ends of the ER. SA2 is the DSAF value used for PIUs that originate at SA1 and are destined for SA2. SA1 is the DSAF value used for PIUs that originate at SA2 and are destined for SA1.

- ERN is the ER number used for PIUs that originate at subarea SA1 and are destined for subarea SA2. RERN (or reverse ERN) is the ER number used for PIUs that originate at SA2 and are destined for SA1.

An ER includes one or more transmission groups (TGs) that provide serial connectivity between the subareas at the ends of the ER--each TG can be denoted by a triple, (SAi, TGN, SAj). If there are no intermediate subarea nodes along an ER, the ER includes only one TG. If there are intermediate subarea nodes along an ER, the number of TGs included in the ER is one greater than the number of intermediate subarea nodes along the ER. When an ER includes more that one TG, the order in which the TGs are traversed for a direction of transmission on the ER is the reverse of the order for the opposite direction. Within subarea nodes, routing tables that contain ER to TG mapping information are used to route PIUs over ERs.

The activation and deactivation of ERs and the association of ERs with virtual routes (VRs) are described in Chapter 12.

The routing of PIUs over ERs is implemented in subarea nodes in explicit route control (ERC). ERC is a component of subarea routing path control (PC_SA); it is positioned between virtual route control and transmission group control--see Figure 3-3 on page 3-7.

As shown in Figure 3-3, ERC routes PIUs received from the virtual route manager (Chapter 12), virtual route control, transmission group control, and, optionally, an undefined protocol machine that may initiate the transmission of non-SNA PIUs.

ERC performs PIU routing as follows:

- Non-SNA PIUs destined for the local subarea are sent to an undefined protocol machine that processes non-SNA PIUs received at the node (non-SNA PIUs always have the SNA indicator (SNAI) set to ¬SNA and the Explicit Route Number (ERN) field set to 0).

- PU-PU flow PIUs destined for the local subarea are sent to the path control route manager component of the PU services manager.

- All other PIUs destined for the local subarea are sent to virtual route control.

- For PIUs destined for other subareas, ERC establishes the transmission group over which the PIU is to be transmitted--transmission group routing--and sends the PIU to transmission group control. ERC uses the DSAF and ERN in the PIU transmission header, the SUBAREA_ROUTING_LIST (Appendix A), and the TGCB_LIST (Appendix A) to establish transmission group routing. If transmission group routing cannot be established, an error is logged, and the PIU is discarded.

ERC consists of a single send-receive procedure, which is presented next.

```
PC.ERC: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE ROUTES PIU'S RECEIVED FROM THE VR     │
 │                 MANAGER (CHAPTER 12), VRC, TGC, AND UPM_NON_SNA_SEND │
 │                 (SEE NOTE).                                          │
 │                                                                      │
 │                 NON-SNA PIU'S DESTINED FOR THE LOCAL SUBAREA ARE     │
 │                 SENT TO UPM_NON_SNA_RCV (SEE NOTE).                  │
 │                                                                      │
 │                 PU-PU FLOW PIU'S DESTINED FOR THE LOCAL SUBAREA ARE  │
 │                 SENT TO THE PATH CONTROL ROUTE MANAGER COMPONENT OF  │
 │                 THE PU SERVICES MANAGER (CHAPTER 12).                │
 │                                                                      │
 │                 ALL OTHER PIU'S DESTINED FOR THE LOCAL SUBAREA ARE   │
 │                 SENT TO VRC.                                         │
 │                                                                      │
 │                 PIU'S DESTINED FOR OTHER SUBAREAS ARE SENT TO TGC IF │
 │                 TG ROUTING CAN BE ESTABLISHED; OTHERWISE, AN ERROR   │
 │                 IS LOGGED, AND THE PIU IS DISCARDED.                 │
 │                                                                      │
 │      INPUT:     PIU FROM PU.SVC_MGR.PC_ROUTE_MGR.VR_MGR, PC.VRC,     │
 │                 PC.TGC, OR UPM_NON_SNA_SEND; MU_PTR POINTS TO PIU    │
 │                                                                      │
 │      OUTPUT:    PIU TO PU.SVC_MGR.PC_ROUTE_MGR.RCV, PC.VRC.RCV,      │
 │                 PC.TGC.LIST_BY_PRTY, OR UPM_NON_SNA_RCV, IF NOT      │
 │                 DISCARDED. IF PIU IS ROUTED TO PC.TGC.LIST_BY_PRTY,  │
 │                 THE TGCB_PTR POINTS TO THE TGCB FOR THE TG OVER      │
 │                 WHICH THE PIU IS TO BE TRANSMITTED.                  │
 │                                                                      │
 │      NOTE:      UPM_NON_SNA_SEND IS AN UNDEFINED PROTOCOL MACHINE BY │
 │                 WHICH THE NODE CAN INITIATE THE TRANSMISSION OF      │
 │                 NON-SNA PIU'S WITHIN THE SNA NETWORK. UPM_NON_SNA_RCV│
 │                 IS AN UNDEFINED PROTOCOL MACHINE TO WHICH NON-SNA    │
 │                 PIU'S DESTINED FOR THE NODE ARE ROUTED. TOGETHER,    │
 │                 THESE UPM'S ALLOW THE NODE TO TRANSMIT AND RECEIVE   │
 │                 NON-SNA TRAFFIC USING THE SNA NETWORK.               │
 │                                                                      │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
 │                 LOG_ERROR_AND_DISCARD_PIU          PAGE 3-101        │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF DSAF = NCB.NODE_SUBAREA_ADDRESS THEN
    DO;                                                 /* LOCAL SUBAREA ROUTING   */
    .
    . MUCB.DIRECTION = RECEIVE;
    .
    . IF SNAI = ¬SNA THEN
    .   SEND MU TO UPM_NON_SNA_RCV;                     /* NOTE                    */
    .
    . ELSE
    .   IF VR_SQTI = NSEQ_NSUP & DEF = 0 & OEF = 0 THEN
    .     SEND MU TO PU.SVC_MGR.PC_ROUTE_MGR.RCV;       /* CHAPTER 12              */
    .
    .   ELSE
    .     SEND MU TO PC.VRC.RCV;                        /* PAGE  3-63              */
    .
    END;

  ELSE                                                  /* OTHER SUBAREA ROUTING   */
    DO;                                                 /* ESTABLISH TG ROUTING    */
    .
    . FIND SUBAREA_ROUTING IN SUBAREA_ROUTING_LIST
    .      WHERE(SUBAREA_ROUTING.DEST_SA = DSAF);
    .
    . IF SUBAREA_ROUTING_PTR ¬= NULL THEN
    .   DO;
    .   .
    .   . FIND TGCB IN TGCB_LIST
    .   .      WHERE(TGCB.TG_ID = SUBAREA_ROUTING.TG_ID(ERN));  /* TG_ID = ADJ_SA,TGN */
    .   .
    .   . IF TGCB_PTR ¬= NULL THEN
    .   .   SEND MU TO PC.TGC.LIST_BY_PRTY;             /* PAGE 3-23               */
    .   .
    .   . ELSE
    .   .   CALL LOG_ERROR_AND_DISCARD_PIU('TG ROUTING ERROR');  /* PAGE 3-101     */
    .   .
    .   END;
    .
    . ELSE
    .   CALL LOG_ERROR_AND_DISCARD_PIU('DSAF ROUTING ERROR');    /* PAGE 3-101     */
    .
    END;

  RETURN;

END PC.ERC;
```

VIRTUAL ROUTE CONTROL

This section discusses the controls provided by *virtual routes* for message units sent within and between subareas.

A virtual route (VR) is a bidirectional logical connection either within a subarea or between subareas, and can be denoted by:

- The address of the subarea at one end of the VR
- The address of the subarea at the other end of the VR
- The VR number (VRN)
- The assigned transmission priority (TPF)

The activation and deactivation of VRs and the association of VRs with ERs are described in Chapter 12.

The sending of message units over VRs is implemented in virtual route control (VRC), which is a component of subarea routing path control (PC.SA)--see Figure 3-3 on page 3-7. Virtual routes and the functions provided by VRC are described next. This is followed by an overview of the structure of VRC and detailed VRC procedures and FSMs.

Relationship of Virtual Routes and Sessions

A session consists of two half-sessions and the path between them. The path consists of a VR, usually between distinct subareas, and, if necessary, a *route extension* between a subarea node providing boundary function and a peripheral node. If the two half-sessions are within the same subarea, the VR exists entirely within the subarea node. Multiple sessions may be assigned to a VR, but a session is assigned to only one VR while it is active. Each session assigned to a VR has one half-session in the subarea at one end of the VR, and another half-session in the subarea at the other end of the VR. All session activation and deactivation requests and responses received at the end of a VR are routed to the common session control manager component of the PU services manager, rather than to a half-session. The common session control manager (Chapter 13) controls the activation and deactivation of sessions and causes the VR manager (Chapter 12) to assign each session to a VR.

Virtual Route Control Block

When a VR is activated, a virtual route control block (VRCB) is created in each subarea at the ends of the VR, to provide storage for the variables and constants associated with the VR. VRC uses the VRCB to effect and control the transmission and reception of PIUs over a VR. The format of the VRCB is shown in Appendix A.

Transmission Priority

Each VR has one of three transmission priorities assigned to it: low, medium, or high. Except for PIUs flowing at network priority, all PIUs flowing on a given VR flow at the priority assigned to the VR. The transmission priority assigned to a PIU is specified in the Transmission Priority field (TPF) in the FID4 TH. TGC, discussed earlier in this chapter, transmits PIUs according to priority as specified in the TPF.

Some PIUs that control a virtual route flow at network priority, rather than the priority of the virtual route, i.e., VR pacing responses. However, even for these PIUs, the TPF indicates the transmission priority assigned to the VR, since transmission priority is part of the identification of a VR.

TYPES OF VIRTUAL ROUTES

There are two distinct types of VRs; VRs that include only nodes that provide ER and VR controls, and VRs that include one or more pre-ER-VR nodes that do not provide ER and VR controls. Whether or not a VR includes a pre-ER-VR node is maintained in the VRCB.

Virtual Routes Including Only Nodes That Provide ER and VR Controls

All PIUs that flow on VRs that include only nodes that provide ER and VR controls have the ER and VR Support indicator (ER_VR_SUPP_IND) in the FID4 TH set to ¬PRE_ER_VR. For these VRs, there are two types of PIUs handled by VRC: VR-sequenced PIUs and VR pacing responses. VR-sequenced PIUs have the Virtual Route Sequence and Type Indicator (VR_SQTI) field in the FID4 set to singly-sequenced (SING_SEQ); they contain session related data. VR pacing responses have the VR_SQTI set to nonsequenced, supervisory (NSEQ_SUP); they are used to control VR pacing (described below).

Virtual Routes Including Nodes That Do Not Provide ER and VR Controls

All PIUs that flow over VRs that include a pre-ER-VR subarea node that does not provide ER and VR controls have the ER_VR_SUPP_IND in the FID4 TH set to PRE_ER_VR. In addition, all PIUs flowing on these VRs have the virtual route number, explicit route number, initial explicit number, and transmission priority field set to 0, and they always contain session related data. VR PIU sequencing (described below) and VR pacing (described below) are not applicable to VRs that include a pre-ER-VR node; VR segmenting and BIU assembly (described below) are applicable.

CONTROLS PROVIDED BY VIRTUAL ROUTE CONTROL

Virtual Route PIU Sequencing

With the exception of VR pacing responses, all PIUs transmitted by VRC over a VR that includes only nodes that provide ER and VR controls are VR-sequenced. VR-sequenced PIUs have the VR_SQTI in the FID4 TH set to SING_SEQ and are numbered sequentially to provide end-to-end integrity on the VR. The sequence number is carried in the VR_SNF_SEND field in the FID4 TH. The sequence numbers for PIUs flowing in one direction are independent of the sequence numbers for PIUs flowing in the opposite direction. Each VRCB contains two counters, one for PIUs sent by VRC and one for PIUs received by VRC. The VRCB sequence number counters are initialized as specified by the VR_SEND_SEQ_NO field of NC_ACTVR (see Chapter 12).

When VRC transmits a VR-sequenced PIU, it sets the VR_SNF_SEND field in the TH equal to the value of the send counter in the VRCB and increments the counter by 1.

When VRC receives a VR-sequenced PIU, it checks the TG_NONFIFO_IND in the TH. If the TG_NONFIFO_IND is set to ¬FIFO, indicating that TGC is not responsible for maintaining the order of the PIU, the PIU is discarded. If the TG_NONFIFO_IND is set to FIFO, indicating that TGC is responsible for maintaining the order of the PIU, VRC compares the VR_SNF_SEND field in the TH to the current value in the VRCB receive counter. If the values are not equal, the PIU is discarded. If the values are equal, the VRCB receive counter is incremented by 1 and the PIU is processed by VRC.

Virtual Route Pacing

Virtual route pacing allows each node along the ER underlying a VR to control the flow of VR-sequenced PIUs on the VR. The pacing of PIUs flowing in one direction on a VR is independent of the pacing of PIUs flowing in the opposite direction.

Pacing is done by limiting the number of PIUs that can be sent from one end of a VR before receiving a VR pacing response from the other end of the VR. The number of PIUs that can be sent is called the pacing window size. There are two pacing window sizes associated with a VR--one for each direction of flow; these change independently of each other as network conditions change. The minimum and maximum window sizes for a particular VR are specified by values in the MIN_WINDOW_SIZE and MAX_WINDOW_SIZE fields of NC_ACTVR when the VR is activated. The minimum and maximum apply to pacing window sizes for both directions on a VR.

The first PIU of a window carries a VR pacing request, which is indicated by setting the VRPRQ bit in the FID4 TH of a VR-sequenced PIU.

A virtual route pacing response is indicated by setting the VRPRS bit in the FID4 TH. VRC sets the VRPRS bit only in a VR pacing response, which consists of a TH with no BIU data. A VR pacing response is a non-sequenced, supervisory PIU that flows at network priority; it may overtake VR-sequenced PIUs being transmitted over the VR. With the exception of the first VR pacing response, VRC sends a VR pacing response only after receiving a PIU with the VRPRQ bit set. A VR pacing response is never sent unless there are sufficient node resources available to receive the PIUs of the next pacing window.

The pacing count is the current number of VR-sequenced PIUs that can be sent on a VR without requiring the receipt of a pacing response. There are two pacing counts--one at each end of the VR; the counts change independently of each other.

Initially the window size for each direction of transmission on a VR is the minimum window size. As PIUs are transmitted over the VR, the window size for each direction of transmission may increase, up to the maximum specified in NC_ACTVR, by increments of 1 as each successive window is sent, if the network can accommodate the larger window, and if there are enough PIUs being transmitted to require the larger window size. Nodes in the ER underlying a VR can cause reductions in the window size, down to the minimum specified in NC_ACTVR, for either direction of transmission by setting the VR Reset Window indicator (VR_RWI), or the Virtual Route Change Window indicator (VR_CWI) in the FID4 TH.

RESET WINDOW INDICATOR

The Virtual Route Reset Window indicator (VR_RWI) provides a means for any node on a VR to reduce a VR pacing window to the minimum window size specified in NC_ACTVR. The change in window size is for the direction of transmission opposite to that of the PIU carrying the VR_RWI. The VR_RWI provides a means for quickly reducing the number of PIUs in a VR pacing window when a node on a VR is experiencing severe congestion. When a VR-sequenced PIU or a VR pacing response PIU is transmitted over a VR, VRC may set the VR_RWI. In addition, as the PIU traverses the VR, any TGC element that receives the PIU may set the VR_RWI. When VRC receives a PIU with the VR_RWI set, it reduces the window size for PIUs

that it sends over the VR to the minimum window size specified in NC_ACTVR. If the current pacing count is greater than the minimum window size, VRC also reduces the current pacing count to the minimum window size.

CHANGE WINDOW AND CHANGE WINDOW REPLY INDICATORS

The Virtual Route Change Window indicator (VR_CWI) and the Virtual Route Change Window Reply indicator (VR_CWRI) provide a means to gradually increase or decrease the size of a VR pacing window within the range of the minimum and maximum specified in NC_ACTVR. The change in window size is for the direction of transmission of the PIU carrying the VR_CWI, and for the direction of transmission opposite to that of the PIU carrying the VR_CWRI. The VR_CWI and VR_CWRI pair provide a means for:

- Increasing the number of PIUs in a VR pacing window when no node on the VR has congestion problems, and a larger window size is needed to prevent pacing delays, or
- Decreasing the number of PIUs in a VR pacing window when a node on the VR is experiencing moderate congestion.

When a VR-sequenced PIU or a VR pacing response PIU is transmitted over a virtual route, VRC sets the VR_CWI to specify that the window size be increased. As the PIU traverses the VR, any TGC element that transmits the PIU may set the VR_CWI to specify that the window size be decreased; once the VR_CWI has been set to specify a decrease, no subsequent TGC along the VR may change its value.

The VR_CWRI is set only by VRC and only when a VR pacing response is sent. When VRC sends a VR pacing response, it determines if a VR-sequenced PIU or a VR pacing response with the VR_CWI set to indicate a decrease in window size has been received since the previous VR pacing response was sent. If so, VRC sets the VR_CWRI to cause a decrease in window size. VRC may also set the VR_CWRI to cause a decrease in window size if the node determines that a gradual decrease is needed, regardless of whether a VR_CWI indicating a decrease in window size has been received. The VR_CWRI has only two settings: if it is not set to indicate a decrease in window size, it indicates an increase.

When a VR pacing response is received, VRC checks the VR_RWI. If the VR_RWI does not indicate that the window size is to be reset to the minimum, then, based on the setting of the VR_CWRI, VRC conditionally increases by 1 or decreases by 1 the window size for PIUs that it sends over the VR and increments the current VR pacing count by the resultant window size. The window size is not incremented unless the current pacing count has been exhausted when the

VR pacing response is received. If the pacing count has not been exhausted, the current pacing window size is sufficiently large to allow PIU traffic to be sent on the VR without pacing delays, and there is no need to increase the pacing window size--in this case the pacing count is not changed. The window size is never decreased below the minimum nor increased above the maximum specified in NC_ACTVR.

PACING COUNT INDICATOR

The VR Pacing Count indicator (VR_PAC_CNT_IND) is set in the TH of a VR-sequenced PIU that when sent reduces the pacing count to 0. Since the pacing count is 0, no additional VR-sequenced PIUs can be sent until a VR pacing response is received. The receiver of a PIU with the VR_PAC_CNT_IND set may take implementation-dependent action to expedite the transmission of a VR pacing response.

Virtual Route Segmenting and BIU Assembly

BIUs or BIU segments transmitted over a VR may be segmented by an implementation-dependent procedure in VRC into multiple PIUs consisting of BIU segments. When segmenting is performed, the resultant PIUs are transmitted over the VR in an order corresponding to the order of the segments in the original BIU or BIU segment.

BIU segments received by VRC and destined for a half-session for a NAU local to the subarea node are assembled on a half-session basis into a BIU before being passed to the appropriate half-session.

BIU segments received by VRC and destined for a NAU supported by the subarea node boundary function are rejected; VRC does not provide BIU assembly for PIUs destined to a NAU supported by boundary function.

The following message units are not segmented when transmitted over a VR.

- BIUs destined for a half-session for a NAU supported by boundary function
- BIUs containing session-activation or session-deactivation requests or responses
- BIUs or first BIU segments less than 11 bytes in length
- VR pacing responses

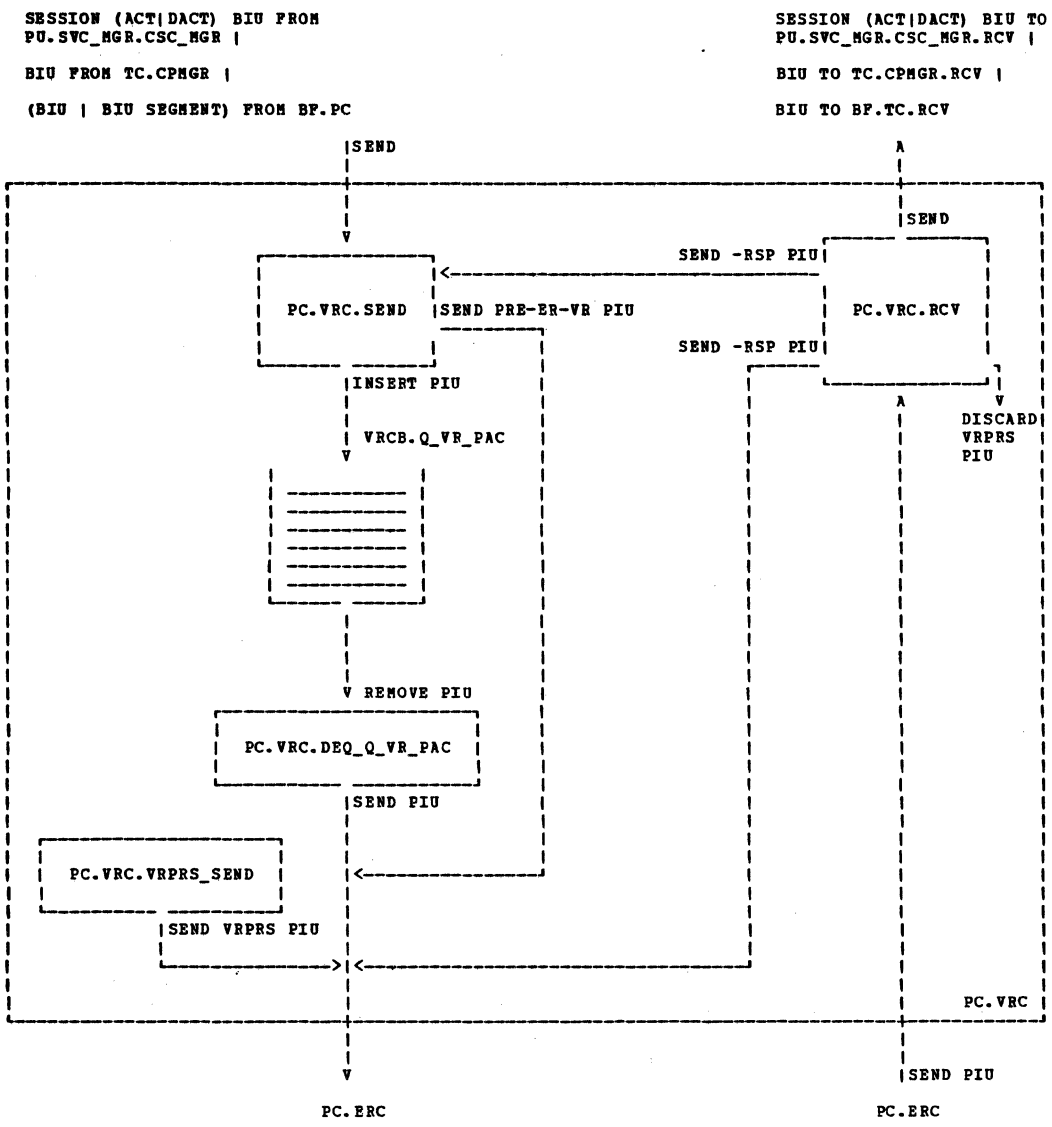


Figure 3-5. Structure of Virtual Route Control (PC.VRC)

```
PC.VRC.SEND: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE, WHEN NECESSARY, ESTABLISHES THE VR   │
  │                 ASSOCIATED WITH A MESSAGE UNIT (SETS VRCB_PTR TO     │
  │                 POINT TO VRCB FOR VR ASSOCIATED WITH MESSAGE UNIT),  │
  │                 MAY SEGMENT THE MESSAGE UNIT INTO MULTIPLE PIU'S,    │
  │                 INITIALIZES MANY OF THE FIELDS IN THE PIU TRANSMISSION│
  │                 HEADER(S), AND EITHER ENQUEUES THE PIU(S) ON THE VR  │
  │                 PACING QUEUE (IF VR DOES NOT INCLUDE A PRE-ER-VR     │
  │                 NODE), OR ROUTES THE PIU(S) TO PC.ERC (IF VR         │
  │                 INCLUDES A PRE-ER-VR NODE).                          │
  │                                                                      │
  │      INPUT:     SESSION ACTIVATION|DEACTIVATION BIU FROM PU.SVC_MGR.CSC_MGR, BIU │
  │                 FROM TC.CPMGR, BIU OR BIU SEGMENT FROM BF.TC, OR NEGATIVE RESPONSE │
  │                 FROM VRC_NEG_RSP; MU_PTR POINTS TO THE MU. SCB_PTR POINTS TO THE │
  │                 SCB ASSOCIATED WITH MU FROM TC.CPMGR OR BF.TC. VRCB_PTR POINTS TO │
  │                 THE VRCB ASSOCIATED WITH MU FROM PU.SVC_MGR.CSC_MGR OR VRC_NEG_RSP. │
  │                                                                      │
  │      OUTPUT:    PIU(S) TO VRCB.Q_VR_PAC OR PC.ERC                    │
  │                                                                      │
  │      NOTES:     1. BBIUI AND EBIUI FOR MU'S FROM BF.TC ARE ALREADY SET, EITHER BY │
  │                    THE PERIPHERAL NODE OR BY BF.TC; MU'S FROM A PERIPHERAL NODE MAY │
  │                    BE A BIU OR BIU SEGMENT.                          │
  │                                                                      │
  │                 2. DSAF, OSAF, DEF, AND OEF ARE ALREADY INITIALIZED FOR MU'S FROM │
  │                    PU.SVC_MGR.CSC_MGR AND VRC_NEG_RSP.               │
  │                                                                      │
  │                 3. PIU'S ARE DEQUEUED FROM VRCB.Q_VR_PAC AND ROUTED TO PC.ERC BY │
  │                    PC.VRC.DEQ_Q_VR_PAC, PAGE 3-60.                   │
  │                                                                      │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
  │                 UPM_VRC_SEGMENTER                PAGE 3-59           │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF DISPATCHED_BY(TC.CPMGR*) | DISPATCHED_BY(BF.TC*) THEN
    VRCB_PTR = SCB.VRCBPTR;

  IF ¬DISPATCHED_BY(BF.TC*) THEN                        /* NOTE 1        */
    DO;
    . BBIUI = BBIU;
    .
    . EBIUI = EBIU;
    END;

  INSERT MU IN VRCB.PIU_SEND_LIST;

  CALL UPM_VRC_SEGMENTER;                               /* PAGE 3-59     */

  DO UNTIL EMPTY(VRCB.PIU_SEND_LIST);
  .
  . REMOVE FIRST(MU) FROM VRCB.PIU_SEND_LIST SET(MU_PTR);
  .
  . FID = FID4;
  .
  . TG_SWEEP = ¬SWEEP;
  .
  . ER_VR_SUPP_IND = VRCB.ER_VR_SUPP;
  .
  . NTWK_PRTY = ¬N_PRTY;
  .
  . IERN = VRCB.VR_NUM;
  .
  . ERN = VRCB.ER_NUM;
  .
  . VRN = VRCB.VR_NUM;
  .
  . TPF = VRCB.TP_FIELD;
  .
  . TG_NONFIFO_IND = FIFO;
  .
  . SNAI = SNA;
  .
  . IF DISPATCHED_BY(TC.CPMGR*) | DISPATCHED_BY(BF.TC*) THEN    /* NOTE 2 */
  .   DO;
  .   . DSAF = SCB.PARTNER_SA;
  .   .
  .   . OSAF = SCB.THIS_SA;
  .   .
  .   . DEF = SCB.PARTNER_EA;
  .   .
  .   . OEF = SCB.THIS_EA;
  .   END;
  .
  . IF VRCB.ER_VR_SUPP = ¬PRE_ER_VR THEN
  .   INSERT MU LAST IN VRCB.Q_VR_PAC;                    /* NOTE 3        */
  .
  . ELSE
  .   SEND MU TO PC.ERC;                                  /* PAGE 3-50     */
  .
  END;

  RETURN;

END PC.VRC.SEND;
```

```
UPM_VRC_SEGMENTER: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS OPTIONAL, IMPLEMENTATION-DEPENDENT  UPM MAY SEGMENT A  BIU OR A │
   │                 BIU SEGMENT INTO MULTIPLE BIU SEGMENTS.  IF SEGMENTING IS PERFORMED, │
   │                 THE RESULTING BIU  SEGMENTS ARE PLACED IN  THE VRCB.PIU_SEND_LIST IN │
   │                 AN ORDER CORRESPONDING TO THE ORDER  OF THE SEGMENTS IN THE ORIGINAL │
   │                 BIU OR BIU SEGMENT, AND WITH:                        │
   │                                                                      │
   │                 • THE BBIUI AND EBIUI APPROPRIATELY SET IN EACH SEGMENT │
   │                                                                      │
   │                 • THE EFI IN EACH  BIU SEGMENT SET EQUAL TO THE  EFI IN THE ORIGINAL │
   │                   BIU OR FIRST BIU SEGMENT                           │
   │                                                                      │
   │                 • THE SNF IN EACH  SEGMENT SET EQUAL TO THE VALUE OF  THE SNF IN THE │
   │                   ORIGINAL BIU OR FIRST BIU SEGMENT                  │
   │                                                                      │
   │      INPUT:     BIU OR  BIU SEGMENT  POINTED TO BY  ENTRY IN  VRCB.PIU_SEND_LIST AND │
   │                 MU_PTR                                               │
   │                                                                      │
   │      OUTPUT:    BIU SEGMENTS IN VRCB.PIU_SEND_LIST IF SEGMENTING IS PERFORMED │
   │                                                                      │
   │      NOTE:      A FIRST BIU SEGMENT  IS REQUIRED TO BE AT LEAST  10 BYTES IN LENGTH; │
   │                 HENCE, BIU'S OR FIRST BIU SEGMENTS LESS  THAN 11 BYTES IN LENGTH ARE │
   │                 NOT SEGMENTED.  IN  ADDITION, BIU'S DESTINED FOR A  NAU SUPPORTED BY │
   │                 BOUNDARY     FUNCTION      AND     BIU'S      CONTAINING     SESSION │
   │                 ACTIVATION|DEACTIVATION RU'S ARE NOT SEGMENTED.      │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 PC.VRC.SEND                          PAGE 3-58       │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  /* FUNCTION AS DESCRIBED ABOVE */

  RETURN;

END UPM_VRC_SEGMENTER;
```

```
PC.VRC.DEQ_Q_VR_PAC: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE SENDS PIU'S FROM THE VR PACING QUEUE. │
  │                                                                      │
  │                 IF THE VR PACING COUNT IS GREATER THAN  0, A PIU IS  │
  │                 REMOVED FROM THE VR  PACING QUEUE,  THE  VR PACING   │
  │                 COUNT IS  DECREMENTED  BY 1,  THE VR_SNF_SEND FIELD  │
  │                 AND VR PACING  INDICATORS IN  THE TH ARE  SET, AS    │
  │                 APPROPRIATE, AND THE PIU IS ROUTED TO PC.ERC         │
  │                                                                      │
  │      INPUT:     OPEN_QUEUE SIGNAL FROM HIGHER-LEVEL SCHEDULER WITH   │
  │                 VRCB_PTR POINTING TO VRCB; VRCB.Q_VR_PAC CONTAINS    │
  │                 (¬PRE-ER-VR) PIU(S).                                 │
  │                                                                      │
  │      OUTPUT:    PIU TO PC.ERC IF VR PACING COUNT IS GREATER THAN 0    │
  │                                                                      │
  │      NOTE:      THE VR_PAC_CNT_IND  IS SET  WHEN RECEIPT OF  A VIRTUAL  ROUTE PACING
  │                 RESPONSE IS  REQUIRED BEFORE ANOTHER  VR-SEQUENCED PIU CAN  BE SENT.
  │                 THERE IS NO  ARCHITECTED CHECK OF THIS  BIT WHEN A PIU  IS RECEIVED,
  │                 BUT  THE BIT  MAY  BE CHECKED  BY  IMPLEMENTATIONS  TO EXPEDITE  THE
  │                 TRANSMISSION OF THE VRPRS.                           │
  │                                                                      │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
  │                 FSM_VRPRQ_SEND                  PAGE 3-74            │
  │                 UPM_SET_RWI                     PAGE 3-60            │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF VRCB.PACING_COUNT > 0 THEN
    DO;
    .
    . REMOVE FIRST(MU) FROM VRCB.Q_VR_PAC SET(MU_PTR);
    .
    . VRCB.PACING_COUNT = VRCB.PACING_COUNT - 1;
    .
    . IF VRCB.PACING_COUNT = 0 THEN
    .   VR_PAC_CNT_IND = PAC_CNT_0;                          /* NOTE    */
    .
    . ELSE
    .   VR_PAC_CNT_IND =  ¬PAC_CNT_0;
    .
    . VR_CWI = INC_WS;
    .
    . VR_SQTI = SING_SEQ;
    .
    . IF VRCB.PACING_COUNT = VRCB.WINDOW_SIZE &
    .      FSM_VRPRQ_SEND = VRPRS_RECEIVED THEN               /* PAGE 3-74 */
    .   VRPRQ = VR_PAC_RQ;
    .
    . ELSE
    .   VRPRQ =  ¬VR_PAC_RQ;
    .
    . CALL FSM_VRPRQ_SEND;                                   /* PAGE 3-74 */
    .
    . VR_RWI = UPM_SET_RWI;                                  /* PAGE 3-60 */
    .
    . VR_SNF_SEND = VRCB.SNF_SEND_CNTR;
    .
    . VRCB.SNF_SEND_CNTR = VRCB.SNF_SEND_CNTR + 1;
    .
    . SEND MU TO PC.ERC;                                     /* PAGE 3-50 */
    .
    END;

  RETURN;

END PC.VRC.DEQ_Q_VR_PAC;



UPM_SET_RWI: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS IMPLEMENTATION-DEPENDENT  UPM DETERMINES, BASED  UPON RESOURCES
  │                 AVAILABLE AT THIS NODE, IF THE PACING WINDOW AT THE OTHER END OF THE
  │                 VR NEEDS TO BE  REDUCED TO THE MINIMUM WINDOW SIZE.   IF SO, IT SETS
  │                 THE VR_RWI TO RESET_WS; IF NOT, IT SETS THE VR_RWI TO ¬RESET_WS.     │
  │                                                                      │
  │      INPUT:     PIU, POINTED TO BY MU_PTR; VRCB_PTR POINTS TO VRCB   │
  │                                                                      │
  │      OUTPUT:    VR_RWI IN TH SET TO RESET_WS OR ¬RESET_WS            │
  │                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
  │                 PC.VRC.DEQ_Q_VR_PAC             PAGE 3-60            │
  │                 PC.VRC.VRPRS_SEND               PAGE 3-61            │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */

 /* FUNCTION AS DESCRIBED ABOVE */

 RETURN;

END UPM_SET_RWI;
```

```
PC.VRC.VRPRS_SEND: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE BUILDS AND SENDS A VR PACING RESPONSE (VRPRS).
   │
   │                 THE INVOCATION OF THIS PROCEDURE IS IMPLEMENTATION DEPENDENT; WITHIN
   │                 THE META-IMPLEMENTATION,  IT IS  INVOKED BY  THE HIGHER-LEVEL
   │                 SCHEDULER.
   │
   │                 A VRPRS IS SENT ONLY IF THE VR IS ACTIVE, THE VR INCLUDES ONLY NODES
   │                 THAT PROVIDE ER AND VR CONTROLS, A VRPRQ HAS BEEN RECEIVED AND NOT
   │                 YET RESPONDED TO BY A VRPRS, AND SUFFICIENT NODE RESOURCES ARE
   │                 AVAILABLE TO HANDLE THE NEXT WINDOW OF PIU'S FROM THE OTHER END OF
   │                 THE VR.
   │
   │      INPUT:     VRCB_PTR POINTS TO VRCB
   │
   │      OUTPUT:    ISOLATED VRPRS PLU TO PC.ERC, IF CONDITIONS ALLOW.
   │
   │      NOTE:      ALL FIELDS IN THE MU (DEFINED IN APPENDIX C) ARE SET TO 0 WHEN IT IS
   │                 CREATED.
   │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):
   │                 FSM_SET_CWRI                    PAGE 3-74
   │                 FSM_VRPRQ_RCV                   PAGE 3-74
   │                 UPM_RESOURCES                   PAGE 3-62
   │                 UPM_SET_CWRI                    PAGE 3-62
   │                 UPM_SET_RWI                     PAGE 3-60
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF FSM_VR = ACTIVE &                                  /* CHAPTER 12     */
       VRCB.ER_VR_SUPP =  ¬PRE_ER_VR &
       FSM_VRPRQ_RCV = VRPRQ_RECEIVED &                 /* PAGE 3-74      */
       UPM_RESOURCES = AVAILABLE THEN                   /* PAGE 3-62      */
    DO;
    .
    . CREATE MU;                                        /* NOTE           */
    .
    . FID = FID4;
    .
    . TG_SWEEP =  ¬SWEEP;
    .
    . ER_VR_SUPP_IND =  ¬PRE_ER_VR;
    .
    . NTWK_PRTY = N_PRTY;
    .
    . IERN = VRCB.VR_NUM;
    .
    . ERN = VRCB.ER_NUM;
    .
    . VRN = VRCB.VR_NUM;
    .
    . TPF = VRCB.TP_FIELD;
    .
    . VR_CWI = INC_WS;
    .
    . TG_NONFIFO_IND = FIFO;
    .
    . VR_SQTI = NSEQ_SUP;
    .
    . VRPRS = VR_PAC_RSP;
    .
    . IF FSM_SET_CWRI = SET_CWRI THEN                   /* PAGE 3-74      */
    .   DO;
    .   .
    .   . VR_CWRI = DEC_WS_RPLY;
    .   .
    .   . CALL FSM_SET_CWRI('RESET');                   /* PAGE 3-74      */
    .   .
    .   END;
    .
    . ELSE
    .   VR_CWRI = UPM_SET_CWRI;                         /* PAGE 3-62      */
    .
    . VR_RWI = UPM_SET_RWI;                             /* PAGE 3-60      */
    .
    . DSAF = VRCB.PARTNER_SA;
    .
    . OSAF = NCB.NODE_SUBAREA_ADDRESS;
    .
    . SNAI = SNA;
    .
    . BBIUI = BBIU;
    .
    . EBIUI = EBIU;
    .
    . CALL FSM_VRPRQ_RCV('RESET');                      /* PAGE 3-74      */
    .
    . SEND MU TO PC.ERC;                                /* PAGE 3-50      */
    .
    END;

  RETURN;

END PC.VRC.VRPRS_SEND;
```

```
UPM_RESOURCES: PROCEDURE RETURNS(CHARACTER(13));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS IMPLEMENTATION-DEPENDENT  UPM RETURNS AN AVAILABLE  RETURN CODE
  │                 IF  SUFFICIENT  RESOURCES ARE  AVAILABLE  TO  ALLOW THE  RECEIPT  OF
  │                 ANOTHER WINDOW OF PIU'S FROM THE OTHER  END OF THE VR; OTHERWISE, IT
  │                 RETURNS A ¬AVAILABLE RETURN CODE.
  │
  │      INPUT:     VRCB_PTR POINTS TO VRCB
  │
  │      OUTPUT:    EITHER AVAILABLE OR ¬AVAILABLE RETURN CODE
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 PC.VRC.VRPRS_SEND                  PAGE 3-61
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);

  RC = AVAILABLE;                                          /* NORMAL RETURN CODE      */

  /* FUNCTION AS DESCRIBED ABOVE */

  RETURN(RC);

END UPM_RESOURCES;



UPM_SET_CWRI: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS IMPLEMENTATION-DEPENDENT UPM DETERMINES, BASED ON THE RESOURCES
  │                 AVAILABLE AT THIS NODE,  IF THE PACING WINDOW SIZE AT  THE OTHER END
  │                 OF THE VR NEEDS TO BE SOMEWHAT  REDUCED.  IF SO, IT SETS THE VR_CWRI
  │                 TO DEC_WS_RPLY; IF NOT, IT SETS THE VR_CWRI TO ¬DEC_WS_RPLY.
  │
  │      INPUT:     VRPRS PIU, POINTED TO BY MU_PTR; VRCB_PTR POINTS TO VRCB
  │
  │      OUTPUT:    VR_CWRI IN VRPRS SET TO DEC_WS_RPLY OR ¬DEC_WS_RPLY
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 PC.VRC.VRPRS_SEND                  PAGE 3-61
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  /* FUNCTION AS DESCRIBED ABOVE */

  RETURN;

END UPM_SET_CWRI;
```

```
PC.VRC.RCV: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │     FUNCTION:  IF  POSSIBLE, THIS  PROCEDURE FINDS  THE  VR ASSOCIATED  WITH A  PIU  │
 │                RECEIVED FROM  PC.ERC AND ROUTES  THE PIU TO  SESSION_RELATED_RCV OR  │
 │                PC.VRPRS_RCV, AS APPROPRIATE.  IF THE VR  CANNOT BE FOUND, IF THE VR  │
 │                IS NOT ACTIVE, OR IF APPROPRIATE  ROUTING CANNOT BE DETERMINED, THEN  │
 │                EITHER AN ERROR  IS LOGGED AND THE  PIU IS DISCARDED, OR  A NEGATIVE  │
 │                RESPONSE IS ROUTED BACK TO PC.ERC.                    │
 │                                                                      │
 │                THIS PROCEDURE CREATES VRCB'S FOR VR'S THAT INCLUDE A NODE THAT DOES  │
 │                NOT PROVIDE ER AND VR CONTROLS--A PRE-ER-VR NODE.  A VRCB IS CREATED  │
 │                THE FIRST TIME A SESSION ACTIVATION REQUEST IS RECEIVED ON A VR THAT  │
 │                INCLUDES A  PRE-ER-VR NODE.  THIS  PROCEDURE DOES NOT  CREATE VRCB'S  │
 │                FOR VR'S  THAT INCLUDE ONLY NODES  THAT PROVIDE ER AND  VR CONTROLS;  │
 │                THEY ARE CREATED BY THE VR MANAGER (CHAPTER 12).      │
 │                                                                      │
 │     INPUT:     PIU FROM FROM PC.ERC; MU_PTR POINTS TO PIU            │
 │                                                                      │
 │     OUTPUT:    PIU ROUTED TO SESSION_RELATED_RCV OR  VRPRS_RCV, IF NOT DISCARDED OR  │
 │                IF NEGATIVE RESPONSE  NOT ROUTED TO PC.ERC.  A VRCB  FOR A PRE-ER-VR  │
 │                VR  MAY BE  CREATED.  IF  PIU  IS ROUTED  TO SESSION_RELATED_RCV  OR  │
 │                PC.VRPRS_RCV, THE VRCB_PTR  IS SET TO POINT  TO THE VRCB FOR  THE VR  │
 │                ASSOCIATED WITH THE VR SPECIFIED IN THE PIU TH.       │
 │                                                                      │
 │     NOTE:      WHEN A VRCB IS CREATED, ALL FIELDS ARE INITIALIZED TO 0.  │
 │                                                                      │
 │     REFERS TO THE FOLLOWING PROCEDURE(S):                            │
 │                LOG_ERROR_AND_DISCARD_PIU          PAGE 3-101         │
 │                SESSION_RELATED_RCV                PAGE 3-66          │
 │                SWAP_FID4_TH_ORIG_DEST_FLDS        PAGE 3-65          │
 │                VRPRS_RCV                          PAGE 3-73          │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  FIND VRCB IN VRCB_LIST
       WHERE(VRCB.VR_ID = VRID & VRCB.PARTNER_SA = OSAF);       /* VRID = VRN,TPF   */

  IF VRCB_PTR = NULL & ER_VR_SUPP_IND = PRE_ER_VR &
       BBIUI = BBIU & EBIUI = EBIU & DCF > 3 & RRI = RQ & RU_CTGY = SC &
       RQ_CODE = (ACTCDRM | ACTPU | ACTLU | BIND) THEN
    DO;
    .
    . CREATE VRCB;                                                /* NOTE             */
    .
    . IF VRCB_PTR ¬= NULL THEN
    .   DO;
    .   .
    .   . VRCB.PARTNER_SA = OSAF;
    .   .
    .   . VRCB.ER_VR_SUPP = PRE_ER_VR;
    .   .
    .   . CALL FSM_VR('PRE_VR_ACT');                              /* CHAPTER 12       */
    .   .
    .   . NEWLIST VRCB.PIU_SEND_LIST ENTRY_NAME(MU) FIFO;
    .   .
    .   . INSERT VRCB IN VRCB_LIST;
    .   .
    .   END;
    .
    . ELSE
    .   DO;
    .   .
    .   . CALL CHANGE_MU_TO_NEG_RSP(X'0812');  /* APPENDIX B, INSUFFICIENT RESOURCES  */
    .   .
    .   .
    .   . CALL SWAP_FID4_TH_ORIG_DEST_FLDS;                      /* PAGE 3-65        */
    .   .
    .   . SEND MU TO PC.ERC;                                      /* PAGE 3-50        */
    .   .
    .   . RETURN;
    .   .
    .   END;
    END;

  IF VRCB_PTR = NULL | FSM_VR ¬= ACTIVE THEN                      /* CHAPTER 12       */
    CALL LOG_ERROR_AND_DISCARD_PIU('NO ACTIVE VR');               /* PAGE 3-101       */

  ELSE
    SELECT ANYORDER;
    .
    . WHEN(ER_VR_SUPP_IND = PRE_ER_VR | VR_SQTI = SING_SEQ)
    .   CALL SESSION_RELATED_RCV;                                 /* PAGE 3-66        */
    .
    . WHEN(VR_SQTI = NSEQ_SUP & VRPRS = VR_PAC_RSP)
    .   CALL VRPRS_RCV;                                           /* PAGE 3-73        */
    .
    . OTHERWISE
    .   CALL LOG_ERROR_AND_DISCARD_PIU('INVALID SQTI|VRPRS BIT');  /* PAGE 3-101       */
    .
    END;

  RETURN;

END PC.VRC.RCV;
```

This page intentionally left blank

```
SWAP_FID4_TH_ORIG_DEST_FLDS: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE  SWAPS THE ORIGIN  AND DESTINATION ADDRESS  FIELDS IN
  │                 THE FID4 TH--THIS FUNCTION IS REQUIRED TO SEND A NEGATIVE RESPONSE.
  │
  │      INPUT:     PIU, POINTED TO BY MU_PTR
  │
  │      OUTPUT:    OSAF SET FROM  DSAF, DSAF SET FROM  OSAF, OEF SET FROM  DEF, AND DEF
  │                 SET FROM OEF.
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 PC.VRC.RCV                          PAGE 3-63
  │                 VRC_NEG_RSP                         PAGE 3-69
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL OSAF_SAVE BINARY(32);
  DCL OEF_SAVE BINARY(16);

  OSAF_SAVE = OSAF;

  OEF_SAVE = OEF;

  OSAF = DSAF;

  DSAF = OSAF_SAVE;

  OEF = DEF;

  DEF = OEF_SAVE;

  RETURN;

END SWAP_FID4_TH_ORIG_DEST_FLDS;
```

```
SESSION_RELATED_RCV: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  THIS PROCEDURE  IS CALLED  BY PC.VRC.RCV  TO PROCESS  AND ROUTE  (1)
│                VR-SEQUENCED PIU'S--WHICH FLOW ON VR'S  THAT INCLUDE ONLY NODES THAT
│                PROVIDE ER AND  VR CONTROLS--AND (2) PRE-ER-VR  PIU'S--WHICH FLOW ON
│                VR'S THAT INCLUDE A  PRE-ER-VR NODE THAT DOES NOT PROVIDE  ER AND VR
│                CONTROLS.
│
│                FOR VR-SEQUENCED PIU'S, IT PERFORMS  A VIRTUAL ROUTE SEQUENCE NUMBER
│                CHECK AND PROCESSES THE VR PACING INDICATORS IN THE TH.
│
│                PIU'S CONTAINING A SESSION ACTIVATION|DEACTIVATION BIU ARE ROUTED TO
│                PU.SVC_MGR.CSC_MGR.RCV.
│
│                FOR OTHER PIU'S, THE HALF-SESSION ASSOCIATED  WITH THE PIU IS FOUND,
│                IF  POSSIBLE.   IF THE  HALF-SESSION  CANNOT  BE FOUND,  A  NEGATIVE
│                RESPONSE IS GENERATED.
│
│                IF THE HALF-SESSION IS FOR A NAU  LOCAL TO THE SUBAREA, BIU ASSEMBLY
│                IS PERFORMED, IF  REQUIRED, AND WHEN A (WHOLE) BIU  IS AVAILABLE, IT
│                IS ROUTED TO TC.CPMGR.RCV.
│
│                IF THE HALF-SESSION IS FOR A NAU SUPPORTED BY BOUNDARY FUNCTION:
│
│                • PIU'S THAT CONTAIN A BIU ARE ROUTED TO BF.TC.RCV.
│
│                • PIU'S THAT CONTAIN A BIU SEGMENT RESULT IN A NEGATIVE RESPONSE, IF
│                  POSSIBLE; OTHERWISE, THEY ARE DISCARDED.
│
│                SOME PIU VALIDITY CHECKS ARE PERFORMED THAT MAY RESULT IN A NEGATIVE
│                RESPONSE OR AN ERROR BEING LOGGED AND THE PIU BEING DISCARDED.
│
│     INPUT:     PIU FROM PC.VRC.RCV, POINTED TO BY MU_PTR
│
│     OUTPUT:    SESSION ACTIVATION|DEACTIVATION  BIU TO  PU.SVC_MGR.CSC_MGR_RCV, BIU
│                TO TC.CPMGR.RCV OR BF.TC.RCV, NEGATIVE  RESPONSE GENERATED, OR ERROR
│                LOGGED AND PIU DISCARDED.
│
│     NOTES:     1. BIU'S  CONTAINING SESSION  ACTIVATION|DEACTIVATION  RU'S ARE  NOT
│                   SEGMENTED.
│
│                2. THE SESSION SPECIFIED  IN THE TH OF  THIS PIU IS NOT  ASSIGNED TO
│                   THE VR SPECIFIED IN THE TH.
│
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                PC.VRC.RCV                          PAGE 3-63
│
│     REFERS TO THE FOLLOWING PROCEDURE(S):
│                FSM_SET_CWRI                        PAGE 3-74
│                FSM_VRPRQ_RCV                       PAGE 3-74
│                LOG_ERROR_AND_DISCARD_PIU           PAGE 3-101
│                VRC_BIU_ASSEMBLER                   PAGE 3-70
│                VRC_NEG_RSP                         PAGE 3-69
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

```
DCL ASSEMBLER_RESULT BIT(1);

IF ER_VR_SUPP_IND =  ¬PRE_ER_VR THEN
  DO;
  . IF TG_NONFIFO_IND = FIFO THEN
  .
  .   IF VR_SNF_SEND = VRCB.SNF_RCV_CNTR THEN
  .     VRCB.SNF_RCV_CNTR = VRCB.SNF_RCV_CNTR + 1;
  .
  .   ELSE
  .     DO;
  .     . CALL LOG_ERROR_AND_DISCARD_PIU('VR SEQ ERROR');      /* PAGE 3-101          */
  .     .
  .     . RETURN;
  .     END;
  .
  . ELSE
  .   DO;
  .   . CALL LOG_ERROR_AND_DISCARD_PIU('TG NONFIFO PIU');      /* PAGE 3-101          */
  .   .
  .   . RETURN;
  .   END;
  .
  . CALL FSM_VRPRQ_RCV;
  .
  . CALL FSM_SET_CWRI;
  .
  . IF VR_RWI = RESET_WS THEN
  .   DO;
  .   . VRCB.WINDOW_SIZE = VRCB.MIN_WINDOW_SIZE;
  .   .
  .   . IF VRCB.PACING_COUNT > VRCB.MIN_WINDOW_SIZE THEN
  .   .   VRCB.PACING_COUNT = VRCB.MIN_WINDOW_SIZE;
  .   END;
  END;

IF BBIUI = BBIU & DCF < 3 THEN
  CALL LOG_ERROR_AND_DISCARD_PIU(X'4005');  /* PAGE 3-101, INCOMPLETE RH          */

ELSE
  IF BBIUI = BBIU & RU_CTGY = SC & DCF < 4 THEN
    CALL VRC_NEG_RSP(X'1002');          /* PAGE 3-69, RU LENGTH ERROR           */

  ELSE
    IF RU_CTGY = SC &
         RQ_CODE = (ACTCDRM | ACTPU | ACTLU | BIND |
         DACTCDRM | DACTPU | DACTLU | UNBIND) THEN

      IF BBIUI =  ¬BBIU | EBIUI =  ¬EBIU THEN
        CALL VRC_NEG_RSP(X'8007');      /* PAGE 3-69, SEGMENTING ERROR, NOTE 1  */

      ELSE
        SEND MU TO PU.SVC_MGR.CSC_MGR.RCV;                  /* CHAPTER 13       */

    ELSE
      DO;
      . FIND SCB IN SCB_LIST WHERE(OSAF = SCB.PARTNER_SA & OEF = SCB.PARTNER_EA &
      .      DSAF = SCB.THIS_SA & DEF = SCB.THIS_EA);
      .
      . IF SCB_PTR = NULL THEN
      .   CALL VRC_NEG_RSP(X'8005');    /* PAGE 3-101, NO SESSION               */
      .
      . ELSE
      .   IF SCB.VRCBPTR ¬= VRCB_PTR THEN
      .     CALL LOG_ERROR_AND_DISCARD_PIU('WRONG VR');       /* PAGE 3-101, NOTE 2 */
      .
      .   ELSE
      .     SELECT ANYORDER(SCB.SCB_TYPE);
      .     .
      .     . WHEN(HALF_SESS)
      .     .   DO;
      .     .   . CALL VRC_BIU_ASSEMBLER(ASSEMBLER_RESULT);   /* PAGE 3-70       */
      .     .   .
      .     .   . IF ASSEMBLER_RESULT = BIU_AVAILABLE THEN
      .     .   .   SEND MU TO TC.CPMGR.RCV;                  /* CHAPTER 4       */
      .     .   END;
      .     .
      .     . WHEN(BF_SESS)
      .     .   IF BBIUI =  ¬BBIU | EBIUI =  ¬EBIU THEN
      .     .     CALL VRC_NEG_RSP(X'8007');  /* PAGE 3-69, SEGMENTING ERROR    */
      .     .
      .     .   ELSE
      .     .     SEND MU TO BF.TC.RCV;                       /* CHAPTER 4       */
      .     END;
      END;

  RETURN;

END SESSION_RELATED_RCV;
```

This page intentionally left blank

```
VRC_NEG_RSP: PROCEDURE(SNC_CODE);

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE IS CALLED TO CHANGE A FID4 PIU OR PARTIALLY ASSEMBLED │
  │                 BIU TO A  NEGATIVE RESPONSE, SET THE  SENSE CODE EQUAL TO  THE VALUE │
  │                 PASSED  IN THE  CALL  PARAMETER, AND  ROUTE  THE RESULTANT  NEGATIVE │
  │                 RESPONSE TO PC.VRC.SEND.                                     │
  │                                                                              │
  │                 IF THE  MU IS  ONE TO  WHICH NO RESPONSE  CAN BE  SENT, AN  ERROR IS │
  │                 LOGGED, AND THE MU IS DISCARDED.                             │
  │                                                                              │
  │      INPUT:     PIU  CONTAINING BIU  OR  BIU SEGMENT,  OR  PARTIALLY ASSEMBLED  BIU; │
  │                 MU_PTR POINTS TO INPUT MU AND VRCB_PTR POINTS TO VRCB        │
  │                                                                              │
  │      OUTPUT:    NEGATIVE RESPONSE RU TO PC.VRC.SEND IF MU IS ONE TO WHICH A NEGATIVE │
  │                 RESPONSE CAN BE SENT; OTHERWISE, MU IS DISCARDED.            │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 SESSION_RELATED_RCV              PAGE 3-66                   │
  │                 VRC_BIU_ASSEMBLER                PAGE 3-70                   │
  │                 VRC_FIRST_SEGMENT_RCV_CHK        PAGE 3-72                   │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 SWAP_FID4_TH_ORIG_DEST_FLDS      PAGE 3-65                   │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL SNC_CODE BIT(32);

  IF BBIUI =  ¬BBIU | DCF < 3 | RQN | RRI = RSP THEN /* SEE APPENDIX B FOR RQN   */
    DO;
    . CALL UPM_LOG(SNC_CODE);                                  /* APPENDIX B    */
    .
    . DISCARD MU;
    END;

  ELSE
    DO;
    .
    . CALL CHANGE_MU_TO_NEG_RSP(SNC_CODE);                     /* APPENDIX B    */
    .
    . CALL SWAP_FID4_TH_ORIG_DEST_FLDS;                        /* PAGE 3-65     */
    .
    . SEND MU TO PC.VRC.SEND;                                  /* PAGE 3-58     */
    .
    END;

  RETURN;

END VRC_NEG_RSP;
```

```
VRC_BIU_ASSEMBLER: PROCEDURE(ASSEMBLER_RESULT);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE IS CALLED WHEN A PIU IS RECEIVED FOR A HALF-SESSION │
│                 FOR A NAU LOCAL TO THE SUBAREA; IT PERFORMS FUNCTIONS REQUIRED │
│                 RELATIVE TO THE RECEPTION OF PIU'S CONTAINING A BIU OR BIU SEGMENT, │
│                 INCLUDING THE ASSEMBLY OF A BIU FROM PIU'S CONTAINING BIU SEGMENTS, │
│                 AND THE RECOGNITION AND PROCESSING OF ERRORS ASSOCIATED WITH BIU │
│                 SEGMENTING.                                              │
│                                                                          │
│                 FSM_SESSION_BIU_ASSEMBLY IS USED TO MAINTAIN THE STATE OF A │
│                 HALF-SESSION RELATIVE TO BIU ASSEMBLY.                   │
│                                                                          │
│      INPUT:     PIU CONTAINING BIU OR FIRST, MIDDLE, OR LAST BIU SEGMENT; MU_PTR │
│                 POINTS TO PIU; SCB_PTR POINTS TO HALF-SESSION CONTROL BLOCK; │
│                 SCB.PARTIAL_BIU_PTR MAY POINT TO A PARTIALLY ASSEMBLED BIU; │
│                 FSM_SESSION_BIU_ASSEMBLY INDICATES CURRENT STATE OF HALF-SESSION │
│                 RELATIVE TO BIU ASSEMBLY                                 │
│                                                                          │
│      OUTPUT:    THE RETURN PARAMETER, ASSEMBLER_RESULT, IS SET TO EITHER │
│                 BIU_AVAILABLE OR ¬BIU_AVAILABLE. IF ASSEMBLER_RESULT IS SET TO │
│                 BIU_AVAILABLE, MU_PTR POINTS BIU. IF ASSEMBLER_RESULT IS SET TO │
│                 ¬BIU_AVAILABLE, SCB.PARTIAL_BIU_PTR MAY POINT TO PARTIALLY ASSEMBLED │
│                 BIU. A NEGATIVE RESPONSE MAY BE GENERATED RELATIVE TO INPUT PIU OR │
│                 PARTIALLY ASSEMBLED BIU, OR INPUT PIU OR PARTIALLY ASSEMBLED BIU MAY │
│                 BE DISCARDED.                                            │
│                                                                          │
│      NOTES:     1. THIS CONCATENATES THE BIU SEGMENT IN THE CURRENT PIU POINTED TO │
│                    BY MU_PTR TO THE END OF THE PARTIALLY ASSEMBLED BIU POINTED TO BY │
│                    SCB.PARTIAL_BIU_PTR.                                  │
│                                                                          │
│                 2. THIS ADDS THE DCF IN THE CURRENT PIU POINTED TO BY MU_PTR TO THE │
│                    DCF IN THE PARTIALLY ASSEMBLED BIU POINTED TO BY │
│                    SCB.PARTIAL_BIU_PTR.                                  │
│                                                                          │
│                 3. THIS SETS THE EBIUI IN THE BIU BEING ASSEMBLED (POINTED TO BY │
│                    SCB.PARTIAL_BIU_PTR) TO THE VALUE OF THE EBIUI IN THE CURRENT PIU │
│                    POINTED TO BY MU_PTR. IF THE EBIUI IN THE CURRENT PIU IS SET TO │
│                    EBIU, THE PARTIAL BIU POINTED TO BY SCB.PARTIAL_BIU_PTR BECOMES A │
│                    (WHOLE) BIU.                                          │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 SESSION_RELATED_RCV              PAGE 3-66               │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 FSM_SESSION_BIU_ASSEMBLY         PAGE 3-102              │
│                 LOG_ERROR_AND_DISCARD_PIU        PAGE 3-101              │
│                 UPM_BIU_ASSEMBLY_CHK             PAGE 3-101              │
│                 VRC_FIRST_SEGMENT_RCV_CHK        PAGE 3-72               │
│                 VRC_NEG_RSP                      PAGE 3-69               │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL ASSEMBLER_RESULT BIT(1);
DCL P PTR;
DCL TEMP_PIU_PTR PTR;

ASSEMBLER_RESULT =  ¬BIU_AVAILABLE;

SELECT ANYORDER;
.
. WHEN(FSM_SESSION_BIU_ASSEMBLY = BETBIU)                    /* PAGE 3-102      */
.   SELECT ANYORDER(BBIUI);
.   .
.   . WHEN( ¬BBIU)
.   .   CALL LOG_ERROR_AND_DISCARD_PIU(X'8007');  /* PAGE 3-101, SEGMENTING ERROR */
.   .
.   . WHEN(BBIU)
.   .   DO;
.   .   . CALL FSM_SESSION_BIU_ASSEMBLY;                     /* PAGE 3-102      */
.   .   .
.   .   . SELECT ANYORDER;
.   .   . .
.   .   . . WHEN(FSM_SESSION_BIU_ASSEMBLY = BETBIU)          /* PAGE 3-102      */
.   .   . .   ASSEMBLER_RESULT = BIU_AVAILABLE;
.   .   . .
.   .   . . WHEN(FSM_SESSION_BIU_ASSEMBLY = INBIU)           /* PAGE 3-102      */
.   .   . .   IF VRC_FIRST_SEGMENT_RCV_CHK = OK THEN         /* PAGE 3-72       */
.   .   . .     SCB.PARTIAL_BIU_PTR = MU_PTR;
.   .   . .
.   .   . .   ELSE
.   .   . .     CALL FSM_SESSION_BIU_ASSEMBLY('RESET');      /* PAGE 3-102      */
.   .   . END;
.   .   END;
.   END;
```

```
. WHEN(FSM_SESSION_BIU_ASSEMBLY = INBIU)                    /* PAGE 3-102          */
.   SELECT ANYORDER(BBIUI);
.   .
.   . WHEN( ¬BBIU)
.   .   IF SNF ¬= SCB.PARTIAL_BIU_PTR->SNF THEN             /* OPTIONAL CHECK      */
.   .     DO;
.   .     . CALL LOG_ERROR_AND_DISCARD_PIU(X'8007');  /* PAGE 3-101, SEGMENTING ERROR */
.   .     .
.   .     . MU_PTR = SCB.PARTIAL_BIU_PTR;
.   .     .
.   .     . CALL VRC_NEG_RSP(X'8007');   /* PAGE 3-99, SEGMENTING ERROR          */
.   .     .
.   .     . CALL FSM_SESSION_BIU_ASSEMBLY('RESET');           /* PAGE 3-102          */
.   .     END;
.   .
.   .   ELSE
.   .     DO;
.   .     . IF UPM_BIU_ASSEMBLY_CHK = NG THEN /* OPTIONAL CHECK, PAGE 3-101       */
.   .     .   DO;
.   .     .   . CALL LOG_ERROR_AND_DISCARD_PIU(X'8010');      /* PAGE 3-101,         */
.   .     .   .                          /* SEGMENTED RU LENGTH ERROR               */
.   .     .   . MU_PTR = SCB.PARTIAL_BIU_PTR;
.   .     .   .
.   .     .   . CALL VRC_NEG_RSP(X'8010');  /* PAGE 3-69, SEGMENTED RU LENGTH ERROR */
.   .     .   .
.   .     .   . CALL FSM_SESSION_BIU_ASSEMBLY('RESET');       /* PAGE 3-102          */
.   .     .   END;
.   .     .
.   .     . ELSE
.   .     .   DO;
.   .     .   . P = SCB.PARTIAL_BIU_PTR;
.   .     .   .
.   .     .   . P->RU(P->DCF:(P->DCF + DCF - 1)) = RU(0:(DCF - 1));  /* NOTE 1     */
.   .     .   .
.   .     .   . SCB.PARTIAL_BIU_PTR->DCF = SCB.PARTIAL_BIU_PTR->DCF + DCF; /* NOTE 2 */
.   .     .   .
.   .     .   . SCB.PARTIAL_BIU_PTR->EBIUI = EBIUI;           /* NOTE 3              */
.   .     .   .
.   .     .   . DISCARD MU;
.   .     .   .
.   .     .   . MU_PTR = SCB.PARTIAL_BIU_PTR;
.   .     .   .
.   .     .   . CALL FSM_SESSION_BIU_ASSEMBLY;                /* PAGE 3-102          */
.   .     .   .
.   .     .   . IF FSM_SESSION_BIU_ASSEMBLY = BETBIU THEN     /* PAGE 3-102          */
.   .     .   .   ASSEMBLER_RESULT = BIU_AVAILABLE;
.   .     .   END;
.   .     END;
.   .
.   . WHEN(BBIU)
.   .   DO;
.   .   . TEMP_PIU_PTR = MU_PTR;
.   .   .
.   .   . MU_PTR = SCB.PARTIAL_BIU_PTR;
.   .   .
.   .   . IF SNF ¬= TEMP_PIU_PTR->SNF THEN
.   .   .   CALL VRC_NEG_RSP(X'8007');   /* PAGE 3-69, SEGMENTING ERROR            */
.   .   .
.   .   . ELSE
.   .   .   DISCARD MU;                                       /* FORWARD ABORT       */
.   .   .
.   .   . MU_PTR = TEMP_PIU_PTR;
.   .   .
.   .   . CALL FSM_SESSION_BIU_ASSEMBLY;                      /* PAGE 3-102          */
.   .   .
.   .   . SELECT ANYORDER;
.   .   . .
.   .   . . WHEN(FSM_SESSION_BIU_ASSEMBLY = BETBIU)           /* PAGE 3-102          */
.   .   . .   ASSEMBLER_RESULT = BIU_AVAILABLE;
.   .   . .
.   .   . . WHEN(FSM_SESSION_BIU_ASSEMBLY = INBIU)           /* PAGE 3-102          */
.   .   . .   IF VRC_FIRST_SEGMENT_RCV_CHK = OK THEN         /* PAGE 3-72           */
.   .   . .     SCB.PARTIAL_BIU_PTR = MU_PTR;
.   .   . .
.   .   . .   ELSE
.   .   . .     CALL FSM_SESSION_BIU_ASSEMBLY('RESET');       /* PAGE 3-102          */
.   .   . END;
.   .   END;
.   END;
END;

RETURN;

END VRC_BIU_ASSEMBLER;
```

```
VRC_FIRST_SEGMENT_RCV_CHK: PROCEDURE RETURNS(BIT(1));

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE  PERFORMS  RECEIVE  CHECKS  APPLICABLE  WHEN  A  PIU  │
   │                 CONTAINING A FIRST BIU SEGMENT IS RECEIVED BY VRC.   │
   │                                                                      │
   │      INPUT:     PIU, POINTED TO BY MU_PTR                            │
   │                                                                      │
   │      OUTPUT:    OK  RETURN CODE  IF PIU  IS VALID;  OTHERWISE, NG  RETURN CODE.   IF  │
   │                 RETURN  CODE IS  NG, A  NEGATIVE RESPONSE  IS GENERATED,  OR PIU  IS  │
   │                 DISCARDED.                                           │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 VRC_BIU_ASSEMBLER                PAGE 3-70           │
   │                                                                      │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
   │                 VRC_NEG_RSP                      PAGE 3-69           │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RC BIT(1);


  RC = OK;

  IF DCF < 10 THEN                                        /* OPTIONAL CHECK          */
    DO;
    . CALL VRC_NEG_RSP(X'8007');          /* PAGE 3-69, SEGMENTING ERROR              */
    .
    . RC = NG;
    END;

  RETURN(RC);

END VRC_FIRST_SEGMENT_RCV_CHK;
```

```
VRPRS_RCV: PROCEDURE;

/*
┌──────────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  THIS  PROCEDURE IS  CALLED  BY PC.VRC.RCV  TO  PROCESS  A VR  PACING │
│                RESPONSE (VRPRS).                                             │
│                                                                              │
│                IF VR_CWI  IS SET TO DEC_WS,  FSM_SET_CWRI IS SET  TO SET_CWRI--THIS │
│                WILL CAUSE THE VR_CWRI IN THE NEXT VRPRS TRANSMITTED ON THE VR TO BE │
│                SET TO DEC_WS_REPLY.                                          │
│                                                                              │
│                FSM_VRPRQ_SEND IS SET  TO VRPRS_RECEIVED--THIS ALLOWS A  VRPRQ TO BE │
│                TRANSMITTED ON THE VR.                                        │
│                                                                              │
│                IF VR_RWI IS SET  TO RESET_WS, THE VR WINDOW SIZE  AND THE VR PACING │
│                COUNT ARE SET TO THE MINIMUM WINDOW SIZE FOR THE VR.          │
│                                                                              │
│                IF VR_RWI  IS SET  TO ¬RESET_WS, THE  VR WINDOW  SIZE MAY  BE EITHER │
│                INCREMENTED  OR DECREMENTED  AS  SPECIFIED BY  THE  VR_CWRI (THE  VR │
│                WINDOW SIZE  IS INCREMENTED  ONLY IF THE  PACING COUNT  IS 0  AND IS │
│                NEVER  INCREMENTED ABOVE  THE MAXIMUM  WINDOW  SIZE NOR  DECREMENTED │
│                BELOW  THE  MINIMUM  WINDOW  SIZE),  AND  THE  VR  PACING  COUNT  IS │
│                INCREMENTED BY THE RESULTANT VR WINDOW SIZE.                  │
│                                                                              │
│                THE VALUE BY  WHICH THE VR WINDOW SIZE  IS CONDITIONALLY INCREMENTED │
│                OR DECREMENTED--VRCB.WINDOW_CHANGE_SIZE--IS 1.                │
│                                                                              │
│                THE VRPRS IS DISCARDED.                                       │
│                                                                              │
│     INPUT:     VRPRS PIU, POINTED TO BY MU_PTR; VRCB_PTR POINTS TO VRCB      │
│                                                                              │
│     OUTPUT:    IF  VR_CWI  IS SET  TO  DEC_WS,  FSM_SET_CWRI  IS SET  TO  SET_CWRI. │
│                FSM_VRPRQ_SEND IS SET TO VRPRS_RECEIVED.  THE  VR WINDOW SIZE MAY OR │
│                MAY NOT BE  INCREMENTED OR DECREMENTED.  THE VR PACING  COUNT MAY BE │
│                SET TO  THE MINIMUM  WINDOW SIZE FOR  THE VR  OR INCREMENTED  BY THE │
│                RESULTANT VR WINDOW SIZE VALUE.  THE VRPRS PIU IS DISCARDED.  │
│                                                                              │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                                │
│                PC.VRC.RCV                          PAGE 3-63                 │
│                                                                              │
│     REFERS TO THE FOLLOWING PROCEDURE(S):                                    │
│                FSM_SET_CWRI                        PAGE 3-74                 │
│                FSM_VRPRQ_SEND                      PAGE 3-74                 │
└──────────────────────────────────────────────────────────────────────────────┘
*/

  CALL FSM_SET_CWRI;                                    /* PAGE 3-74          */

  CALL FSM_VRPRQ_SEND;                                  /* PAGE 3-74          */

  IF VR_RWI = RESET_WS THEN
    DO;
    .
    . VRCB.WINDOW_SIZE = VRCB.MIN_WINDOW_SIZE;
    .
    . VRCB.PACING_COUNT = VRCB.MIN_WINDOW_SIZE;
    .
    END;

  ELSE
    DO;
    .
    . SELECT ANYORDER(VR_CWRI);
    . .
    . . WHEN(DEC_WS_RPLY)
    . .   DO;
    . .   .
    . .   . VRCB.WINDOW_SIZE = VRCB.WINDOW_SIZE - VRCB.WINDOW_SIZE_CHANGE;
    . .   .
    . .   . IF VRCB.WINDOW_SIZE < VRCB.MIN_WINDOW_SIZE THEN
    . .   .   VRCB.WINDOW_SIZE = VRCB.MIN_WINDOW_SIZE;
    . .   .
    . .   END;
    . .
    . . WHEN(INC_WS_RPLY)
    . .   DO;
    . .   .
    . .   . IF VRCB.PACING_COUNT = 0 THEN
    . .   .   DO;
    . .   .   .
    . .   .   . VRCB.WINDOW_SIZE = VRCB.WINDOW_SIZE + VRCB.WINDOW_SIZE_CHANGE;
    . .   .   .
    . .   .   . IF VRCB.WINDOW_SIZE > VRCB.MAX_WINDOW_SIZE THEN
    . .   .   .   VRCB.WINDOW_SIZE = VRCB.MAX_WINDOW_SIZE;
    . .   .   .
    . .   .   END;
    . .   END;
    . END;
    .
    . VRCB.PACING_COUNT = VRCB.PACING_COUNT + VRCB.WINDOW_SIZE;
    .
    END;

  DISCARD MU;

  RETURN;

END VRPRS_RCV;
```

```
FSM_VRPRQ_SEND: FSM_DEFINITION CONTEXT(VRCB);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  FINITE-STATE  MACHINE  RECORDS  THE RECEIPT  OF  A  VR  PACING │
   │                 RESPONSE (VRPRS),  TO ALLOW THE  TRANSMISSION OF A  VR-SEQUENCED PIU │
   │                 WITH VRPRQ  BIT SET TO  VR_PAC_RQ--THE START  OF THE NEXT  WINDOW OF │
   │                 VR-SEQUENCED PIU'S.  IT IS RESET WHEN  A VR-SEQUENCED PIU WITH VRPRQ │
   │                 BIT SET TO VR_PAC_RQ IS SENT.                        │
   │                                                                      │
   │      NOTE:      THIS FSM  IS CALLED  BY THE  VR MANAGER  (CHAPTER 12)  WHEN NC_ACTVR │
   │                 RESPONSE IS  RECEIVED--VRPRS IS SET  IN NC_ACTVR  RESPONSE--TO ALLOW │
   │                 THE FIRST WINDOW OF PIU'S TO BE TRANSMITTED FROM THIS END OF THE VR. │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 PC.VRC.DEQ_Q_VR_PAC              PAGE 3-60           │
   │                 VRPRS_RCV                        PAGE 3-73           │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

   ┌──────────────────┬─────────┬──────────────────┐
   │   STATE NAMES---->│ RESET   │ VRPRS_RECEIVED   │
   │ INPUT             │ 01      │ 02               │
   ├──────────────────┼─────────┼──────────────────┤
   │ VR_PAC_RSP        │ 2       │ /                │
   │ VR_PAC_RQ         │ /       │ 1                │
   │ 'RESET'           │ -       │ 1                │
   └──────────────────┴─────────┴──────────────────┘

END FSM_VRPRQ_SEND;



FSM_VRPRQ_RCV: FSM_DEFINITION CONTEXT(VRCB);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS FINITE-STATE MACHINE RECORDS THE  RECEIPT OF A VR-SEQUENCED PIU │
   │                 WITH VRPRQ BIT SET  TO VR_PAC_RQ, TO ALLOW THE TRANSMISSION  OF A VR │
   │                 PACING RESPONSE (VRPRS).  IT IS RESET WHEN A VRPRS IS SENT.          │
   │                                                                      │
   │                 THIS FSM  IS CALLED WITH  A "FIRST_VRPRS"  SIGNAL BY THE  VR MANAGER │
   │                 (CHAPTER 12)  WHEN NC_ACTVR  RESPONSE IS  RECEIVED--THIS ALLOWS  THE │
   │                 FIRST  VRPRS TO  BE TRANSMITTED,  WHICH  IN TURN  PERMITS THE  FIRST │
   │                 WINDOW OF PIU'S TO BE TRANSMITTED FROM THE OTHER END OF THE VR.      │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 PC.VRC.VRPRS_SEND                PAGE 3-61           │
   │                 SESSION_RELATED_RCV              PAGE 3-66           │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

   ┌──────────────────┬─────────┬──────────────────┐
   │   STATE NAMES---->│ RESET   │ VRPRQ_RECEIVED   │
   │ INPUT             │ 01      │ 02               │
   ├──────────────────┼─────────┼──────────────────┤
   │ VR_PAC_RQ         │ 2       │ /                │
   │ 'FIRST_VRPRS'     │ 2       │ -                │
   │ 'RESET'           │ -       │ 1                │
   └──────────────────┴─────────┴──────────────────┘

END FSM_VRPRQ_RCV;



FSM_SET_CWRI: FSM_DEFINITION CONTEXT(VRCB);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS FINITE-STATE MACHINE RECORDS THE  RECEIPT OF A VR-SEQUENCED PIU │
   │                 OR A VR PACING  RESPONSE (VRPRS) PIU WITH THE VR_CWI  SET TO DEC_WS, │
   │                 SO THAT A VRPRS MAY BE SENT WITH THE VR_CWRI SET TO DEC_WS_RPLY.  IT │
   │                 IS RESET WHEN A VRPRS WITH THE VR_CWRI SET TO DEC_WS_RPLY IS SENT.   │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 PC.VRC.VRPRS_SEND                PAGE 3-61           │
   │                 SESSION_RELATED_RCV              PAGE 3-66           │
   │                 VRPRS_RCV                        PAGE 3-73           │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

   ┌──────────────────┬─────────┬──────────────────┐
   │   STATE NAMES---->│ RESET   │ SET_CWRI         │
   │ INPUT             │ 01      │ 02               │
   ├──────────────────┼─────────┼──────────────────┤
   │ DEC_WS            │ 2       │ -                │
   │ 'RESET'           │ -       │ 1                │
   └──────────────────┴─────────┴──────────────────┘

END FSM_SET_CWRI;
```

ROUTE EXTENSION PATH CONTROL

BF.PC, PC_T1, and PC_T2 provide for the sending of message units over route extensions between PU_T4 or PU_T5 subarea nodes and adjacent PU_T1 or PU_T2 peripheral nodes.

FID3 PIUs (described in Chapter 2) flow between BF.PC in a subarea node providing boundary function support and PC_T1 in a PU_T1 peripheral node.

FID2 PIUs (described in Chapter 2) flow between BF.PC in a subarea node providing boundary function support and PC_T2 in a PU_T2 peripheral node.

BIUs transmitted by BF.PC may be segmented into multiple BIU segments by an implementation-dependent procedure in BF.PC and transmitted as multiple PIUs.

BIUs transmitted by PC_T1 or PC_T2 may be segmented into multiple BIU segments by an implementation-dependent procedure in PC_T1 or PC_T2, respectively, and transmitted as multiple PIUs.

BF.PC does not perform BIU assembly; BIUs and BIU segments received from PC_T1 or PC_T2 are routed to boundary function half-sessions.

PC_T1 and PC_T2 may--as an implementation option--perform no BIU assembly, BIU assembly on a station basis, or BIU assembly on a session basis; however, only (whole) BIUs are routed to half-sessions in peripheral nodes. If BIU assembly is not supported, BIU segments are rejected.

An additional function performed by PC_T1 and PC_T2 is the routing of PU control point (PUCP) to PU and PU to PUCP BIUs--(PUCP-PU BIUs)--that may flow internal to PU_T1 and PU_T2 nodes. These message units, which provide for node self-activation, are described in Chapter 7.

The detailed procedures and FSMs for BF.PC, PC_T1, and PC_T2 are described next.

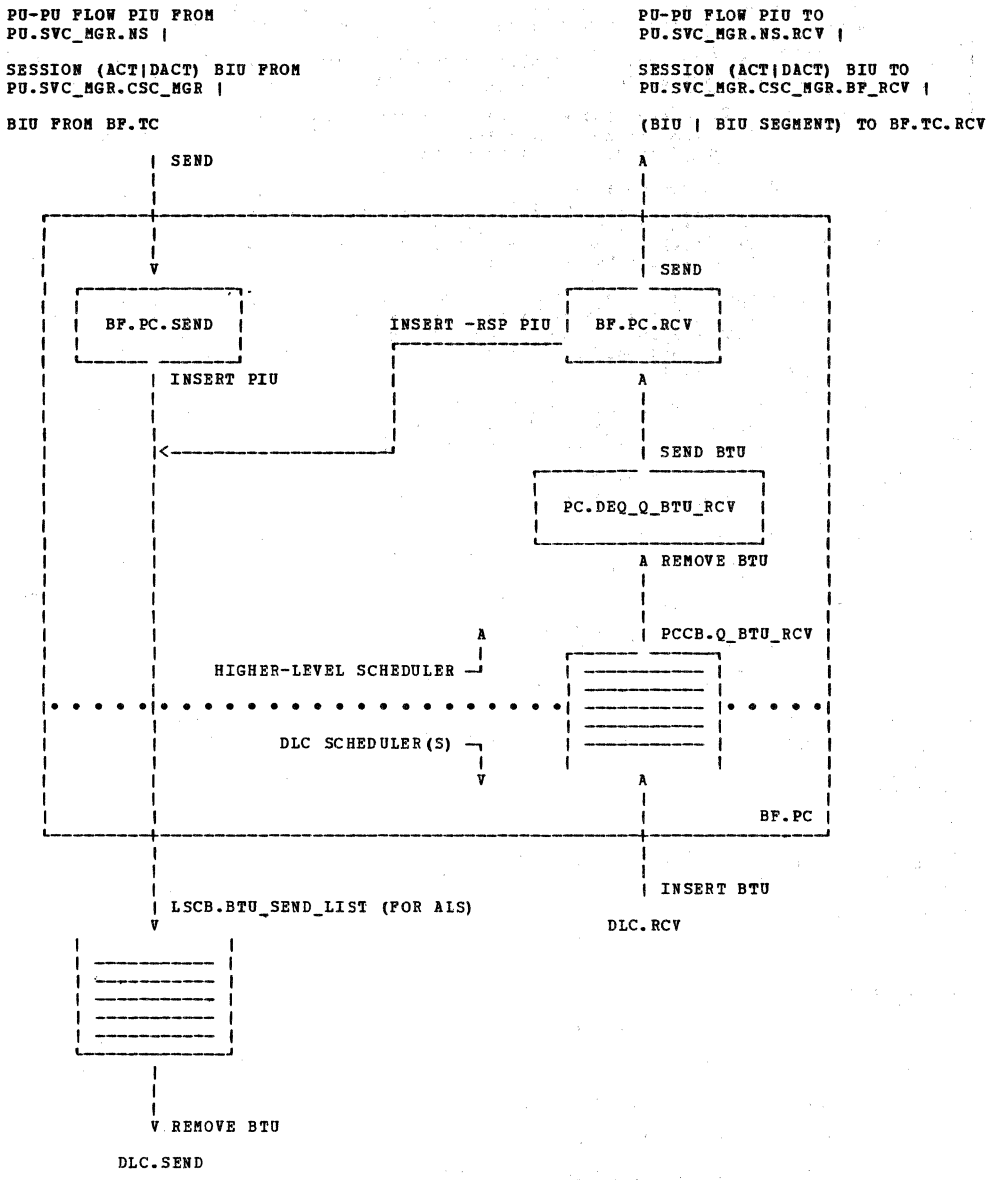


Figure 3-6. Structure of Boundary Function Path Control (BF.PC) for Subarea Nodes

```
BF.PC.SEND: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  THIS PROCEDURE SENDS PIU'S TO PU_T1 OR PU_T2 NODES.   │
   │                                                                      │
   │                IF THE ADJACENT  LINK STATION TO WHICH  THE INPUT MU IS  DESTINED IS
   │                INOPERATIVE, THE MU IS DISCARDED; OTHERWISE:          │
   │                                                                      │
   │                • FOR A PU-PU FLOW PIU FROM THE NETWORK SERVICES COMPONENT OF THE PU
   │                  SERVICES  MANAGER  (CHAPTER  11),  THE  PIU  IS  ENQUEUED  ON  THE
   │                  APPROPRIATE ADJACENT LINK STATION BTU SEND LIST.     │
   │                                                                      │
   │                • FOR A BIU CONTAINING A SESSION ACTIVATION|DEACTIVATION RU FROM THE
   │                  COMMON  SESSION  CONTROL  COMPONENT OF  THE  PU  SERVICES  MANAGER
   │                  (CHAPTER 13), THE  APPROPRIATE PIU TRANSMISSION HEADER  FIELDS ARE
   │                  INITIALIZED, AND THE  PIU IS ENQUEUED ON  THE APPROPRIATE ADJACENT
   │                  LINK STATION BTU SEND LIST.                         │
   │                                                                      │
   │                • FOR A BIU FROM BOUNDARY FUNCTION TRANSMISSION CONTROL (CHAPTER 4),
   │                  THE  BIU MAY  BE SEGMENTED  INTO MULTIPLE  PIU'S, THE  APPROPRIATE
   │                  TRANSMISSION HEADER  FIELDS ARE INITIALIZED  IN EACH PIU,  AND THE
   │                  PIU(S) ARE ENQUEUED  ON THE APPROPRIATE ADJACENT  LINK STATION BTU
   │                  SEND LIST.                                          │
   │                                                                      │
   │     INPUT:     PU-PU FLOW PIU FROM PU.SVC_MGR.NS OR SESSION ACTIVATION|DEACTIVATION
   │                BIU FROM PU.SVC_MGR.CSC_MGR,  WITH ALS LSCB_PTR ESTABLISHED;  OR BIU
   │                FROM BF.TC, WITH SCB_PTR ESTABLISHED.  MU_PTR POINTS TO MU.
   │                                                                      │
   │     OUTPUT:    PIU(S)  ENQUEUED ON  APPROPRIATE ALS  LSCB.BTU_SEND_LIST  IF ALS  IS
   │                OPERATIVE; OTHERWISE, INPUT MU IS DISCARDED.          │
   │                                                                      │
   │     NOTE:      THE TH  ADDRESS FIELD(S), LSID  (FOR  FID3) OR DAFPRIME  AND OAFPRIME
   │                (FOR    FID2),    ARE    ALREADY    INITIALIZED    FOR    BIU'S    FROM
   │                PU.SVC_MGR.CSC_MGR.                                   │
   │                                                                      │
   │     REFERS TO THE FOLLOWING PROCEDURE(S):                            │
   │                ROUTE_EXTENSION_PIU_SEND          PAGE 3-89           │
   │                UPM_ALS_OPERATIVE_CHECK           PAGE 3-97           │
   │                UPM_BIU_SEGMENTER                 PAGE 3-88           │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF DISPATCHED_BY(BF.TC*) THEN                                         /* CHAPTER 4                */
    FIND LSCB IN LSCB_LIST WHERE(LSCB.EA = SCB.BF_ALS_EA);  /* CONNECTION ESTABLISHED AT */
                                                                        /* SESSION ACTIVATION       */
  IF UPM_ALS_OPERATIVE_CHECK = ¬OPERATIVE THEN                          /* PAGE 3-97                */
    DISCARD MU;

  ELSE
    DO;
    . IF DISPATCHED_BY(PU.SVC_MGR.NS.*) THEN                            /* CHAPTER 11               */
    .   CALL ROUTE_EXTENSION_PIU_SEND;                                  /* PAGE 3-89                */
    .
    . ELSE
    .   DO;
    .   . BBIUI = BBIU;
    .   .
    .   . EBIUI = EBIU;
    .   .
    .   . INSERT MU IN PCCB.PIU_SEND_LIST;
    .   .
    .   . CALL UPM_BIU_SEGMENTER;                                       /* PAGE 3-88                */
    .   .
    .   . DO UNTIL EMPTY(PCCB.PIU_SEND_LIST);
    .   . .
    .   . . REMOVE FIRST(MU) FROM PCCB.PIU_SEND_LIST SET(MU_PTR);
    .   . .
    .   . . SELECT ANYORDER(SCB.SUPPORTED_NODE_TYPE);
    .   . . .
    .   . . . WHEN(PU_T1)
    .   . . .   DO;
    .   . . .   . FID = FID3;
    .   . . .   .
    .   . . .   . IF DISPATCHED_BY(BF.TC*) THEN                         /* NOTE                     */
    .   . . .   .   LSID = SCB.LOCAL_SESSION_ID;
    .   . . .   END;
    .   . . .
    .   . . . WHEN(PU_T2)
    .   . . .   DO;
    .   . . .   . FID = FID2;
    .   . . .   .
    .   . . .   . IF DISPATCHED_BY(BF.TC*) THEN                         /* NOTE                     */
    .   . . .   .   DO;
    .   . . .   .   . DAFPRIME = SCB.THIS_ID;
    .   . . .   .   .
    .   . . .   .   . OAFPRIME = SCB.PARTNER_ID;
    .   . . .   .   END;
    .   . . .   END;
    .   . . END;
    .   . .
    .   . . CALL ROUTE_EXTENSION_PIU_SEND;                              /* PAGE 3-89                */
    .   . END;
    .   END;
    END;

  RETURN;

END BF.PC.SEND;
```

```
BF.PC.RCV: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE PROCESSES PIU'S RECEIVED FROM PU_T1 OR PU_T2 NODES.
 │
 │                 • INVALID PIU'S ARE DISCARDED OR RESULT IN A NEGATIVE RESPONSE.
 │
 │                 • PU-PU FLOW PIU'S  ARE ROUTED TO THE NETWORK  SERVICES COMPONENT OF
 │                   THE PU SERVICES MANAGER (CHAPTER 11).
 │
 │                 • BIU'S CONTAINING  SESSION ACTIVATION|DEACTIVATION RU'S  ARE ROUTED
 │                   TO THE COMMON SESSION CONTROL MANAGER COMPONENT OF THE PU SERVICES
 │                   MANAGER (CHAPTER 13).
 │
 │                 • FOR ALL OTHER PIU'S, THE  BOUNDARY FUNCTION HALF-SESSION IS FOUND,
 │                   IF  POSSIBLE,  AND  THE  BIU  IS  ROUTED  TO  BOUNDARY  FUNCTION
 │                   TRANSMISSION CONTROL (CHAPTER 4).
 │
 │      INPUT:     A "BTU" SIGNAL FROM PC.DEQ_Q_BTU_RCV WITH PARM_PTR POINTING TO BTU.
 │
 │      OUTPUT:    PU-PU FLOW PIU TO PU.SVC_MGR.NS.RCV; SESSION ACTIVATION|DEACTIVATION
 │                 BIU  TO PU.SVC_MGR.CSC_MGR.BF_RCV;  BIU TO  BF.TC.RCV, WITH  SCB_PTR
 │                 ESTABLISHED; NEGATIVE RESPONSE GENERATED; OR PIU DISCARDED.
 │
 │      NOTE:      THIS FUNCTION  CONVERTS THE  PIU FROM LINK  FORM TO  CANONICAL FORM;
 │                 THIS IS REQUIRED WITHIN THE ARCHITECTURAL DESCRIPTION, BUT IS NOT AN
 │                 IMPLEMENTATION REQUIREMENT.
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 ROUTE_EXTENSION_NEG_RSP            PAGE 3-99
 │                 ROUTE_EXTENSION_TH_RCV_CHK         PAGE 3-98
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  BTU_PTR = PARM_PTR;

  LSCB_PTR = BTUCB.LSCBPTR;

  IF ROUTE_EXTENSION_TH_RCV_CHK(LSCB.XID_RCV.PU_TYPE) = OK THEN  /* PAGE 3-98          */
    DO;
    . CREATE MU;
    .
    . CALL MAP_TO_CANONICAL(ADDR(BTU_DATA),MU_PTR,BTUCB.BTU_LENGTH);  /* APPENDIX B, NOTE */
    .
    . IF FID = FID2 & DAFPRIME = X'FF' & OAFPRIME = X'00' THEN
    .
    .   IF BBIUI =  ¬BBIU | EBIUI =  ¬EBIU THEN
    .     CALL ROUTE_EXTENSION_NEG_RSP(X'8007');  /* PAGE 3-99, SEGMENTING ERROR      */
    .
    .   ELSE
    .     SEND MU TO PU.SVC_MGR.NS.RCV;                          /* CHAPTER 11       */
    .
    . ELSE
    .   IF BBIUI = BBIU & RU_CTGY = SC & DCF < 4 THEN
    .     CALL ROUTE_EXTENSION_NEG_RSP(X'1002');  /* PAGE 3-99, RU LENGTH ERROR       */
    .
    .   ELSE
    .     IF RU_CTGY = SC &
    .          RQ_CODE = (ACTPU | DACTPU | ACTLU | DACTLU | BIND | UNBIND) THEN
    .
    .       IF BBIUI =  ¬BBIU | EBIUI =  ¬EBIU THEN
    .         CALL ROUTE_EXTENSION_NEG_RSP(X'8007');  /* PAGE 3-99, SEGMENTING ERROR */
    .
    .       ELSE
    .         SEND MU TO PU.SVC_MGR.CSC_MGR.BF_RCV;              /* CHAPTER 13       */
    .
    .     ELSE
    .       DO;
    .       . SELECT ANYORDER(FID);
    .       . .
    .       . . WHEN(FID3)
    .       . .   FIND SCB IN SCB_LIST
    .       . .        WHERE(SCB.SCB_TYPE = BF_SESS &
    .       . .        SCB.LOCAL_SESSION_ID = LSID &
    .       . .        SCB.BF_ALS_EA = LSCB.EA);
    .       . .
    .       . . WHEN(FID2)
    .       . .   FIND SCB IN SCB_LIST
    .       . .        WHERE(SCB.SCB_TYPE = BF_SESS &
    .       . .        SCB.PARTNER_ID = DAFPRIME &
    .       . .        SCB.THIS_ID = OAFPRIME &
    .       . .        SCB.BF_ALS_EA = LSCB.EA);
    .       . END;
    .       .
    .       . IF SCB_PTR = NULL THEN
    .       .   CALL ROUTE_EXTENSION_NEG_RSP(X'8005');          /* PAGE 3-99, NO SESSION */
    .       .
    .       . ELSE
    .       .   SEND MU TO BF.TC.RCV;                           /* CHAPTER 4        */
    .       END;
    END;

  DISCARD BTU;

  RETURN;

END BF.PC.RCV;
```

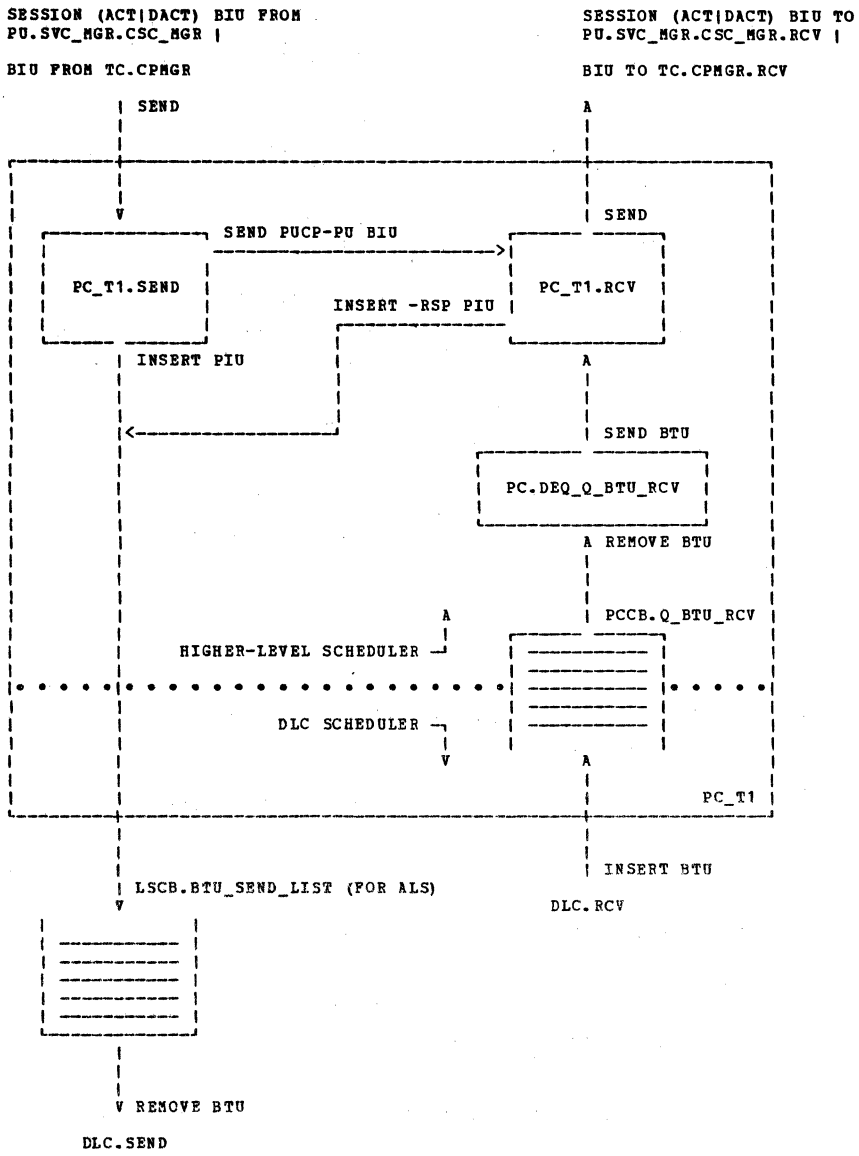


Figure 3-7. Structure of Path Control for PU_T1 Node (PC_T1)

```
PC_T1.RCV: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE PROCESSES  PIU'S RECEIVED AT A PU_T1  NODE AND ROUTES
│                 PUCP-PU BIU'S THAT FLOW INTERNAL TO A PU_T1 NODE.
│
│                 • INVALID  PIU'S|BIU'S  ARE  DISCARDED  OR  RESULT  IN  A  NEGATIVE
│                   RESPONSE.
│
│                 • BIU'S CONTAINING  SESSION ACTIVATION|DEACTIVATION RU'S  ARE ROUTED
│                   TO THE COMMON SESSION CONTROL COMPONENT OF THE PU SERVICES MANAGER
│                   (CHAPTER 13).
│
│                 • IF SUPPORTED AND REQUIRED, BIU ASSEMBLY IS PERFORMED.
│
│                 • FOR BIU'S DESTINED FOR A  HALF-SESSION, THE HALF-SESSION IS FOUND,
│                   IF  POSSIBLE,  AND  THE  BIU IS  ROUTED  TO  TRANSMISSION  CONTROL
│                   CONNECTION POINT MANAGER (CHAPTER 4).
│
│      INPUT:     A "BTU" SIGNAL FROM PC.DEQ_Q_BTU_RCV  WITH PARM_PTR POINTING TO BTU,
│                 OR A PUCP-PU BIU FROM PC_T1.SEND WITH MU-PTR POINTING TO BIU
│
│      OUTPUT:    BIU  CONTAINING SESSION  ACTIVATION|DEACTIVATION RU,  POINTED TO  BY
│                 MU_PTR, TO PU.SVC_MGR.CSC_MGR.RCV; OR BIU,  POINTED TO BY MU_PTR, TO
│                 TC.CPMGR.RCV; OR  BIU SEGMENT SAVED OR  APPENDED AS PART  OF PARTIAL
│                 BIU; OR  PIU, BIU,  OR PARTIAL BIU  DISCARDED; OR  NEGATIVE RESPONSE
│                 GENERATED  RELATIVE  TO  PIU,  BIU,  OR  PARTIAL  BIU.   SCB_PTR  IS
│                 ESTABLISHED FOR BIU'S ROUTED TO TC.CPMGR.RCV.
│
│      NOTE:      THIS FUNCTION  CONVERTS THE  PIU FROM LINK  FORM TO  CANONICAL FORM;
│                 THIS IS REQUIRED WITHIN THE ARCHITECTURAL DESCRIPTION, BUT IS NOT AN
│                 IMPLEMENTATION REQUIREMENT.
│
│      REFERS TO THE FOLLOWING PROCEDURE(S):
│                 ROUTE_EXTENSION_NEG_RSP            PAGE 3-99
│                 ROUTE_EXTENSION_TH_RCV_CHK         PAGE 3-98
│                 T1_OR_T2_NO_BIU_ASSEMBLY_RCV_CK    PAGE 3-91
│                 T1_OR_T2_SESSION_BIU_ASSEMBLER     PAGE 3-94
│                 T1_OR_T2_STATION_BIU_ASSEMBLER     PAGE 3-92
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL ASSEMBLER_RESULT BIT(1);

  IF DISPATCHED_BY(PC_T1.SEND) THEN                          /* ROUTE PUCP-PU BIU      */
    DO;
    . MUCB.DIRECTION = RECEIVE;
    .
    . IF RQ_CODE = ACTPU THEN
    .   SEND MU TO PU.SVC_MGR.CSC_MGR.RCV;                   /* CHAPTER 13             */
    .
    . ELSE
    .   DO;
    .   . SELECT ANYORDER(MUCB.PUCP_BASED_SESSION);
    .   . .
    .   . . WHEN(PUCP_TO_PU)
    .   . .   FIND SCB IN SCB_LIST
    .   . .        WHERE(SCB.TYPE_OF_SESSION = PUCP_PU & SCB.HALF_SESSION = SEC);
    .   . .
    .   . . WHEN(PU_TO_PUCP)
    .   . .   FIND SCB IN SCB_LIST
    .   . .        WHERE(SCB.TYPE_OF_SESSION = PUCP_PU & SCB.HALF_SESSION = PRI);
    .   . .
    .   . END;
    .   .
    .   . SEND MU TO TC.CPMGR.RCV;                           /* CHAPTER 4              */
    .   END;
    END;
```

```
ELSE                                                          /* PROCESS RECEIVED PIU     */
  DO;
  . BTU_PTR = PARM_PTR;
  .
  . LSCB_PTR = BTUCB.LSCBPTR;
  .
  . IF ROUTE_EXTENSION_TH_RCV_CHK(PU_T1) = OK THEN            /* PAGE 3-98                */
  .   DO;
  .   . CREATE MU;
  .   .
  .   . CALL MAP_TO_CANONICAL(ADDR(BTU_DATA),MU_PTR,BTUCB.BTU_LENGTH);  /* APPENDIX B      */
  .   .                                                       /* NOTE                     */
  .   . SELECT ANYORDER(PCCB.BIU_ASSEMBLY_OPTION);
  .   . .
  .   . . WHEN(NO_ASSEMBLY)
  .   . .   IF T1_OR_T2_NO_BIU_ASSEMBLY_RCV_CK = OK THEN      /* PAGE 3-91                */
  .   . .
  .   . .     IF RU_CTGY = SC & RQ_CODE = (ACTLU | DACTLU | BIND | UNBIND) THEN
  .   . .       SEND MU TO PU.SVC_MGR.CSC_MGR.RCV;          /* CHAPTER 13               */
  .   . .
  .   . .     ELSE
  .   . .       DO;
  .   . .       . FIND SCB IN SCB_LIST WHERE(SCB.LOCAL_SESSION_ID = LSID);
  .   . .       .
  .   . .       . IF SCB_PTR = NULL THEN
  .   . .       .   CALL ROUTE_EXTENSION_NEG_RSP(X'8005');  /* PAGE 3-99, NO SESSION    */
  .   . .       .
  .   . .       . ELSE
  .   . .       .   SEND MU TO TC.CPMGR.RCV;                /* CHAPTER 4                */
  .   . .       END;
  .   . .
  .   . . WHEN(STATION_ASSEMBLY)
  .   . .   DO;
  .   . .   . CALL T1_OR_T2_STATION_BIU_ASSEMBLER(ASSEMBLER_RESULT);  /* PAGE 3-92        */
  .   . .   .
  .   . .   . IF ASSEMBLER_RESULT = BIU_AVAILABLE THEN
  .   . .   .
  .   . .   .   IF RU_CTGY = SC & RQ_CODE = (ACTLU | DACTLU | BIND | UNBIND) THEN
  .   . .   .     SEND MU TO PU.SVC_MGR.CSC_MGR.RCV;        /* CHAPTER 13               */
  .   . .   .
  .   . .   .   ELSE
  .   . .   .     DO;
  .   . .   .     . FIND SCB IN SCB_LIST WHERE(SCB.LOCAL_SESSION_ID = LSID);
  .   . .   .     .
  .   . .   .     . IF SCB_PTR = NULL THEN
  .   . .   .     .   CALL ROUTE_EXTENSION_NEG_RSP(X'8005');  /* PAGE 3-99, NO SESSION    */
  .   . .   .     .
  .   . .   .     . ELSE
  .   . .   .     .   SEND MU TO TC.CPMGR.RCV;               /* CHAPTER 4                */
  .   . .   .     END;
  .   . .   END;
  .   . .
  .   . . WHEN(SESSION_ASSEMBLY)
  .   . .   IF BBIUI = BBIU & RU_CTGY = SC & DCF < 4 THEN
  .   . .     CALL ROUTE_EXTENSION_NEG_RSP(X'1002');  /* PAGE 3-99, RU LENGTH ERROR       */
  .   . .
  .   . .   ELSE
  .   . .     IF RU_CTGY = SC &
  .   . .          RQ_CODE = (ACTLU | DACTLU | BIND | UNBIND) THEN
  .   . .
  .   . .       IF BBIUI =  ¬BBIU | EBIUI =  ¬EBIU THEN
  .   . .         CALL ROUTE_EXTENSION_NEG_RSP(X'8007');  /* PAGE 3-99, SEGMENTING ERROR  */
  .   . .
  .   . .       ELSE
  .   . .         SEND MU TO PU.SVC_MGR.CSC_MGR.RCV;        /* CHAPTER 13               */
  .   . .
  .   . .     ELSE
  .   . .       DO;
  .   . .       . FIND SCB IN SCB_LIST
  .   . .       .      WHERE(SCB.LOCAL_SESSION_ID = LSID);
  .   . .       .
  .   . .       . IF SCB_PTR = NULL THEN
  .   . .       .   CALL ROUTE_EXTENSION_NEG_RSP(X'8005');  /* PAGE 3-99, NO SESSION    */
  .   . .       .
  .   . .       . ELSE
  .   . .       .   DO;
  .   . .       .   . CALL T1_OR_T2_SESSION_BIU_ASSEMBLER(ASSEMBLER_RESULT);  /* PAGE 3-94 */
  .   . .       .   .
  .   . .       .   . IF ASSEMBLER_RESULT = BIU_AVAILABLE THEN
  .   . .       .   .   SEND MU TO TC.CPMGR.RCV;            /* CHAPTER 4                */
  .   . .       .   END;
  .   . .       END;
  .   . END;
  .   END;
  .
  . DISCARD BTU;
  END;

RETURN;

END PC_T1.RCV;
```

```
PC_T1.SEND: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE SENDS PIU'S FROM A PU_T1 NODE AND ROUTES PUCP-PU
   │                 BIU'S THAT FLOW INTERNAL TO A PU_T1 NODE.
   │
   │                 PUCP-PU BIU'S ARE ROUTED TO PC_T1.RCV, ALL OTHER BIU'S ARE PROCESSED
   │                 AS FOLLOWS.
   │
   │                 IF THE ADJACENT LINK STATION IS INOPERATIVE, THE INPUT BIU IS
   │                 DISCARDED; OTHERWISE:
   │
   │                 • FOR A BIU CONTAINING A SESSION ACTIVATION|DEACTIVATION RU FROM THE
   │                   COMMON SESSION CONTROL COMPONENT OF THE PU SERVICES MANAGER
   │                   (CHAPTER 13), THE APPROPRIATE PIU TRANSMISSION HEADER FIELDS ARE
   │                   INITIALIZED, AND THE PIU IS ENQUEUED ON THE ADJACENT LINK STATION
   │                   BTU SEND LIST.
   │
   │                 • FOR A BIU FROM TRANSMISSION CONTROL CONNECTION POINT MANAGER
   │                   (CHAPTER 4), THE BIU MAY BE SEGMENTED INTO MULTIPLE PIU'S, THE
   │                   APPROPRIATE TRANSMISSION HEADER FIELDS ARE INITIALIZED IN EACH
   │                   PIU, AND THE PIU(S) ARE ENQUEUED ON THE ADJACENT LINK STATION BTU
   │                   SEND LIST.
   │
   │      INPUT:     SESSION ACTIVATION|DEACTIVATION BIU FROM PU.SVC_MGR.CSC_MGR OR BIU
   │                 FROM TC.CPMGR WITH SCB_PTR ESTABLISHED. MU_PTR POINTS TO BIU.
   │
   │      OUTPUT:    PUCP-PU BIU, POINTED TO BY MU_PTR, TO PC_T1.RCV; OR PIU(S) ENQUEUED
   │                 ON ALS LSCB.BTU_SEND_LIST IF ALS IS OPERATIVE; OTHERWISE, INPUT BIU
   │                 IS DISCARDED.
   │
   │      NOTE:      LSID IS ALREADY INITIALIZED FOR BIU FROM PU.SVC_MGR.CSC_MGR.
   │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):
   │                 ROUTE_EXTENSION_PIU_SEND          PAGE 3-89
   │                 UPM_ALS_OPERATIVE_CHECK           PAGE 3-97
   │                 UPM_BIU_SEGMENTER                 PAGE 3-88
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF SCB_PTR ¬= NULL & SCB.TYPE_OF_SESSION = PUCP_PU THEN
    DO;                                                 /* ROUTE PUCP-PU BIU    */
    . SELECT ANYORDER(SCB.HALF_SESSION);
    . .
    . . WHEN(PRI)
    . .   MUCB.PUCP_BASED_SESSION = PUCP_TO_PU;
    . .
    . . WHEN(SEC)
    . .   MUCB.PUCP_BASED_SESSION = PU_TO_PUCP;
    . .
    . END;
    .
    . SEND MU TO PC_T1.RCV;                             /* PAGE 3-80            */
    END;

  ELSE                                                  /* SEND PIU             */
    DO;
    . FIND LSCB IN LSCB_LIST WHERE(LSCB.LSCB_TYPE = ALS);  /* DEFINED AT SYSTEM DEFINITION */
    .
    . IF UPM_ALS_OPERATIVE_CHECK = ¬OPERATIVE THEN      /* PAGE 3-97            */
    .   DISCARD MU;
    .
    . ELSE
    .   DO;
    .   . BBIUI = BBIU;
    .   .
    .   . EBIUI = EBIU;
    .   .
    .   . INSERT MU IN PCCB.PIU_SEND_LIST;
    .   .
    .   . CALL UPM_BIU_SEGMENTER;                      /* PAGE 3-88            */
    .   .
    .   . DO UNTIL EMPTY(PCCB.PIU_SEND_LIST);
    .   . .
    .   . . REMOVE FIRST(MU) FROM PCCB.PIU_SEND_LIST SET(MU_PTR);
    .   . .
    .   . . FID = FID3;
    .   . .
    .   . . IF DISPATCHED_BY(TC.CPMGR*) THEN           /* NOTE                 */
    .   . .   LSID = SCB.LOCAL_SESSION_ID;
    .   . .
    .   . . CALL ROUTE_EXTENSION_PIU_SEND;             /* PAGE 3-89            */
    .   . END;
    .   END;
    END;

  RETURN;

END PC_T1.SEND;
```

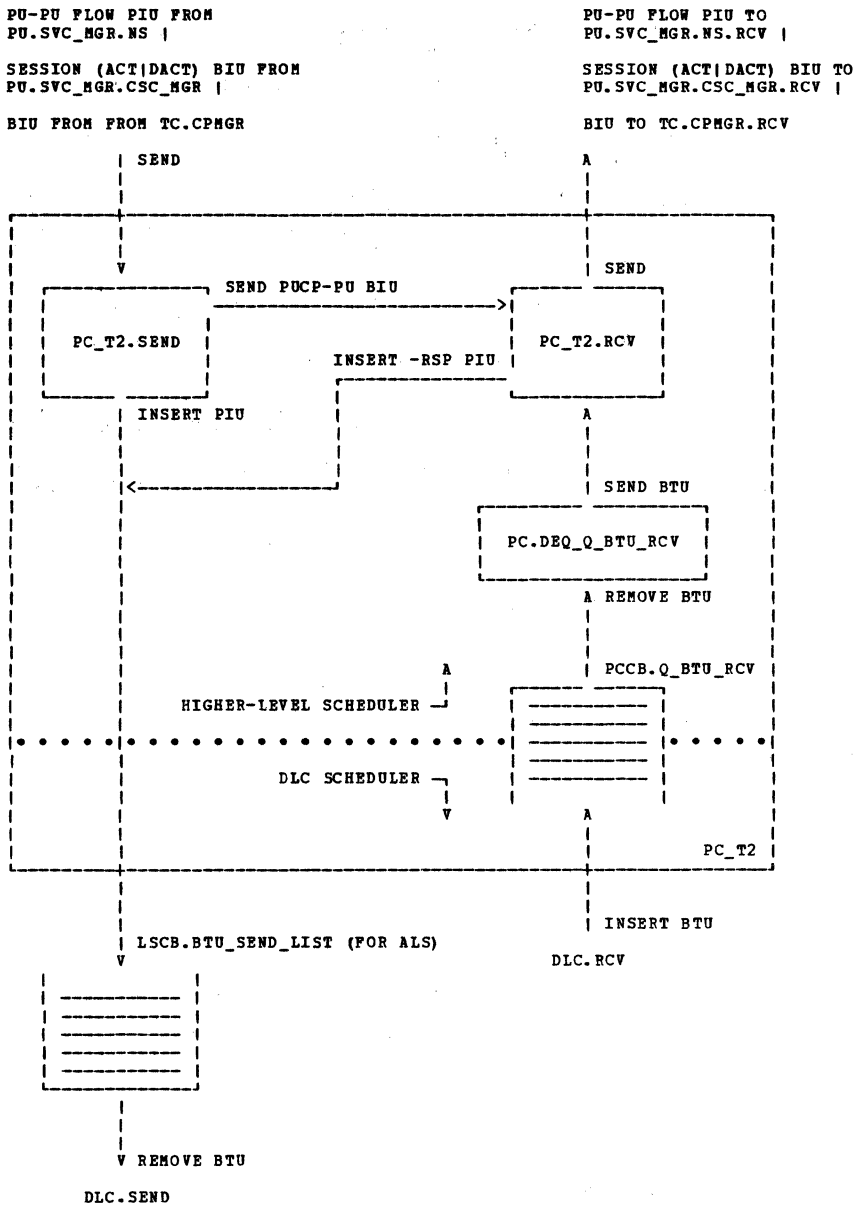


Figure 3-8. Structure of Path Control for PU_T2 Node (PC_T2)

```
PC_T2.RCV: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE PROCESSES  PIU'S RECEIVED AT A PU_T2  NODE AND ROUTES
│                 PUCP-PU BIU'S THAT FLOW INTERNAL TO A PU_T2 NODE.
│
│                 • INVALID  PIU'S|BIU'S  ARE  DISCARDED  OR  RESULT  IN  A  NEGATIVE
│                   RESPONSE.
│
│                 • PU-PU FLOW PIU'S  ARE ROUTED TO THE NETWORK  SERVICES COMPONENT OF
│                   THE PU SERVICES MANAGER (CHAPTER 11).
│
│                 • BIU'S CONTAINING  SESSION ACTIVATION|DEACTIVATION RU'S  ARE ROUTED
│                   TO THE COMMON SESSION CONTROL COMPONENT OF THE PU SERVICES MANAGER
│                   (CHAPTER 13).
│
│                 • IF SUPPORTED AND REQUIRED, BIU ASSEMBLY IS PERFORMED.
│
│                 • FOR BIU'S DESTINED FOR A  HALF-SESSION, THE HALF-SESSION IS FOUND,
│                   IF  POSSIBLE,  AND  THE  BIU  IS  ROUTED  TO  TRANSMISSION  CONTROL
│                   CONNECTION POINT MANAGER (CHAPTER 4).
│
│      INPUT:     A "BTU" SIGNAL FROM PC.DEQ_Q_BTU_RCV  WITH PARM_PTR POINTING TO BTU,
│                 OR A PUCP-PU BIU FROM PC_T2.SEND WITH MU_PTR POINTING TO BIU.
│
│      OUTPUT:    PU-PU  FLOW PIU,  POINTED  TO BY  MU_PTR,  TO PU.SVC_MGR.NS.RCV;  OR
│                 SESSION  ACTIVATION|DEACTIVATION  BIU,  POINTED  TO  BY  MU_PTR,  TO
│                 PU.SVC_MGR.CSC_MGR.RCV;  OR  BIU,  POINTED  TO  BY  MU_PTR,  TO
│                 TC.CPMGR.RCV; OR  BIU SEGMENT SAVED OR  APPENDED AS PART  OF PARTIAL
│                 BIU; OR  PIU, BIU,  OR PARTIAL BIU  DISCARDED; OR  NEGATIVE RESPONSE
│                 GENERATED  RELATIVE  TO  PIU,  BIU,  OR  PARTIAL  BIU.   SCB_PTR  IS
│                 ESTABLISHED FOR BIU'S ROUTED TO TC.CPMGR.RCV.
│
│      NOTE:      THIS FUNCTION  CONVERTS THE  PIU FROM LINK  FORM TO  CANONICAL FORM;
│                 THIS IS REQUIRED WITHIN THE ARCHITECTURAL DESCRIPTION, BUT IS NOT AN
│                 IMPLEMENTATION REQUIREMENT.
│
│      REFERS TO THE FOLLOWING PROCEDURE(S):
│                 ROUTE_EXTENSION_NEG_RSP            PAGE 3-99
│                 ROUTE_EXTENSION_TH_RCV_CHK         PAGE 3-98
│                 T1_OR_T2_NO_BIU_ASSEMBLY_RCV_CK    PAGE 3-91
│                 T1_OR_T2_SESSION_BIU_ASSEMBLER     PAGE 3-94
│                 T1_OR_T2_STATION_BIU_ASSEMBLER     PAGE 3-92
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL ASSEMBLER_RESULT BIT(1);

IF DISPATCHED_BY(PC_T2.SEND) THEN                       /* ROUTE PUCP-PU BIU   */
  DO;
  . MUCB.DIRECTION = RECEIVE;
  .
  . IF RQ_CODE = ACTPU THEN
  .   SEND MU TO PU.SVC_MGR.CSC_MGR.RCV;                /* PAGE 13             */
  .
  . ELSE
  .   DO;
  .   . SELECT ANYORDER(MUCB.PUCP_BASED_SESSION);
  .   . .
  .   . . WHEN(PUCP_TO_PU)
  .   . .   FIND SCB IN SCB_LIST
  .   . .        WHERE(SCB.TYPE_OF_SESSION = PUCP_PU & SCB.HALF_SESSION = SEC);
  .   . .
  .   . . WHEN(PU_TO_PUCP)
  .   . .   FIND SCB IN SCB_LIST
  .   . .        WHERE(SCB.TYPE_OF_SESSION = PUCP_PU & SCB.HALF_SESSION = PRI);
  .   . .
  .   . END;
  .   .
  .   . SEND MU TO TC.CPMGR.RCV;                        /* CHAPTER 4           */
  .   END;
  END;

ELSE                                                    /* PROCESS RECEIVED PIU */
  DO;
  . BTU_PTR = PARM_PTR;
  .
  . LSCB_PTR = BTUCB.LSCBPTR;
  .
  . IF ROUTE_EXTENSION_TH_RCV_CHK(PU_T2) = OK THEN      /* PAGE 3-98           */
  .   DO;
  .   . CREATE MU;
  .   .
  .   . CALL MAP_TO_CANONICAL(ADDR(BTU_DATA),MU_PTR,BTUCB.BTU_LENGTH);  /* APPENDIX B */
  .   .                                                 /* NOTE                */
  .   . SELECT ANYORDER(PCCB.BIU_ASSEMBLY_OPTION);
```

```
. . . WHEN(NO_ASSEMBLY)
. . .   IF T1_OR_T2_NO_BIU_ASSEMBLY_RCV_CK = OK THEN        /* PAGE 3-91                */
. . .
. . .     IF DAFPRIME = X'00' & OAFPRIME = X'FF' THEN
. . .       SEND MU TO PU.SVC_MGR.NS.RCV;                  /* CHAPTER 11               */
. . .
. . .     ELSE
. . .       IF RU_CTGY = SC & RQ_CODE = (ACTLU | DACTLU | BIND | UNBIND) THEN
. . .         SEND MU TO PU.SVC_MGR.CSC_MGR.RCV;           /* CHAPTER 13               */
. . .
. . .       ELSE
. . .         DO;
. . .         . FIND SCB IN SCB_LIST
. . .         .      WHERE(SCB.THIS_ID = DAFPRIME & SCB.PARTNER_ID = OAFPRIME);
. . .         .
. . .         . IF SCB_PTR = NULL THEN
. . .         .   CALL ROUTE_EXTENSION_NEG_RSP(X'8005');   /* PAGE 3-99, NO SESSION    */
. . .         .
. . .         . ELSE
. . .         .   SEND MU TO TC.CPMGR.RCV;                 /* CHAPTER 4                */
. . .           END;
. . .
. . . WHEN(STATION_ASSEMBLY)
. . .   DO;
. . .   . CALL T1_OR_T2_STATION_BIU_ASSEMBLER(ASSEMBLER_RESULT);  /* PAGE 3-92          */
. . .   .
. . .   . IF ASSEMBLER_RESULT = BIU_AVAILABLE THEN
. . .   .
. . .   .   IF DAFPRIME = X'00' & OAFPRIME = X'FF' THEN
. . .   .     SEND MU TO PU.SVC_MGR.NS.RCV;                /* CHAPTER 11               */
. . .   .
. . .   .     ELSE
. . .   .       IF RU_CTGY = SC &
. . .   .            RQ_CODE = (ACTPU | DACTPU | ACTLU | DACTLU | BIND | UNBIND) THEN
. . .   .         SEND MU TO PU.SVC_MGR.CSC_MGR.RCV;       /* CHAPTER 13               */
. . .   .
. . .   .       ELSE
. . .   .         DO;
. . .   .         . FIND SCB IN SCB_LIST
. . .   .         .      WHERE(SCB.THIS_ID = DAFPRIME & SCB.PARTNER_ID = OAFPRIME);
. . .   .         .
. . .   .         . IF SCB_PTR = NULL THEN
. . .   .         .   CALL ROUTE_EXTENSION_NEG_RSP(X'8005');  /* PAGE 3-99, NO SESSION  */
. . .   .         .
. . .   .         . ELSE
. . .   .         .   SEND MU TO TC.CPMGR.RCV;              /* CHAPTER 4                */
. . .   .           END;
. . .   END;
. . .
. . . WHEN(SESSION_ASSEMBLY)
. . .   IF DAFPRIME = X'00' & OAFPRIME = X'FF' THEN
. . .
. . .     IF BBIUI =  ¬BBIU | EBIUI =  ¬EBIU THEN
. . .       CALL ROUTE_EXTENSION_NEG_RSP(X'8007');  /* PAGE 3-99, SEGMENTING ERROR     */
. . .
. . .     ELSE
. . .       SEND MU TO PU.SVC_MGR.NS.RCV;                  /* CHAPTER 11               */
. . .
. . .   ELSE
. . .     IF BBIUI = BBIU & RU_CTGY = SC & DCF < 4 THEN
. . .       CALL ROUTE_EXTENSION_NEG_RSP(X'1002');  /* PAGE 3-99, RU LENGTH ERROR      */
. . .
. . .     ELSE
. . .       IF RU_CTGY = SC &
. . .            RQ_CODE = (ACTPU | DACTPU | ACTLU | DACTLU | BIND | UNBIND) THEN
. . .
. . .         IF BBIUI =  ¬BBIU | EBIUI =  ¬EBIU THEN
. . .           CALL ROUTE_EXTENSION_NEG_RSP(X'8007');  /* PAGE 3-99, SEGMENTING ERROR */
. . .
. . .         ELSE
. . .           SEND MU TO PU.SVC_MGR.CSC_MGR.RCV;         /* CHAPTER 13               */
. . .
. . .       ELSE
. . .         DO;
. . .         . FIND SCB IN SCB_LIST
. . .         .      WHERE(SCB.THIS_ID = DAFPRIME & SCB.PARTNER_ID = OAFPRIME);
. . .         .
. . .         . IF SCB_PTR = NULL THEN
. . .         .   CALL ROUTE_EXTENSION_NEG_RSP(X'8005');   /* PAGE 3-99, NO SESSION    */
. . .         .
. . .         . ELSE
. . .         .   DO;
. . .         .   . CALL T1_OR_T2_SESSION_BIU_ASSEMBLER(ASSEMBLER_RESULT);  /*
. . .         .   .                                                PAGE 3-94           */
. . .         .   . IF ASSEMBLER_RESULT = BIU_AVAILABLE THEN
. . .         .   .   SEND MU TO TC.CPMGR.RCV;             /* CHAPTER 4                */
. . .         .     END;
. . .           END;
.   . END;
.   END;
.
. DISCARD BTU;
  END;

RETURN;

END PC_T2.RCV;
```

```
PC_T2.SEND: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│                                                                          │
│     FUNCTION:  THIS PROCEDURE SENDS PIU'S FROM A PU_T2 NODE AND ROUTES   │
│                PUCP-PU BIU'S THAT FLOW INTERNAL TO A PU_T2 NODE.         │
│                                                                          │
│                PUCP-PU BIU'S ARE ROUTED TO PC_T2.RCV, ALL OTHER MU'S ARE │
│                PROCESSED AS FOLLOWS.                                     │
│                                                                          │
│                IF THE ADJACENT LINK STATION IS INOPERATIVE, THE INPUT MU │
│                IS DISCARDED; OTHERWISE:                                  │
│                                                                          │
│                • FOR A PU-PU FLOW PIU FROM THE NETWORK SERVICES          │
│                  COMPONENT OF THE PU SERVICES MANAGER (CHAPTER 11), THE  │
│                  PIU IS ENQUEUED ON THE ADJACENT LINK STATION BTU SEND   │
│                  LIST.                                                   │
│                                                                          │
│                • FOR A BIU CONTAINING A SESSION ACTIVATION|DEACTIVATION  │
│                  RU FROM THE COMMON SESSION CONTROL MANAGER COMPONENT OF  │
│                  THE PU SERVICES MANAGER (CHAPTER 13), THE APPROPRIATE   │
│                  PIU TRANSMISSION HEADER FIELDS ARE INITIALIZED, AND THE │
│                  PIU IS ENQUEUED ON THE ADJACENT LINK STATION BTU SEND   │
│                  LIST.                                                   │
│                                                                          │
│                • FOR A BIU FROM TRANSMISSION CONTROL CONNECTION POINT    │
│                  MANAGER (CHAPTER 4), THE BIU MAY BE SEGMENTED INTO      │
│                  MULTIPLE PIU'S, THE APPROPRIATE TRANSMISSION HEADER     │
│                  FIELDS ARE INITIALIZED IN EACH PIU, AND THE PIU(S) ARE  │
│                  ENQUEUED ON THE ADJACENT LINK STATION BTU SEND LIST.     │
│                                                                          │
│     INPUT:     PU-PU FLOW PIU FROM PU.SVC_MGR.NS, OR SESSION             │
│                ACTIVATION|DEACTIVATION BIU FROM PU.SVC_MGR.CSC_MGR; OR   │
│                BIU FROM TC.CPMGR, WITH SCB_PTR ESTABLISHED. MU_PTR       │
│                POINTS TO MU.                                             │
│                                                                          │
│     OUTPUT:    PUCP-PU BIU, POINTED TO BY MU-PTR, TO PC_T2.RCV; OR PIU(S) │
│                ENQUEUED ON ALS LSCB.BTU_SEND_LIST IF ALS IS OPERATIVE;   │
│                OTHERWISE, INPUT MU IS DISCARDED.                         │
│                                                                          │
│     NOTE:      DAFPRIME AND OAFPRIME ARE ALREADY INITIALIZED FOR BIU     │
│                FROM PU.SVC_MGR.CSC_MGR.                                  │
│                                                                          │
│     REFERS TO THE FOLLOWING PROCEDURE(S):                                │
│                ROUTE_EXTENSION_PIU_SEND          PAGE 3-89               │
│                UPM_ALS_OPERATIVE_CHECK           PAGE 3-97               │
│                UPM_BIU_SEGMENTER                 PAGE 3-88               │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

```
IF SCB_PTR ¬= NULL & SCB.TYPE_OF_SESSION = PUCP_PU THEN
  DO;                                                          /* ROUTE PUCP-PU BIU        */
  . SELECT ANYORDER(SCB.HALF_SESSION);
  . .
  . . WHEN(PRI)
  . .   MUCB.PUCP_BASED_SESSION = PUCP_TO_PU;
  . .
  . . WHEN(SEC)
  . .   MUCB.PUCP_BASED_SESSION = PU_TO_PUCP;
  . .
  . END;
  .
  . SEND MU TO PC_T2.RCV;                                      /* PAGE 3-84                */
  END;

ELSE                                                           /* SEND PIU                 */
  DO;
  . FIND LSCB IN LSCB_LIST WHERE(LSCB.LSCB_TYPE = ALS);  /* DEFINED AT SYSTEM DEFINITION   */
  .
  . IF UPM_ALS_OPERATIVE_CHECK =  ¬OPERATIVE THEN              /* PAGE 3-97                */
  .   DISCARD MU;
  .
  . ELSE
  .   DO;
  .   . IF DISPATCHED_BY(PU.SVC_MGR.NS_*) THEN                /* CHAPTER 11               */
  .   .   CALL ROUTE_EXTENSION_PIU_SEND;                      /* PAGE 3-89                */
  .   .
  .   . ELSE
  .   .   DO;
  .   .   . BBIUI = BBIU;
  .   .   .
  .   .   . EBIUI = EBIU;
  .   .   .
  .   .   . INSERT MU IN PCCB.PIU_SEND_LIST;
  .   .   .
  .   .   . CALL UPM_BIU_SEGMENTER;                           /* PAGE 3-88                */
  .   .   .
  .   .   . DO UNTIL EMPTY(PCCB.PIU_SEND_LIST);
  .   .   . .
  .   .   . . REMOVE FIRST(MU) FROM PCCB.PIU_SEND_LIST SET(MU_PTR);
  .   .   . .
  .   .   . . FID = FID2;
  .   .   . .
  .   .   . . IF DISPATCHED_BY(TC.CPMGR*) THEN                /* NOTE                     */
  .   .   . .   DO;
  .   .   . .   . DAFPRIME = SCB.PARTNER_ID;
  .   .   . .   .
  .   .   . .   . OAFPRIME = SCB.THIS_ID;
  .   .   . .   END;
  .   .   . .
  .   .   . . CALL ROUTE_EXTENSION_PIU_SEND;                  /* PAGE 3-89                */
  .   .   . END;
  .   .   END;
  .   END;
  END;

RETURN;

END PC_T2.SEND;
```

```
PC.DEQ_Q_BTU_RCV: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE  DEQUEUES A  BTU FROM  THE PATH  CONTROL BTU  RECEIVE
   │                 QUEUE AND ROUTES IT TO THE PATH CONTROL RECEIVE COMPONENT APPLICABLE
   │                 TO THE NODE.
   │
   │                 • FOR PU_T4 OR PU_T5 NODES, THE BTU IS ROUTED TO BF.PC.RCV.
   │
   │                 • FOR PU_T1 NODES, THE BTU IS ROUTED TO PC_T1.RCV.
   │
   │                 • FOR PU_T2 NODES, THE BTU IS ROUTED TO PC_T2.RCV.
   │
   │      INPUT:     AN  OPEN_QUEUE SIGNAL  FROM HIGHER-LEVEL  SCHEDULER.  FOR PU_T4  OR
   │                 PU_T5 NODES,  PCCB.Q_BTU_RCV CONTAINS BTU(S)  RECEIVED VIA  DLC FROM
   │                 PU_T1 OR  PU_T2 NODES.  FOR A PU_T1  OR PU_T2  NODE, PCCB.Q_BTU_RCV
   │                 CONTAINS BTU(S) RECEIVED  VIA DLC FROM BOUNDARY FUNCTION  IN A PU_T4
   │                 OR PU_T5 NODE.
   │
   │      OUTPUT:    A  "BTU" SIGNAL  TO  BF.PC.RCV, PC_T1.RCV,  OR  PC_T2.RCV, WITH  THE
   │                 PARM_PTR POINTING TO BTU.
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  LOCK PCCB.Q_BTU_RCV;
  .
  . REMOVE FIRST(BTU) FROM PCCB.Q_BTU_RCV SET(BTU_PTR);
  .
  UNLOCK;

  SELECT ANYORDER(NCB.PU_TYPE);
  .
  . WHEN(PU_T4 | PU_T5)
  .   SEND 'BTU' TO BF.PC.RCV USING(PARM_PTR = BTU_PTR);        /* PAGE 3-78      */
  .
  . WHEN(PU_T1)
  .   SEND 'BTU' TO PC_T1.RCV USING(PARM_PTR = BTU_PTR);        /* PAGE 3-80      */
  .
  . WHEN(PU_T2)
  .   SEND 'BTU' TO PC_T2.RCV USING(PARM_PTR = BTU_PTR);        /* PAGE 3-84      */
  .
  END;

  RETURN;
END PC.DEQ_Q_BTU_RCV;



UPM_BIU_SEGMENTER: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS OPTIONAL, IMPLEMENTATION-DEPENDENT  UPM MAY SEGMENT A  BIU INTO
   │                 MULTIPLE BIU  SEGMENTS.  IF SEGMENTING  IS PERFORMED,  THE RESULTING
   │                 BIU SEGMENTS ARE PLACED  IN THE PATH CONTROL PIU SEND  LIST WITH THE
   │                 FIRST ENTRY  BEING THE  FIRST SEGMENT,  THE SECOND  ENTRY BEING  THE
   │                 SECOND SEGMENT (IF REQUIRED), ... (IF  REQUIRED), AND THE LAST ENTRY
   │                 BEING THE LAST SEGMENT, AND WITH:
   │
   │                 • THE BBIUI AND EBIUI APPROPRIATELY SET IN EACH SEGMENT
   │
   │                 • THE EFI  IN EACH  SEGMENT SET  EQUAL TO  THE VALUE  OF EFI  IN THE
   │                   ORIGINAL BIU
   │
   │                 • THE SNF (IF APPLICABLE) IN EACH SEGMENT  SET EQUAL TO THE VALUE OF
   │                   THE SNF IN THE ORIGINAL BIU.
   │
   │      INPUT:     BIU POINTED TO BY ENTRY IN PCCB.PIU_SEND_LIST AND MU_PTR
   │
   │      OUTPUT:    MULTIPLE  BIU  SEGMENTS  IN   PCCB.PIU_SEND_LIST  IF  SEGMENTING  IS
   │                 PERFORMED
   │
   │      NOTE:      A FIRST BIU SEGMENT  IS REQUIRED TO BE AT LEAST  10 BYTES IN LENGTH;
   │                 HENCE, BIU'S LESS THAT 11 BYTES  IN LENGTH ARE NOT SEGMENTED.  ALSO,
   │                 BIU'S  CONTAINING  SESSION  ACTIVATION|DEACTIVATION  RU'S  ARE  NOT
   │                 SEGMENTED.
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 BF.PC.SEND                          PAGE 3-77
   │                 PC_T1.SEND                          PAGE 3-82
   │                 PC_T2.SEND                          PAGE 3-86
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  /* FUNCTION AS DESCRIBED ABOVE */

  RETURN;
END UPM_BIU_SEGMENTER;
```

```
ROUTE_EXTENSION_PIU_SEND: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE SENDS A FID2 OR FID3 PIU, AS FOLLOWS: │
  │                                                                      │
  │                 1. THE PIU IS CONVERTED FROM CANONICAL  FORM TO THE FORM REQUIRED TO
  │                    BE SENT OVER A LINK.                              │
  │                                                                      │
  │                 2. A PIU_VECTOR (DEFINED IN APPENDIX A) IS CREATED.  │
  │                                                                      │
  │                 3. THE PIU_VECTOR.PIU_PTR IS SET TO POINT TO THE PIU.│
  │                                                                      │
  │                 4. THE PIU_VECTOR.PIU_LENGTH IS SET EQUAL TO THE LENGTH OF THE PIU.
  │                                                                      │
  │                 5. A   POINTER  TO   THE   PIU_VECTOR  IS   ADDED   (FIFO)  TO   THE
  │                    LSCB.BTU_SEND_LIST FOR THE ADJACENT LINK STATION TO WHICH THE PIU
  │                    IS DESTINED.                                      │
  │                                                                      │
  │                 SEE NOTE 1.                                          │
  │                                                                      │
  │      INPUT:     MU_PTR POINTS  TO PIU;  LSCB_PTR POINTS  TO LSCB  FOR ADJACENT  LINK
  │                 STATION TO WHICH PIU IS DESTINED.                    │
  │                                                                      │
  │      OUTPUT:    A PIU_VECTOR--WHICH SPECIFIES THE LOCATION AND THE LENGTH OF THE PIU
  │                 TO BE  TRANSMITTED--IS INSERTED  (FIFO) INTO  LSCB.BTU_SEND_LIST FOR
  │                 DESTINATION ADJACENT LINK STATION.                   │
  │                                                                      │
  │      NOTES:     1. DLC.SEND  DEQUEUES THE  PIU_VECTOR  FROM THE  LSCB.BTU_SEND_LIST,
  │                    TRANSMITS THE PIU SPECIFIED BY THE PIU_VECTOR--AS THE BTU PORTION
  │                    OF A BLU--TO THE ADJACENT LINK STATION, AND DESTROYS BOTH THE PIU
  │                    AND THE PIU_VECTOR  WHEN THEY ARE NO LONGER  REQUIRED--THE PIU IS
  │                    SUCCESSFULLY TRANSMITTED OR THE TRANSMISSION IS ABANDONED.
  │                                                                      │
  │                 2. THIS FUNCTION CONVERTS  THE PIU FROM CANONICAL FORM  TO LINK FORM
  │                    AND SETS PIU_LENGTH EQUAL TO THE LENGTH OF THE PIU.
  │                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
  │                 BF.PC.SEND                          PAGE 3-77        │
  │                 PC_T1.SEND                          PAGE 3-82        │
  │                 PC_T2.SEND                          PAGE 3-86        │
  │                 ROUTE_EXTENSION_NEG_RSP             PAGE 3-99        │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL PIU_LENGTH FIXED(15);

  CALL MAP_FROM_CANONICAL(PIU_LENGTH);                    /* APPENDIX B, NOTE 2   */

  CREATE PIU_VECTOR;                      /* PIU_VECTOR IS DEFINED IN APPENDIX A   */

  PIU_VECTOR.PIU_PTR = MU_PTR;

  PIU_VECTOR.PIU_LENGTH = PIU_LENGTH;

  LOCK LSCB.BTU_SEND_LIST;
  .
  . INSERT PIU_VECTOR LAST IN LSCB.BTU_SEND_LIST;
  .
  UNLOCK;

  RETURN;

END ROUTE_EXTENSION_PIU_SEND;
```

This page intentionally left blank

```
T1_OR_T2_NO_BIU_ASSEMBLY_RCV_CK: PROCEDURE RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE  PERFORMS PIU RECEIVE CHECKS  REQUIRED AT A  PU_T1 OR │
  │                 PU_T2 NODE THAT DOES NOT SUPPORT BIU ASSEMBLY.               │
  │                                                                              │
  │      INPUT:     PIU, POINTED TO BY MU_PTR; LSCB_PTR POINTS TO LSCB FOR ADJACENT LINK │
  │                 STATION FROM WHICH PIU WAS RECEIVED.                         │
  │                                                                              │
  │      OUTPUT:    OK  RETURN CODE  IF PIU  IS VALID;  OTHERWISE, NG  RETURN CODE.  IF │
  │                 RETURN  CODE IS  NG, A  NEGATIVE RESPONSE  IS GENERATED,  OR PIU  IS │
  │                 DISCARDED.                                                   │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 PC_T1.RCV                          PAGE 3-80                 │
  │                 PC_T2.RCV                          PAGE 3-84                 │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 ROUTE_EXTENSION_NEG_RSP            PAGE 3-99                 │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);


  RC = NG;

  IF BBIUI =  ¬BBIU | EBIUI =  ¬EBIU THEN
    CALL ROUTE_EXTENSION_NEG_RSP(X'8007');  /* PAGE 3-99, SEGMENTING ERROR          */

  ELSE
    IF RU_CTGY = SC & DCF < 4 THEN
      CALL ROUTE_EXTENSION_NEG_RSP(X'1002');  /* PAGE 3-99, RU LENGTH ERROR         */

    ELSE
      IF MUCB.LDI = LOST_DATA THEN
        CALL ROUTE_EXTENSION_NEG_RSP(X'800A');  /* PAGE 3-99, TOO-LONG PIU         */

      ELSE
        RC = OK;

  RETURN(RC);

END T1_OR_T2_NO_BIU_ASSEMBLY_RCV_CK;
```

```
T1_OR_T2_STATION_BIU_ASSEMBLER: PROCEDURE(ASSEMBLER_RESULT);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE PERFORMS FUNCTIONS REQUIRED RELATIVE TO THE RECEPTION │
│                 OF PIU'S CONTAINING  A BIU OR BIU  SEGMENT AT A PU_T1  OR PU_T2 NODE │
│                 THAT  SUPPORTS  BIU  ASSEMBLY  ON A  STATION  BASIS,  INCLUDING  THE │
│                 ASSEMBLY  OF A  BIU  FROM PIU'S  CONTAINING  BIU  SEGMENTS, AND  THE │
│                 RECOGNITION AND PROCESSING OF ERRORS ASSOCIATED WITH BIU SEGMENTING. │
│                                                                          │
│                 FSM_STATION_BIU_ASSEMBLY IS USED  TO MAINTAIN THE STATE  OF THE NODE │
│                 RELATIVE TO BIU ASSEMBLY.                                │
│                                                                          │
│      INPUT:     PIU CONTAINING  BIU OR  FIRST, MIDDLE, OR  LAST BIU  SEGMENT; MU_PTR │
│                 POINTS TO  PIU; PCCB_PTR  POINTS TO  PCCB; PCCB.PARTIAL_BIU_PTR  MAY │
│                 POINT  TO  A  PARTIALLY  ASSEMBLED  BIU;  FSM_STATION_BIU_ASSEMBLY │
│                 INDICATES CURRENT STATE OF NODE RELATIVE TO BIU ASSEMBLY         │
│                                                                          │
│      OUTPUT:    THE  RETURN  PARAMETER,  ASSEMBLER_RESULT,  IS  SET  TO  EITHER │
│                 BIU_AVAILABLE  OR ¬BIU_AVAILABLE.  IF  ASSEMBLER_RESULT  IS SET  TO │
│                 BIU_AVAILABLE, MU_PTR  POINTS BIU.  IF ASSEMBLER_RESULT  IS SET  TO │
│                 ¬BIU_AVAILABLE,  PCCB.PARTIAL_BIU_PTR  MAY  POINT  TO  PARTIALLY │
│                 ASSEMBLED BIU.  A NEGATIVE  RESPONSE MAY  BE GENERATED  RELATIVE TO │
│                 INPUT PIU  OR PARTIALLY  ASSEMBLED BIU,  OR INPUT  PIU OR  PARTIALLY │
│                 ASSEMBLED BIU MAY BE DISCARDED.                          │
│                                                                          │
│      NOTES:     1. THIS CONCATENATES THE  BIU SEGMENT IN THE CURRENT  PIU POINTED TO │
│                    BY MU_PTR TO THE END OF THE PARTIALLY ASSEMBLED BIU POINTED TO BY │
│                    PCCB.PARTIAL_BIU_PTR.                                 │
│                                                                          │
│                 2. THIS ADDS THE DCF IN THE CURRENT  PIU POINTED TO BY MU_PTR TO THE │
│                    DCF  IN  THE  PARTIALLY  ASSEMBLED  BIU  POINTED  TO  BY │
│                    PCCB.PARTIAL_BIU_PTR.                                 │
│                                                                          │
│                 3. THIS SETS  THE EBIUI IN  THE BIU  BEING ASSEMBLED (POINTED  TO BY │
│                    PCCB.PARTIAL_BIU_PTR) TO  THE VALUE OF  THE EBIUI IN  THE CURRENT │
│                    PIU POINTED TO BY MU_PTR.  IF THE EBIUI IN THE CURRENT PIU IS SET │
│                    TO  EBIU, THE  PARTIAL  BIU  POINTED TO  BY  PCCB.PARTIAL_BIU_PTR │
│                    BECOMES A (WHOLE) BIU.                                │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 PC_T1.RCV                          PAGE 3-80             │
│                 PC_T2.RCV                          PAGE 3-84             │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 FSM_STATION_BIU_ASSEMBLY           PAGE 3-100            │
│                 LOG_ERROR_AND_DISCARD_PIU          PAGE 3-101            │
│                 ROUTE_EXTENSION_NEG_RSP            PAGE 3-99             │
│                 T1_OR_T2_FIRST_SEGMENT_RCV_CHK     PAGE 3-96             │
│                 T1_OR_T2_UNSEGMENTED_RCV_CHK       PAGE 3-96             │
│                 UPM_BIU_ASSEMBLY_CHK               PAGE 3-101            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL ASSEMBLER_RESULT BIT(1);
DCL P PTR;
DCL TEMP_PIU_PTR PTR;

ASSEMBLER_RESULT =  ¬BIU_AVAILABLE;

SELECT ANYORDER;
.
. WHEN(FSM_STATION_BIU_ASSEMBLY = BETBIU)                  /* PAGE 3-100      */
.   SELECT ANYORDER(BBIUI);
.   .
.   . WHEN( ¬BBIU)
.   .   CALL LOG_ERROR_AND_DISCARD_PIU(X'8007');  /* PAGE 3-101, SEGMENTING ERROR */
.   .
.   . WHEN(BBIU)
.   .   DO;
.   .   . CALL FSM_STATION_BIU_ASSEMBLY;                   /* PAGE 3-100      */
.   .   .
.   .   . SELECT ANYORDER;
.   .   . .
.   .   . . WHEN(FSM_STATION_BIU_ASSEMBLY = BETBIU)        /* PAGE 3-100      */
.   .   . .   IF T1_OR_T2_UNSEGMENTED_RCV_CHK = OK THEN    /* PAGE 3-96       */
.   .   . .     ASSEMBLER_RESULT = BIU_AVAILABLE;
.   .   . .
.   .   . . WHEN(FSM_STATION_BIU_ASSEMBLY = INBIU)         /* PAGE 3-100      */
.   .   . .   IF T1_OR_T2_FIRST_SEGMENT_RCV_CHK = OK THEN  /* PAGE 3-96       */
.   .   . .     PCCB.PARTIAL_BIU_PTR = MU_PTR;
.   .   . .
.   .   . .   ELSE
.   .   . .     CALL FSM_STATION_BIU_ASSEMBLY('RESET');     /* PAGE 3-100      */
.   .   . END;
.   .   END;
.   END;
```

```
. WHEN(FSM_STATION_BIU_ASSEMBLY = INBIU)                    /* PAGE 3-100          */
.   SELECT ANYORDER(BBIUI);
.   .
.   . WHEN( ¬BBIU)
.   .   IF (NCB.PU_TYPE = PU_T1 &
.   .         LSID ¬= PCCB.PARTIAL_BIU_PTR->LSID) |
.   .         (NCB.PU_TYPE = PU_T2 &
.   .         (DAFPRIME ¬= PCCB.PARTIAL_BIU_PTR->DAFPRIME |
.   .         OAFPRIME ¬= PCCB.PARTIAL_BIU_PTR->OAFPRIME |
.   .         SNF ¬= PCCB.PARTIAL_BIU_PTR->SNF)) THEN          /* OPTIONAL CHECK      */
.   .     DO;
.   .     . CALL LOG_ERROR_AND_DISCARD_PIU(X'8007');  /* PAGE 3-101, SEGMENTING ERROR  */
.   .     .
.   .     . MU_PTR = PCCB.PARTIAL_BIU_PTR;
.   .     .
.   .     . CALL ROUTE_EXTENSION_NEG_RSP(X'8007');  /* PAGE 3-99, SEGMENTING ERROR   */
.   .     .
.   .     . CALL FSM_STATION_BIU_ASSEMBLY('RESET');          /* PAGE 3-100          */
.   .     END;
.   .
.   .   ELSE
.   .     IF MUCB.LDI = LOST_DATA THEN
.   .       DO;
.   .       . CALL LOG_ERROR_AND_DISCARD_PIU(X'800A');  /* PAGE 3-101, TOO-LONG PIU  */
.   .       .
.   .       . MU_PTR = PCCB.PARTIAL_BIU_PTR;
.   .       .
.   .       . CALL ROUTE_EXTENSION_NEG_RSP(X'800A');  /* PAGE 3-99, TOO-LONG PIU   */
.   .       .
.   .       . CALL FSM_STATION_BIU_ASSEMBLY('RESET');          /* PAGE 3-100          */
.   .       END;
.   .
.   .     ELSE
.   .       IF UPM_BIU_ASSEMBLY_CHK = NG THEN /* OPTIONAL CHECK, PAGE 3-101        */
.   .         DO;
.   .         . CALL LOG_ERROR_AND_DISCARD_PIU(X'8010');       /* PAGE 3-101,         */
.   .         .                              /* SEGMENTED RU LENGTH ERROR           */
.   .         . MU_PTR = PCCB.PARTIAL_BIU_PTR;
.   .         .
.   .         . CALL ROUTE_EXTENSION_NEG_RSP(X'8010');         /* PAGE 3-99,          */
.   .         .                              /* SEGMENTED RU LENGTH ERROR           */
.   .         . CALL FSM_STATION_BIU_ASSEMBLY('RESET');        /* PAGE 3-100          */
.   .         END;
.   .
.   .       ELSE
.   .         DO;
.   .         . P = PCCB.PARTIAL_BIU_PTR;
.   .         .
.   .         . P->RU(P->DCF:(P->DCF + DCF - 1)) = RU(0:(DCF - 1));  /* NOTE 1       */
.   .         .
.   .         . PCCB.PARTIAL_BIU_PTR->DCF = PCCB.PARTIAL_BIU_PTR->DCF + DCF; /* NOTE 2  */
.   .         .
.   .         . PCCB.PARTIAL_BIU_PTR->EBIUI = EBIUI;           /* NOTE 3              */
.   .         .
.   .         . DISCARD MU;
.   .         .
.   .         . MU_PTR = PCCB.PARTIAL_BIU_PTR;
.   .         .
.   .         . CALL FSM_STATION_BIU_ASSEMBLY;                 /* PAGE 3-100          */
.   .         .
.   .         . IF FSM_STATION_BIU_ASSEMBLY = BETBIU THEN      /* PAGE 3-100          */
.   .         .   ASSEMBLER_RESULT = BIU_AVAILABLE;
.   .         END;
.   .
.   . WHEN(BBIU)
.   .   DO;
.   .   . TEMP_PIU_PTR = MU_PTR;
.   .   .
.   .   . MU_PTR = PCCB.PARTIAL_BIU_PTR;
.   .   .
.   .   . CALL ROUTE_EXTENSION_NEG_RSP(X'8007');  /* PAGE 3-99, SEGMENTING ERROR   */
.   .   .
.   .   . MU_PTR = TEMP_PIU_PTR;
.   .   .
.   .   . CALL FSM_STATION_BIU_ASSEMBLY;                       /* PAGE 3-100          */
.   .   .
.   .   . SELECT ANYORDER;
.   .   . .
.   .   . . WHEN(FSM_STATION_BIU_ASSEMBLY = BETBIU)            /* PAGE 3-100          */
.   .   . .   IF T1_OR_T2_UNSEGMENTED_RCV_CHK = OK THEN       /* PAGE 3-96           */
.   .   . .     ASSEMBLER_RESULT = BIU_AVAILABLE;
.   .   . .
.   .   . . WHEN(FSM_STATION_BIU_ASSEMBLY = INBIU)             /* PAGE 3-100          */
.   .   . .   IF T1_OR_T2_FIRST_SEGMENT_RCV_CHK = OK THEN     /* PAGE 3-96           */
.   .   . .     PCCB.PARTIAL_BIU_PTR = MU_PTR;
.   .   . .
.   .   . .   ELSE
.   .   . .     CALL FSM_STATION_BIU_ASSEMBLY('RESET');       /* PAGE 3-100          */
.   .   . .
.   .   . END;
.   .   END;
.   END;
. END;

. RETURN;

END T1_OR_T2_STATION_BIU_ASSEMBLER;
```

```
T1_OR_T2_SESSION_BIU_ASSEMBLER: PROCEDURE(ASSEMBLER_RESULT);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE PERFORMS FUNCTIONS REQUIRED RELATIVE TO THE RECEPTION │
│                 OF PIU'S CONTAINING  A BIU OR BIU  SEGMENT AT A PU_T1  OR PU_T2 NODE │
│                 THAT  SUPPORTS  BIU  ASSEMBLY  ON A  SESSION  BASIS,  INCLUDING  THE │
│                 ASSEMBLY  OF A  BIU  FROM PIU'S  CONTAINING  BIU  SEGMENTS, AND  THE │
│                 RECOGNITION AND PROCESSING OF ERRORS ASSOCIATED WITH BIU SEGMENTING. │
│                                                                          │
│                 FSM_SESSION_BIU_ASSEMBLY  IS  USED  TO  MAINTAIN   THE  STATE  OF  A │
│                 HALF-SESSION RELATIVE TO BIU ASSEMBLY.                   │
│                                                                          │
│      INPUT:     PIU CONTAINING  BIU OR  FIRST, MIDDLE, OR  LAST BIU  SEGMENT; MU_PTR │
│                 POINTS  TO  PIU;  SCB_PTR  POINTS  TO  HALF-SESSION  CONTROL  BLOCK; │
│                 SCB.PARTIAL_BIU_PTR  MAY   POINT  TO  A  PARTIALLY   ASSEMBLED  BIU; │
│                 FSM_SESSION_BIU_ASSEMBLY  INDICATES  CURRENT STATE  OF  HALF-SESSION │
│                 RELATIVE TO BIU ASSEMBLY                                 │
│                                                                          │
│      OUTPUT:    THE   RETURN  PARAMETER,   ASSEMBLER_RESULT,   IS   SET  TO   EITHER │
│                 BIU_AVAILABLE  OR ¬BIU_AVAILABLE.   IF  ASSEMBLER_RESULT  IS SET  TO │
│                 BIU_AVAILABLE, MU_PTR POINTS TO BIU.   IF ASSEMBLER_RESULT IS SET TO │
│                 ¬BIU_AVAILABLE, SCB.PARTIAL_BIU_PTR MAY POINT TO PARTIALLY ASSEMBLED │
│                 BIU.  A NEGATIVE RESPONSE MAY BE  GENERATED RELATIVE TO INPUT PIU OR │
│                 PARTIALLY ASSEMBLED BIU, OR INPUT PIU OR PARTIALLY ASSEMBLED BIU MAY │
│                 BE DISCARDED.                                            │
│                                                                          │
│      NOTES:     1. THIS CONCATENATES THE  BIU SEGMENT IN THE CURRENT  PIU POINTED TO │
│                    BY MU_PTR TO THE END OF THE PARTIALLY ASSEMBLED BIU POINTED TO BY │
│                    SCB.PARTIAL_BIU_PTR.                                  │
│                                                                          │
│                 2. THIS ADDS THE DCF IN THE CURRENT  PIU POINTED TO BY MU_PTR TO THE │
│                    DCF   IN   THE   PARTIALLY   ASSEMBLED    BIU   POINTED   TO   BY │
│                    SCB.PARTIAL_BIU_PTR.                                  │
│                                                                          │
│                 3. THIS SETS  THE EBIUI IN  THE BIU  BEING ASSEMBLED (POINTED  TO BY │
│                    SCB.PARTIAL_PIU_PTR) TO THE VALUE OF THE EBIUI IN THE CURRENT PIU │
│                    POINTED TO BY MU_PTR.  IF THE EBIUI  IN THE CURRENT PIU IS SET TO │
│                    EBIU, THE PARTIAL BIU POINTED TO BY SCB.PARTIAL_BIU_PTR BECOMES A │
│                    (WHOLE) BIU.                                          │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 PC_T1.RCV                          PAGE 3-80             │
│                 PC_T2.RCV                          PAGE 3-84             │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 FSM_SESSION_BIU_ASSEMBLY           PAGE 3-102            │
│                 LOG_ERROR_AND_DISCARD_PIU          PAGE 3-101            │
│                 ROUTE_EXTENSION_NEG_RSP            PAGE 3-99             │
│                 T1_OR_T2_FIRST_SEGMENT_RCV_CHK     PAGE 3-96             │
│                 T1_OR_T2_UNSEGMENTED_RCV_CHK       PAGE 3-96             │
│                 UPM_BIU_ASSEMBLY_CHK               PAGE 3-101            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL ASSEMBLER_RESULT BIT(1);
  DCL P PTR;
  DCL TEMP_PIU_PTR PTR;

  ASSEMBLER_RESULT =  ¬BIU_AVAILABLE;

  SELECT ANYORDER;
  .
  . WHEN(FSM_SESSION_BIU_ASSEMBLY = BETBIU)                  /* PAGE 3-102   */
  .   SELECT ANYORDER(BBIUI);
  .   .
  .   . WHEN( ¬BBIU)
  .   .   CALL LOG_ERROR_AND_DISCARD_PIU(X'8007');  /* PAGE 3-101, SEGMENTING ERROR */
  .   .
  .   . WHEN(BBIU)
  .   .   DO;
  .   .   . CALL FSM_SESSION_BIU_ASSEMBLY;                  /* PAGE 3-102   */
  .   .   .
  .   .   . SELECT ANYORDER;
  .   .   . .
  .   .   . . WHEN(FSM_SESSION_BIU_ASSEMBLY = BETBIU)       /* PAGE 3-102   */
  .   .   . .   IF T1_OR_T2_UNSEGMENTED_RCV_CHK = OK THEN   /* PAGE 3-96    */
  .   .   . .     ASSEMBLER_RESULT = BIU_AVAILABLE;
  .   .   . .
  .   .   . . WHEN(FSM_SESSION_BIU_ASSEMBLY = INBIU)        /* PAGE 3-102   */
  .   .   . .   IF T1_OR_T2_FIRST_SEGMENT_RCV_CHK = OK THEN /* PAGE 3-96    */
  .   .   . .     SCB.PARTIAL_BIU_PTR = MU_PTR;
  .   .   . .
  .   .   . .   ELSE
  .   .   . .     CALL FSM_SESSION_BIU_ASSEMBLY('RESET');   /* PAGE 3-102   */
  .   .   . END;
  .   .   END;
  .   END;
```

```
. WHEN(FSM_SESSION_BIU_ASSEMBLY = INBIU)                    /* PAGE 3-102          */
.   SELECT ANYORDER(BBIUI);
.   .
.   . WHEN( ¬BBIU)
.   .   IF (NCB.PU_TYPE = PU_T2) &
.   .        (SNF ¬= SCB.PARTIAL_BIU_PTR->SNF) THEN          /* OPTIONAL CHECK      */
.   .     DO;
.   .     . CALL LOG_ERROR_AND_DISCARD_PIU(X'8007');  /* PAGE 3-101, SEGMENTING ERROR */
.   .     .
.   .     . MU_PTR = SCB.PARTIAL_BIU_PTR;
.   .     .
.   .     . CALL ROUTE_EXTENSION_NEG_RSP(X'8007');  /* PAGE 3-99, SEGMENTING ERROR  */
.   .     .
.   .     . CALL FSM_SESSION_BIU_ASSEMBLY('RESET');           /* PAGE 3-102          */
.   .     END;
.   .
.   .   ELSE
.   .     IF MUCB.LDI = LOST_DATA THEN
.   .       DO;
.   .       . CALL LOG_ERROR_AND_DISCARD_PIU(X'800A');  /* PAGE 3-101, TOO-LONG PIU */
.   .       .
.   .       . MU_PTR = SCB.PARTIAL_BIU_PTR;
.   .       .
.   .       . CALL ROUTE_EXTENSION_NEG_RSP(X'800A');  /* PAGE 3-99, TOO-LONG PIU  */
.   .       .
.   .       . CALL FSM_SESSION_BIU_ASSEMBLY('RESET');           /* PAGE 3-102          */
.   .       END;
.   .
.   .     ELSE
.   .       IF UPM_BIU_ASSEMBLY_CHK = NG THEN /* OPTIONAL CHECK, PAGE 3-101       */
.   .         DO;
.   .         . CALL LOG_ERROR_AND_DISCARD_PIU(X'8010');        /* PAGE 3-101,       */
.   .         .                               /* SEGMENTED RU LENGTH ERROR         */
.   .         . MU_PTR = SCB.PARTIAL_BIU_PTR;
.   .         .
.   .         . CALL ROUTE_EXTENSION_NEG_RSP(X'8010');          /* PAGE 3-99,        */
.   .         .                               /* SEGMENTED RU LENGTH ERROR         */
.   .         . CALL FSM_SESSION_BIU_ASSEMBLY('RESET');         /* PAGE 3-102        */
.   .         END;
.   .
.   .       ELSE
.   .         DO;
.   .         . P = SCB.PARTIAL_BIU_PTR;
.   .         .
.   .         . P->RU(P->DCF:(P->DCF + DCF - 1)) = RU(0:(DCF - 1));  /* NOTE 1       */
.   .         .
.   .         . SCB.PARTIAL_BIU_PTR->DCF = SCB.PARTIAL_BIU_PTR->DCF + DCF; /* NOTE 2   */
.   .         .
.   .         . SCB.PARTIAL_BIU_PTR->EBIUI = EBIUI;             /* NOTE 3            */
.   .         .
.   .         . DISCARD MU;
.   .         .
.   .         . MU_PTR = SCB.PARTIAL_BIU_PTR;
.   .         .
.   .         . CALL FSM_SESSION_BIU_ASSEMBLY;                  /* PAGE 3-102        */
.   .         .
.   .         . IF FSM_SESSION_BIU_ASSEMBLY = BETBIU THEN       /* PAGE 3-102        */
.   .         .   ASSEMBLER_RESULT = BIU_AVAILABLE;
.   .         END;
.   .
.   . WHEN(BBIU)
.   .   DO;
.   .   . TEMP_PIU_PTR = MU_PTR;
.   .   .
.   .   . MU_PTR = SCB.PARTIAL_BIU_PTR;
.   .   .
.   .   . CALL ROUTE_EXTENSION_NEG_RSP(X'8007');  /* PAGE 3-99, SEGMENTING ERROR  */
.   .   .
.   .   . MU_PTR = TEMP_PIU_PTR;
.   .   .
.   .   . CALL FSM_SESSION_BIU_ASSEMBLY;                      /* PAGE 3-102        */
.   .   .
.   .   . SELECT ANYORDER;
.   .   . .
.   .   . . WHEN(FSM_SESSION_BIU_ASSEMBLY = BETBIU)           /* PAGE 3-102        */
.   .   . .   IF T1_OR_T2_UNSEGMENTED_RCV_CHK = OK THEN       /* PAGE 3-96         */
.   .   . .     ASSEMBLER_RESULT = BIU_AVAILABLE;
.   .   . .
.   .   . . WHEN(FSM_SESSION_BIU_ASSEMBLY = INBIU)            /* PAGE 3-102        */
.   .   . .   IF T1_OR_T2_FIRST_SEGMENT_RCV_CHK = OK THEN     /* PAGE 3-96         */
.   .   . .     SCB.PARTIAL_BIU_PTR = MU_PTR;
.   .   . .
.   .   . .   ELSE
.   .   . .     CALL FSM_SESSION_BIU_ASSEMBLY('RESET');       /* PAGE 3-102        */
.   .   . END;
.   .   END;
.   END;
  END;

  RETURN;

END T1_OR_T2_SESSION_BIU_ASSEMBLER;
```

```
T1_OR_T2_UNSEGMENTED_RCV_CHK: PROCEDURE RETURNS(BIT(1));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE  PERFORMS  RECEIVE  CHECKS  APPLICABLE  WHEN  A  PIU  │
   │                 CONTAINING AN  UNSEGMENTED (WHOLE)  BIU IS  RECEIVED AT  A PU_T1  OR  │
   │                 PU_T2 NODE THAT SUPPORTS BIU ASSEMBLY.                   │
   │                                                                          │
   │      INPUT:     PIU, POINTED TO BY MU_PTR; LSCB_PTR POINTS TO LSCB FOR ADJACENT LINK  │
   │                 STATION FROM WHICH PIU WAS RECEIVED.                     │
   │                                                                          │
   │      OUTPUT:    OK  RETURN CODE  IF PIU  IS VALID;  OTHERWISE, NG  RETURN CODE.   IF  │
   │                 RETURN  CODE IS  NG, A  NEGATIVE RESPONSE  IS GENERATED,  OR PIU  IS  │
   │                 DISCARDED.                                               │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 T1_OR_T2_SESSION_BIU_ASSEMBLER     PAGE 3-94             │
   │                 T1_OR_T2_STATION_BIU_ASSEMBLER     PAGE 3-92             │
   │                                                                          │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
   │                 ROUTE_EXTENSION_NEG_RSP            PAGE 3-99             │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);


  RC = NG;

  IF PCCB.BIU_ASSEMBLY_OPTION = STATION_ASSEMBLY &
       RU_CTGY = SC & DCF < 4 THEN
    CALL ROUTE_EXTENSION_NEG_RSP(X'1002');  /* PAGE 3-99, RU LENGTH ERROR          */

  ELSE
    IF MUCB.LDI = LOST_DATA THEN
      CALL ROUTE_EXTENSION_NEG_RSP(X'800A');  /* PAGE 3-99, TOO-LONG PIU           */

    ELSE
      RC = OK;

  RETURN(RC);

END T1_OR_T2_UNSEGMENTED_RCV_CHK;



T1_OR_T2_FIRST_SEGMENT_RCV_CHK: PROCEDURE RETURNS(BIT(1));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE  PERFORMS  RECEIVE  CHECKS  APPLICABLE  WHEN  A  PIU  │
   │                 CONTAINING A FIRST BIU SEGMENT IS RECEIVED  AT A PU_T1 OR PU_T2 NODE  │
   │                 THAT SUPPORTS BIU ASSEMBLY.                              │
   │                                                                          │
   │      INPUT:     PIU, POINTED TO BY MU_PTR; LSCB_PTR POINTS TO LSCB FOR ADJACENT LINK  │
   │                 STATION FROM WHICH PIU WAS RECEIVED.                     │
   │                                                                          │
   │      OUTPUT:    OK  RETURN CODE  IF PIU  IS VALID;  OTHERWISE, NG  RETURN CODE.   IF  │
   │                 RETURN  CODE IS  NG, A  NEGATIVE RESPONSE  IS GENERATED,  OR PIU  IS  │
   │                 DISCARDED.                                               │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 T1_OR_T2_SESSION_BIU_ASSEMBLER     PAGE 3-94             │
   │                 T1_OR_T2_STATION_BIU_ASSEMBLER     PAGE 3-92             │
   │                                                                          │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
   │                 ROUTE_EXTENSION_NEG_RSP            PAGE 3-99             │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);


  RC = NG;

  IF DCF < 10 THEN                                    /* OPTIONAL CHECK        */
    CALL ROUTE_EXTENSION_NEG_RSP(X'8007');  /* PAGE 3-99, SEGMENTING ERROR       */

  ELSE
    IF MUCB.LDI = LOST_DATA THEN
      CALL ROUTE_EXTENSION_NEG_RSP(X'800A');  /* PAGE 3-99, TOO-LONG PIU          */

    ELSE
      RC = OK;

  RETURN(RC);

END T1_OR_T2_FIRST_SEGMENT_RCV_CHK;
```

```
UPM_ALS_OPERATIVE_CHECK: PROCEDURE RETURNS(BIT(1));
                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS UPM DETERMINES IF AN ADJACENT LINK STATION (ALS) IS OPERATIVE.
  │
  │      INPUT:     LSCB_PTR POINTS TO LSCB FOR ALS.
  │
  │      OUTPUT:    AN OPERATIVE RETURN CODE IF  ALS IS OPERATIVE; OTHERWISE, ¬OPERATIVE
  │                 RETURN CODE.
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 BF.PC.SEND                          PAGE 3-77
  │                 PC_T1.SEND                          PAGE 3-82
  │                 PC_T2.SEND                          PAGE 3-86
  │                 ROUTE_EXTENSION_NEG_RSP             PAGE 3-99
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);

  RC = OPERATIVE;                                       /* NORMAL RETURN CODE    */

  /* FUNCTION AS DESCRIBED ABOVE */

  RETURN(RC);

END UPM_ALS_OPERATIVE_CHECK;
```

```
ROUTE_EXTENSION_TH_RCV_CHK: PROCEDURE(PU_TYPE) RETURNS(BIT(1));

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS  PROCEDURE  PERFORMS  BASIC  VALIDITY  CHECKS  ON  THE  PIU  TH
 │                 CONTAINED IN A BTU RECEIVED AT OR FROM A PU_T1 OR PU_T2 NODE.
 │
 │                 THE BTU DATA  ANALYZED BY THIS PROCEDURE  CONSISTS OF A PIU  IN LINK
 │                 FORM (AS OPPOSED TO CANONICAL FORM).
 │
 │      INPUT:     BTU, POINTED TO BY BTU_PTR; CALL PARAMETER SPECIFIES PU_TYPE
 │
 │      OUTPUT:    OK RETURN CODE  IF PIU TH IS  VALID; OTHERWISE, AN ERROR  IS LOGGED,
 │                 AND RETURN CODE IS SET TO NG.
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 BF.PC.RCV                          PAGE 3-78
 │                 PC_T1.RCV                          PAGE 3-80
 │                 PC_T2.RCV                          PAGE 3-84
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL PU_TYPE BINARY(8);

  DCL RC BIT(1);

  DCL PIU_PTR PTR;

  DCL 1 FIRST_BYTE_OF_FID2_OR_FID3_PIU UNALIGNED BASED(PIU_PTR),

        2 PIU_FID BIT(4),

        2 PIU_BBIUI BIT(1),

        2 PIU_EBIUI BIT(1),

        2 RESERVED BIT(1),

        2 PIU_EFI BIT(1);


  RC = NG;

  IF BTUCB.BTU_LENGTH < 1 THEN
    CALL UPM_LOG(X'800B');              /* APPENDIX B, INCOMPLETE TH          */

  ELSE
    DO;
    . PIU_PTR = ADDR(BTU_DATA);
    .
    . SELECT ANYORDER(PU_TYPE);
    . .
    . . WHEN(PU_T1)
    . .
    . .   IF PIU_FID ¬= FID3 THEN
    . .     CALL UPM_LOG(X'8006');      /* APPENDIX B, INVALID FID            */
    . .
    . .   ELSE
    . .     IF BTUCB.BTU_LENGTH < 2 THEN
    . .       CALL UPM_LOG(X'800B');    /* APPENDIX B, INCOMPLETE TH          */
    . .
    . .     ELSE
    . .       IF PIU_BBIUI = BBIU & BTUCB.BTU_LENGTH < 5 THEN
    . .         CALL UPM_LOG(X'4005');  /* APPENDIX B, INCOMPLETE RH          */
    . .
    . .       ELSE
    . .         RC = OK;
    . .
    . . WHEN(PU_T2)
    . .
    . .   IF PIU_FID ¬= FID2 THEN
    . .     CALL UPM_LOG(X'8006');      /* APPENDIX B, INVALID FID            */
    . .
    . .   ELSE
    . .     IF BTUCB.BTU_LENGTH < 6 THEN
    . .       CALL UPM_LOG(X'800B');    /* APPENDIX B, INCOMPLETE TH          */
    . .
    . .     ELSE
    . .       IF PIU_BBIUI = BBIU & BTUCB.BTU_LENGTH < 9 THEN
    . .         CALL UPM_LOG(X'4005');  /* APPENDIX B, INCOMPLETE RH          */
    . .
    . .       ELSE
    . .         RC = OK;
    . END;
    END;

  RETURN(RC);

END ROUTE_EXTENSION_TH_RCV_CHK;
```

```
ROUTE_EXTENSION_NEG_RSP: PROCEDURE(SNC_CODE);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  IF POSSIBLE,  THIS PROCEDURE  CHANGES A  FID2|FID3 PIU  OR PARTIALLY │
│                 ASSEMBLED BIU TO A NEGATIVE RESPONSE  PIU, SETS THE SENSE CODE EQUAL │
│                 TO  THE  VALUE  PASSED  IN THE  CALL  PARAMETER,  AND  ENQUEUES  THE │
│                 RESULTANT  NEGATIVE  RESPONSE PIU  ON  THE  BTU  SEND LIST  FOR  THE │
│                 ADJACENT LINK STATION FROM WHICH THE MU WAS RECEIVED.                │
│                                                                      │
│                 IF  THE MU  IS ONE  TO WHICH  NO  RESPONSE CAN  BE SENT,  OR IF  THE │
│                 ADJACENT LINK STATION IS NO LONGER OPERATIVE, AN ERROR IS LOGGED AND │
│                 THE MU IS DISCARDED.                                 │
│                                                                      │
│      INPUT:     FID2|FID3  PIU OR  PARTIALLY ASSEMBLED  BIU, POINTED  TO BY  MU_PTR; │
│                 LSCB_PTR POINTS TO LSCB FOR ADJACENT  LINK STATION FROM WHICH MU WAS │
│                 RECEIVED                                             │
│                                                                      │
│      OUTPUT:    NEGATIVE RESPONSE  PIU ENQUEUED  ON LSCB.BTU_SEND_LIST  FOR ADJACENT │
│                 LINK STATION IF MU  IS ONE TO WHICH A NEGATIVE  RESPONSE CAN BE SENT │
│                 AND  ADJACENT LINK  STATION  IS OPERATIVE;  OTHERWISE,  AN ERROR  IS │
│                 LOGGED AND MU IS DISCARDED.                          │
│                                                                      │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
│                 BF.PC.RCV                          PAGE 3-78         │
│                 PC_T1.RCV                          PAGE 3-80         │
│                 PC_T2.RCV                          PAGE 3-84         │
│                 T1_OR_T2_FIRST_SEGMENT_RCV_CHK     PAGE 3-96         │
│                 T1_OR_T2_NO_BIU_ASSEMBLY_RCV_CK    PAGE 3-91         │
│                 T1_OR_T2_SESSION_BIU_ASSEMBLER     PAGE 3-94         │
│                 T1_OR_T2_STATION_BIU_ASSEMBLER     PAGE 3-92         │
│                 T1_OR_T2_UNSEGMENTED_RCV_CHK       PAGE 3-96         │
│                                                                      │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
│                 ROUTE_EXTENSION_PIU_SEND           PAGE 3-89         │
│                 UPM_ALS_OPERATIVE_CHECK            PAGE 3-97         │
└──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL SNC_CODE BIT(32);
  DCL OAFPRIME_SAVE BINARY(8);

  IF BBIUI =  ¬BBIU | DCF < 3 | RQN | RRI = RSP |  /* SEE APPENDIX B FOR RQN      */
       UPM_ALS_OPERATIVE_CHECK =  ¬OPERATIVE THEN          /* PAGE 3-97        */
    DO;
    . CALL UPM_LOG(SNC_CODE);                              /* APPENDIX B       */
    .
    . DISCARD MU;
    END;

  ELSE
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(SNC_CODE);                 /* APPENDIX B       */
    .
    . IF FID = FID2 THEN
    .   DO;
    .   . OAFPRIME_SAVE = OAFPRIME;
    .   .
    .   . OAFPRIME = DAFPRIME;
    .   .
    .   . DAFPRIME = OAFPRIME_SAVE;
    .   END;
    .
    . CALL ROUTE_EXTENSION_PIU_SEND;                       /* PAGE 3-89        */
    END;

  RETURN;

END ROUTE_EXTENSION_NEG_RSP;
```

```
FSM_STATION_BIU_ASSEMBLY: FSM_DEFINITION CONTEXT(PCCB);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS FINITE-STATE MACHINE  IS USED TO MAINTAIN THE STATE  OF A PU_T1 │
   │                 OR PU_T2 NODE  RELATIVE TO BIU ASSEMBLY.  IT IS  APPLICABLE WHEN BIU │
   │                 ASSEMBLY IS PERFORMED ON A STATION BASIS.                    │
   │                                                                              │
   │                 THE  BETBIU STATE  INDICATES  "BETWEEN BIU'S"--A  BIU  IS NOT  BEING │
   │                 ASSEMBLED FROM PIU'S CONTAINING BIU SEGMENTS.                │
   │                                                                              │
   │                 THE  INBIU  STATE  INDICATES  "IN  BIU  ASSEMBLY"--A  BIU  IS  BEING │
   │                 ASSEMBLED FROM PIU'S CONTAINING BIU SEGMENTS.                │
   │                                                                              │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
   │                 T1_OR_T2_STATION_BIU_ASSEMBLER     PAGE 3-92                 │
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

   ┌──────────────────┬─────────┬─────────┐
   │   STATE NAMES---->│ BETBIU  │ INBIU   │
   │ INPUT             │ 01      │ 02      │
   ├──────────────────┼─────────┼─────────┤
   │  BBIU, EBIU       │ -       │ 1       │
   │  BBIU,¬EBIU       │ 2       │ -       │
   │ ¬BBIU, EBIU       │ /       │ 1       │
   │ ¬BBIU,¬EBIU       │ /       │ -       │
   │ 'RESET'           │ -       │ 1       │
   └──────────────────┴─────────┴─────────┘

END FSM_STATION_BIU_ASSEMBLY;
```

```
LOG_ERROR_AND_DISCARD_PIU: PROCEDURE(ERROR_MESSAGE);
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE IS CALLED WHEN AN ERROR IS DETECTED  AND THE CURRENT
   │                 PIU IS TO BE DISCARDED.
   │
   │                 THIS PROCEDURE:
   │
   │                 1. CALLS AN IMPLEMENTATION-DEPENDENT UPM TO LOG THE ERROR IDENTIFIED
   │                    IN THE  CALL PARAMETER, AND, OPTIONALLY,  ALL OR PART OF  THE PIU
   │                    ASSOCIATED WITH THE ERROR
   │
   │                 2. DISCARDS THE PIU
   │
   │      INPUT:     DETECTED ERROR IS IDENTIFIED IN CALL PARAMETER, AND MU_PTR POINTS TO
   │                 PIU THAT IS ASSOCIATED WITH ERROR AND IS TO BE DISCARDED.
   │
   │      OUTPUT:    ERROR LOGGED AND PIU DISCARDED.
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 PC.ERC                              PAGE 3-50
   │                 PC.TGC.RCV                          PAGE 3-40
   │                 PC.VRC.RCV                          PAGE 3-63
   │                 SESSION_RELATED_RCV                 PAGE 3-66
   │                 T1_OR_T2_SESSION_BIU_ASSEMBLER      PAGE 3-94
   │                 T1_OR_T2_STATION_BIU_ASSEMBLER      PAGE 3-92
   │                 VRC_BIU_ASSEMBLER                   PAGE 3-70
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL ERROR_MESSAGE CHAR(31);

  CALL UPM_LOG(ERROR_MESSAGE);                          /* APPENDIX B    */

  DISCARD MU;

  RETURN;

END LOG_ERROR_AND_DISCARD_PIU;



UPM_BIU_ASSEMBLY_CHK: PROCEDURE RETURNS(BIT(1));
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  OPTIONAL,  IMPLEMENTATION-DEPENDENT  UPM   DETERMINES  IF  THE
   │                 ADDITION OF A BIU SEGMENT TO A BIU BEING ASSEMBLED WOULD RESULT IN A
   │                 BIU THAT  EXCEEDS THE  (IMPLEMENTATION-DEFINED) MAXIMUM  RECEIVE BIU
   │                 LENGTH ALLOWED AT THE NODE, OR WOULD RESULT IN BUFFER DEPLETION.  IF
   │                 SO, IT  RETURNS A NG  RETURN CODE; IF NOT,  IT RETURNS AN  OK RETURN
   │                 CODE.
   │
   │      INPUT:     MU_PTR   POINTS  TO   BIU  SEGMENT;   FOR   STATION  BIU   ASSEMBLY,
   │                 PCCB.PARTIAL_BIU_PTR   POINTS   TO   BIU    BEING   ASSEMBLED;   FOR
   │                 SESSION_BIU_ASSEMBLY,  SCB.PARTIAL_BIU_PTR   POINTS  TO   BIU  BEING
   │                 ASSEMBLED.
   │
   │      OUTPUT:    OK OR NG RETURN CODE
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 T1_OR_T2_SESSION_BIU_ASSEMBLER      PAGE 3-94
   │                 T1_OR_T2_STATION_BIU_ASSEMBLER      PAGE 3-92
   │                 VRC_BIU_ASSEMBLER                   PAGE 3-70
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RC BIT(1);

  RC = OK;                                              /* NORMAL RETURN CODE */

  /* FUNCTION AS DESCRIBED ABOVE */

  RETURN(RC);

END UPM_BIU_ASSEMBLY_CHK;
```

```
FSM_SESSION_BIU_ASSEMBLY: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  FINITE-STATE  MACHINE IS  USED  TO  MAINTAIN  THE STATE  OF  A  │
   │                 HALF-SESSION RELATIVE  TO BIU ASSEMBLY.   IT IS APPLICABLE  WHEN BIU  │
   │                 ASSEMBLY IS PERFORMED ON A SESSION BASIS.                             │
   │                                                                                       │
   │                 THE  BETBIU STATE  INDICATES  "BETWEEN BIU'S"--A  BIU  IS NOT  BEING  │
   │                 ASSEMBLED FROM PIU'S CONTAINING BIU SEGMENTS.                         │
   │                                                                                       │
   │                 THE  INBIU  STATE  INDICATES  "IN  BIU  ASSEMBLY"--A  BIU  IS  BEING  │
   │                 ASSEMBLED FROM PIU'S CONTAINING BIU SEGMENTS.                         │
   │                                                                                       │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                        │
   │                 T1_OR_T2_SESSION_BIU_ASSEMBLER     PAGE 3-94                          │
   │                 VRC_BIU_ASSEMBLER                  PAGE 3-70                          │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

   ┌──────────────────┬──────────┬──────────┐
   │   STATE NAMES---->│ BETBIU   │ INBIU    │
   │ INPUT             │ 01       │ 02       │
   ├──────────────────┼──────────┼──────────┤
   │  BBIU, EBIU       │ -        │ 1        │
   │  BBIU,¬EBIU       │ 2        │ -        │
   │ ¬BBIU, EBIU       │ /        │ 1        │
   │ ¬BBIU,¬EBIU       │ /        │ -        │
   │ 'RESET'           │ -        │ 1        │
   └──────────────────┴──────────┴──────────┘

END FSM_SESSION_BIU_ASSEMBLY;
```

```
/*
┌──────────────────────────────────────────────────────────────────────┐
│                  PATH CONTROL FSM INPUT DEFINITIONS                  │
└──────────────────────────────────────────────────────────────────────┘
*/

FSM_INPUT_DEFINITION:

BBIU                    BBIUI    = BBIU;
DEC_WS                  VR_CWI   = DEC_WS;
EBIU                    EBIUI    = EBIU;
'FIRST_VRPRS'           FSMINPUT = 'FIRST_VRPRS';
'MODERATE_CONGESTION'   FSMINPUT = 'MODERATE_CONGESTION';
'RESET'                 FSMINPUT = 'RESET';
'SEVERE_CONGESTION'     FSMINPUT = 'SEVERE_CONGESTION';
'SNF_0_POST_SWEEP'      FSMINPUT = 'SNF_0_POST_SWEEP';
'SNF_0_PRE_SWEEP'       FSMINPUT = 'SNF_0_PRE_SWEEP';
'SUSPEND'               FSMINPUT = 'SUSPEND';
'SWEEP_BIT_PRE_SWEEP'   FSMINPUT = 'SWEEP_BIT_PRE_SWEEP';
'SWEEP_COMPLETE'        FSMINPUT = 'SWEEP_COMPLETE';
VR_PAC_RSP              VRPRS    = VR_PAC_RSP;
VR_PAC_RQ               VRPRQ    = VR_PAC_RQ;

END FSM_INPUT_DEFINITION;
```

This page intentionally left blank

INTRODUCTION

A distinct transmission control (TC) element (Figure 4-1) is provided for each half-session supported in a node, and is identified as HSID.TC. Whenever it is not ambiguous, the qualifying half-session prefix, HSID, will be omitted.

TC elements provide two protocol machines for each locally supported half-session:

- TC.CPMGR
- TC.SC

These protocol machines are interconnected as shown in Figure 4-2.

The protocol machine for session control, TC.SC, provides session-specific support for starting, clearing, and resynchronizing session-related data flows. The session control RUs providing activation or deactivation for a half-session are handled by PU.SVC_MGR.CSC_MGR (see Chapter 13).

The connection point manager (TC.CPMGR) controls sequence number checking, pacing, enciphering/deciphering, and other support functions relating to the half-session flows.

Each half-session with boundary function (BF) support has a BF.TC protocol machine in the node providing the BF support.

This chapter describes transmission control for locally supported half-sessions separately from transmission control in the boundary function; the details of BF.TC are presented at the end of the chapter.

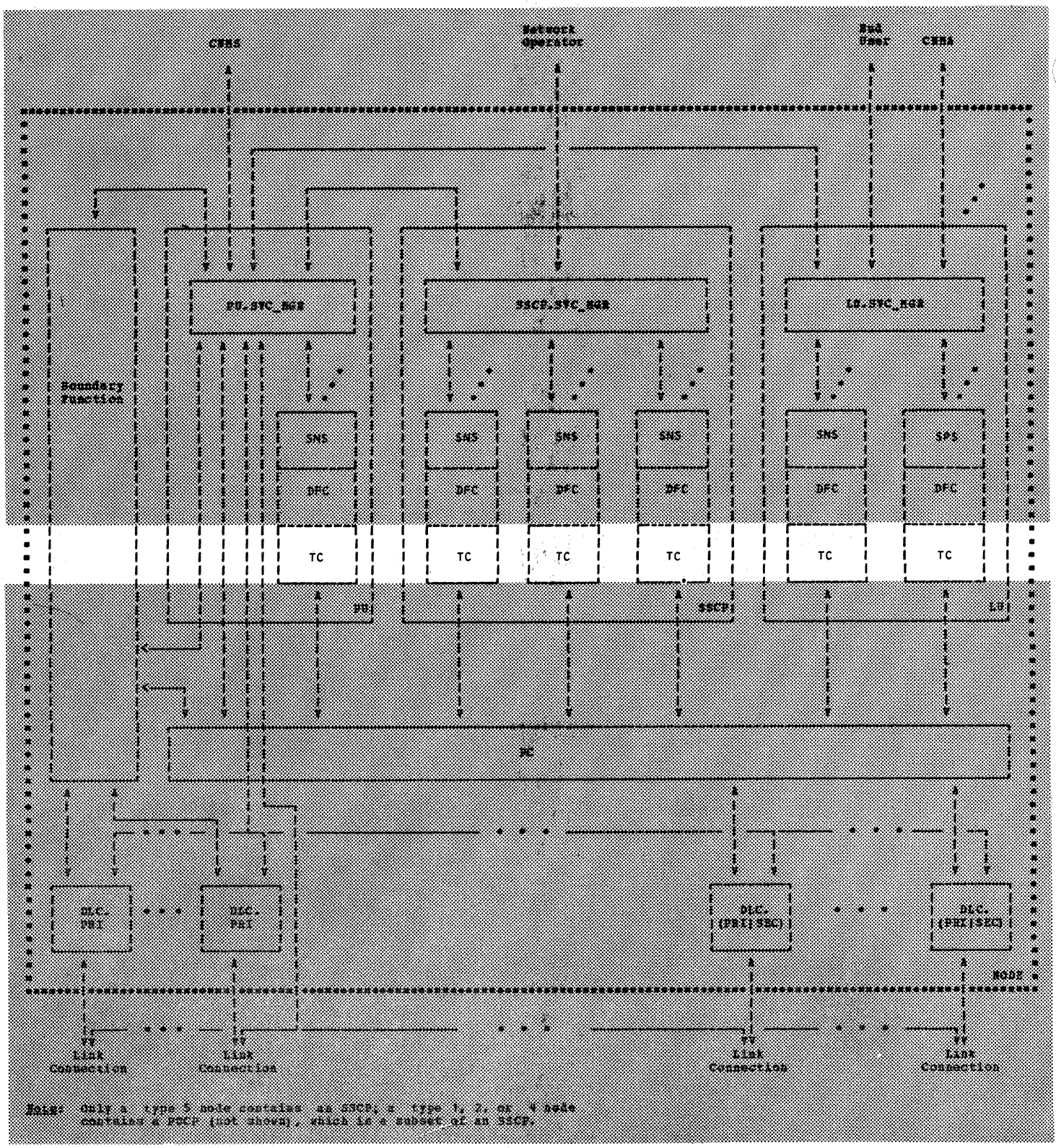


Figure 4-1. Structural Overview of a Node

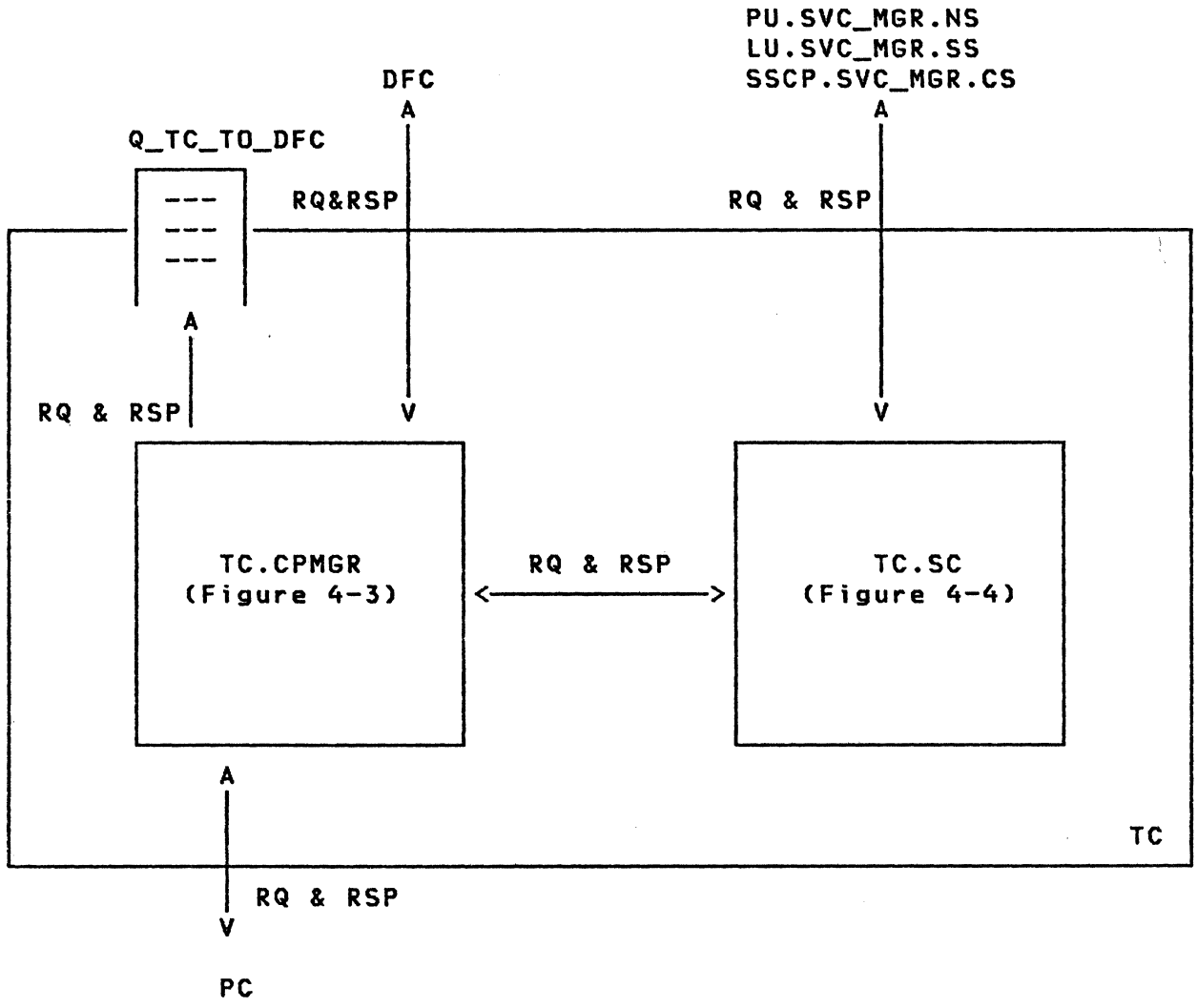


Figure 4-2. Structure of a TC element

INITIALIZATION PROCEDURES

Procedure SESSACT.TC_INITIALIZE (page 4-24) is called by PU.SVC_MGR.CSC_MGR (Chapter 13) when a half-session is being activated. This procedure and the two that it calls, SESSACT.PRIMARY_INITIALIZE (page 4-25) and SESSACT.SECONDARY_INITIALIZE (page 4-26), establish the component names of other layers in the node, and set up pacing parameters and the required finite-state machines for the profiles in use.

RESET HIERARCHY

Explicit reset signals are generated by certain TC FSMs and are directed to FSM subsets defined by the reset hierarchy; e.g., FSM_DT_SEND_SDT_AND_CLEAR (page 4-62) sends a reset signal to the FSMs in the CLEAR_RESET subtree (page 4-27) simultaneously with issuing CLEAR. Reset signals are also generated by the NAU services managers; e.g., the LU services manager sends a reset signal to the FSMs in the TC_RESET subtree (page 4-27) when +RSP(ACTLU) is sent.

There are three reset procedures defined in this chapter: TC_RESET (page 4-27), CLEAR_RESET (page 4-27), and CPMGR_RESET (page 4-28). TC_RESET is called by PU.SVC_MGR.CSC_MGR and resets all TC-related FSMs, queues, and variables. CLEAR_RESET is called when a CLEAR is processed and resets the appropriate TC-related FSMs, queues, and variables, all DFC FSMs, queues, and variables, and any FSMs, queues, and variables that are required for session presentation services (see SNA LU-LU Session Types). CPMGR_RESET resets all TC-related queues and variables and all TC-related FSMs except those for data traffic and cryptography.

SCHEDULER-INVOKED PROCEDURES

Procedures TC_OR_BF_TC.DEQUEUE.Q_PAC (page 4-29) and TC_OR_BF_TC.IPR_SEND (page 4-29) are invoked by the higher-level scheduler. (See Appendix C for details.) Both of these procedures appear in half-session TC elements and in boundary-function TC elements. TC_OR_BF_TC.DEQUEUE.Q_PAC is responsible for removing requests and responses from the pacing queue, Q_PAC, and sending them on to path control (see "Pacing," page 4-9). TC_OR_BF_TC.IPR_SEND is responsible for generating an isolated pacing response (IPR, see "Pacing") when both the architectural and resource requirements are satisfied.

CONNECTION POINT MANAGER

Each half-session contains a TC.CPMGR protocol machine having the structure shown in Figure 4-3. Detailed definitions for TC.CPMGR.SEND and TC.CPMGR.RCV, the major TC.CPMGR procedures, are shown on pages 4-31 and 4-36, respectively.

The protocols supported by a half-session TC.CPMGR include:

- Checking of sequence numbers on received normal-flow requests (Sequence numbers are assigned to normal-flow requests by DFC (Chapter 5)

- Proper separation of the normal flows from the expedited flows with respect to sequencing, pacing, and other TC protocols

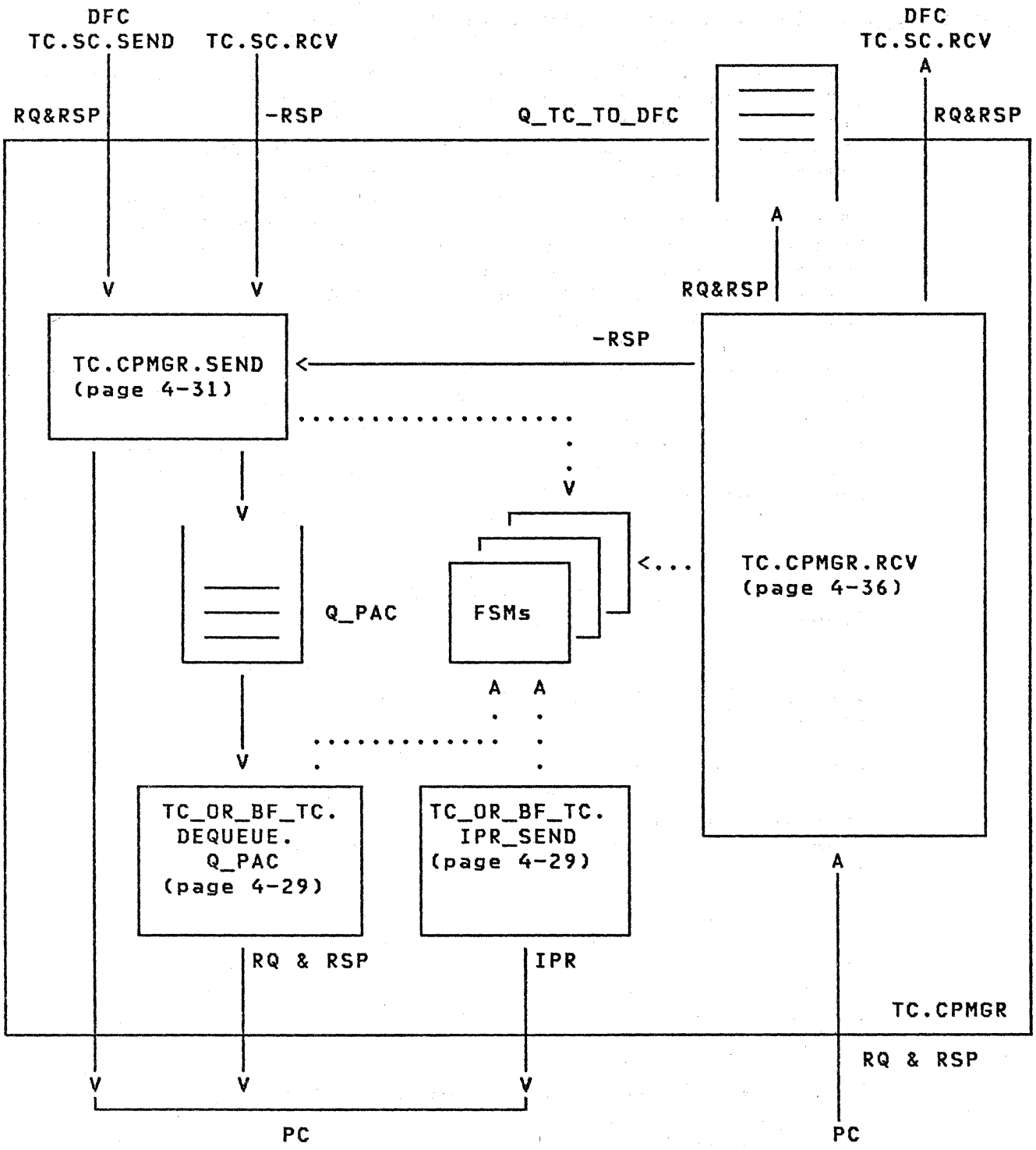
- Sending of normal-flow requests using pacing; this involves a queue (Q_PAC) for temporarily holding outgoing requests, and a set of coupled FSMs and procedures that manage the sending and receiving of pacing requests and responses (FSM_PAC_RQ_SEND (page 4-60) and FSM_PAC_RQ_RCV (page 4-61)).

- Sending of requests on the expedited flow using immediate request mode (see "Request and Response Control Modes," page 4-11) using FSM_CNTL_IMMED_EXP (page 4-61)

- Enqueuing, on Q_TC_TO_DFC, of requests destined for the DFC element

- Proper routing of requests and responses to PC (Chapter 3), DFC (Chapter 5), and TC.SC.RCV (page 4-44)

- Enciphering/deciphering control: For all LU-LU FM data RUs using session-level mandatory cryptography, and for those LU-LU FM data RUs with the Enciphered Data indicator (EDI) set to ED using session-level selective cryptography (see TC.CPMGR.SEND.NORM_RQ (page 4-33) and TC.CPMGR.RCV.NORM_RQ (page 4-40))



Note: TC_OR_BF_TC.DEQUEUE.Q_PAC and TC_OR_BF_TC.IPR_SEND are invoked by the higher-level scheduler.

Figure 4-3. Structure of TC.CPMGR

THE SEQUENCE NUMBERING OF REQUESTS AND RESPONSES

For some TS profiles (see Appendix F), 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; 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.) This orderly progression may be altered by a CLEAR or STSN request. Sequence numbers are assigned in the sending half-session by DFC and are checked in the receiving half-session by TC.CPMGR.

For the expedited flow, an identifier is assigned to each request sent. The identifier is not necessarily managed as a sequence number, but is unique for each outstanding expedited request sent within a layer. Expedited DFC RUs (QEC, RELQ, RSHUTD, SBI, SHUTC, SHUTD, SIG) are assigned identifiers by DFC; The SC requests CLEAR, CRV, RQR, SDT, and STSN (all of which are expedited) are assigned identifiers by TC.SC.

For other TS profiles, 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 DFC component is given to the sending end user or NAU services manager and is retained for use in correlating responses to requests (a response carries the sequence number or identifier of the corresponding request).

Because the FID3 TH format does not include a Sequence Number field, half-sessions located in a type 1 node do not use sequence numbers or identifiers. Sequence number and identifier assignment and checking for these half-sessions are performed in the boundary function by BF.TC.RCV (page 4-53).

Since the half-session responsible for recovery must be able to correlate responses to requests within a chain, restrictions are placed on the protocols used on sessions involving half-sessions located in type 1 nodes. Sessions involving these half-sessions use one of the following protocols, so that responses to requests flowing in the secondary-to-primary direction can be properly correlated by the half-session responsible for recovery:

- Immediate request mode and definite-response chains (and/or exception-response chains carrying CD) for the secondary-to-primary direction, or

- Primary half-session responsible for recovery

Sessions involving these half-sessions also use one of the following protocols, so that responses to requests flowing in the primary-to-secondary direction can be properly correlated by the primary half-session:

- Immediate request mode and definite-response chains (and/or exception-response chains carrying CD) for the primary-to-secondary direction, or

- Pacing with N=1 to the secondary TC.CPMGR. (In two-stage pacing (see "Boundary Function Considerations for Pacing," page 4-22), only the pacing from the boundary function to the secondary requires a window size of 1.) In addition, an IPR cannot precede any positive or negative response that may be returned; the receipt of an IPR thus indicates that processing of the previous request is complete and no response to that request will be returned.

These protocols always match the correct response and request. In the meta-implementation, however, DFC is not aware of the node type in which it resides; therefore in a type 1 node, TC.CPMGR inserts a dummy sequence number in requests and inserts the last sequence number sent in responses in order to allow DFC to function.

SESSIONS WITH CRYPTOGRAPHY

If session-level mandatory cryptography is selected when the session is activated, TC.CPMGR enciphers all FMD request RUs being sent and deciphers all FMD request RUs being received. If session-level selective cryptography is selected, only those FMD request RUs with the Enciphered Data indicator (EDI) set to ED are enciphered or deciphered. The end user sets this bit. The process of enciphering involves the following actions:

- The RU is padded, when necessary, to an integral multiple of 8 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 padding is required, the Padded Data indicator (PDI) is set to PD.

- Prior to enciphering, the first 8 bytes of an RU are exclusive-ORed with the value of the session cryptography seed; 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."

- 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.

Valid cryptography options are defined under the BIND format in Appendix E. Session-seed generation is described in Appendix E under RSP(BIND). Session-seed distribution is described in this chapter under "Cryptography Verification (CRV)" (page 4-18) and in Appendix E under RSP(BIND). The RH bits used for cryptography are defined in Chapter 2 and are displayed in Appendix D.

SESSION-LEVEL PACING

Session-level pacing allows a TC.CPMGR to control the rate at which it receives requests on the normal flow. (Virtual route pacing is described in Chapter 3.) If pacing is selected when the session is activated, all normal-flow requests are paced. 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.CPMGR is capable of sending requests faster than the receiving TC.CPMGR can process them. (Where a BF.TC element is interposed between primary and secondary TC.CPMGRs, pacing may occur in either one or two stages. See the section "Boundary Function Considerations for Pacing", page 4-22, for details.)

The pacing environment assumes that the receiving TC.CPMGR 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.CPMGR initially may send up to N requests. On the first request, it turns on the Pacing Request indicator. After the receiving TC.CPMGR receives the request that contains the Pacing Request indication, it can signal the sending TC.CPMGR (by using the Pacing Response indication) when it is ready to receive another group of requests.

The sending TC.CPMGR 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 (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). When a pacing response is received, the sending TC.CPMGR can send N more requests and therefore increases the pacing count by N. 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 ISOLATED PACING RESPONSE (IPR). 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.

The decision as to when a session-level pacing response can be sent is implementation-dependent and determined by an undefined protocol machine, UPM_RESOURCE. This procedure is invoked by TC_OR_BF_TC.IPR_SEND (page 4-29), when it is invoked by the higher-level scheduler, or by TC.CPMGR.SEND_NORM_RSP (page 4-33) or TC_OR_BF_TC.DEQUEUE.Q_PAC (page 4-29) when either is processing a response.

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 normal-flow responses indicate ¬QR, they can pass requests at queuing points in TC and BF.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 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 TC.CPMGRs' 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.

ISOLATED PACING RESPONSE (IPR)

An IPR is sent by TC.CPMGR.SEND to return a Pacing Response indication as discussed in the preceding section.

IPRs are the only way possible to send pacing responses to pacing requests when operating under no-response protocols (RQN).

The following fields of the TH and RH are set for an IPR:

TH: 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).

RH: IPRs are coded all-zeros 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.

REQUEST AND RESPONSE CONTROL MODES

In order to simplify implementation and to better manage error recovery situations, every half-session issues requests and responses according to defined control mode options.

The following request control modes are defined:

- Immediate request mode: All request chains are sent under a single constraint--no request may be sent on the flow by a given half-session when a previously sent definite-response request is still outstanding on that flow.

- Delayed request mode: There are no constraints on the sending of request chains.

Delayed request mode is less restrictive than immediate request mode; a sender that satisfies the restrictions of immediate request mode also satisfies the restrictions of delayed request mode.

The immediate request mode is used generally on the expedited flow in each direction in a session (exceptions are CLEAR and RQR). For expedited-flow requests on PU-PU flows, see Chapters 11 and 12. One of the control modes is used on the normal flow in each direction (primary-to-secondary and secondary-to-primary) for a given session activation.

The immediate request mode is enforced on the expedited flows by each TC.CPMGR.SEND using FSM_CNTL_IMMED_EXP (page 4-61). It is enforced in the TC layer instead of DFC, where other control modes are enforced, because TC SC RUs use the protocol.

When FSM_CNTL_IMMED_EXP is in the reset state, any number of expedited responses may be sent to path control; but once a request is passed, the BLOCK_RQ state is entered. Responses are still passed, but requests are rejected by send checks (that are dependent on the state of FSM_CNTL_IMMED_EXP) in TC.CPMGR.SEND, TC.SC, and DFC. When the response to the outstanding request is received, FSM_CNTL_IMMED_EXP returns to the reset state. Then the next request may be passed.

The request control modes used on the normal flows are enforced by DFC (see Chapter 5 for details).

The following response control modes are defined:

1. Immediate response mode: Responses are 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.

2. Delayed response mode: With the exception of the response to CHASE, responses may be sent in any order. All valid responses to requests received before CHASE must be sent before the response to CHASE is sent.

The particular request and response control modes to be used on the normal flows in any session are a function of the session-activation parameters. The modes to be used in one direction may be chosen independently of, and do not affect, the modes to be used in the other direction.

The response control modes used on the normal flows are enforced by DFC (see Chapter 5 for details).

TC.SC

Each session control element (TC.SC) (Figure 4-4) supports protocols related to data traffic activation, deactivation, and recovery. It also assigns the identifier for each request. The state-dependent checks made on received and sent requests and responses are defined in the various FSMs (pages 4-62 - 4-71). Boundary function considerations for session control requests and responses are described in the section "BF.TC."

COMMON TH VALUES

All SC requests and responses are sent expedited (the EFI bit is on in the TH).

COMMON RH VALUES

All SC requests are issued by TC.SC or by PU.SVC_MGR.CSC_MGR (see Chapter 13) with the following RH values:

RU Category	11
Format indicator	1
Sense Data Included indicator	0
Begin Chain indicator	1
End Chain indicator	1
Definite Response 1 indicator	1
Definite Response 2 indicator	0
Exception Response indicator	0
Queued Response indicator	0
Pacing indicator	0
Begin Bracket indicator	0
End Bracket indicator	0
Change Direction indicator	0
Code Selection indicator	0
Enciphered Data indicator	0
Padded Data indicator	0

All SC responses are issued by TC.SC or by PU.SVC_MGR.CSC_MGR (see Chapter 13) with the following RH values:

RU Category	11
Format indicator	1
Sense Data Included indicator	0 or 1
Begin Chain indicator	1
End Chain indicator	1
Definite Response 1 indicator	1
Definite Response 2 indicator	0
Response Type indicator	0 or 1
Queued Response indicator	0
Pacing indicator	0

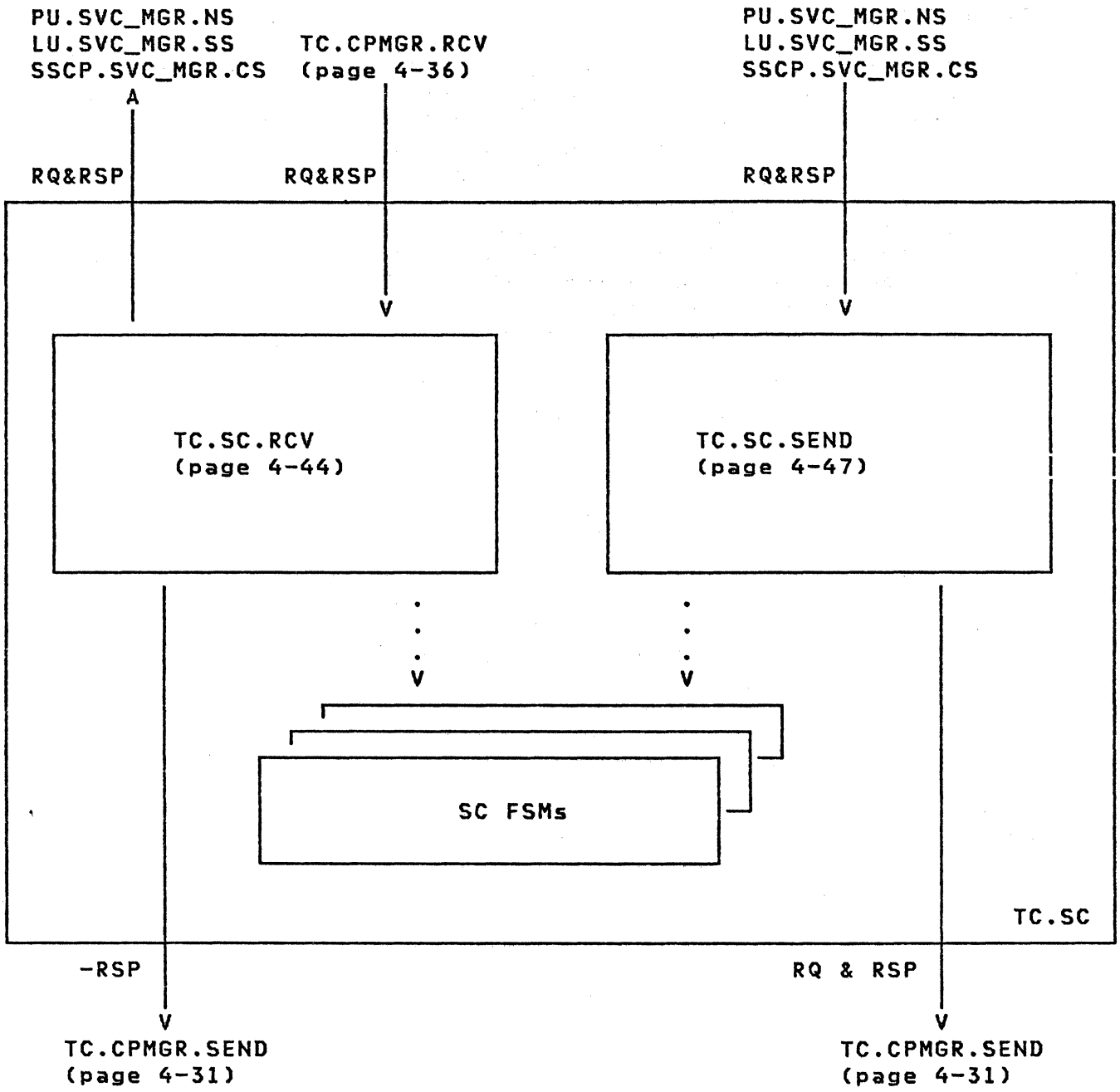


Figure 4-4. TC.SC

DATA TRAFFIC PROTOCOLS

The flow of FMD and DFC requests and responses in each active half-session is controlled by the state of a data traffic FSM; no FMD or DFC requests or responses are sent or validly received by a TC.CPMGR if its DT FSM is not in the active state. Data traffic flow is also affected by the state of the CRV FSM if session-level cryptography was specified in BIND. The data traffic protocols are useful, in that they allow session activation to be accomplished without permitting user-oriented data to flow before both half-sessions (and end users) are ready to receive such data or have completed required STSN processing.

There are four types of data traffic protocols. The type used in any active session is determined by the TS profile associated with session activation. (See Appendix F for other details of TS profiles.) The type distinguishes whether START DATA TRAFFIC (SDT) and/or CLEAR are valid for the session, as defined in the following table:

SDT	CLEAR	TS Profile	Pages
yes	yes	3 and 4	4-62 and 4-63
yes	no	5 and 17	4-64 and 4-65
no	yes	2	4-66 and 4-67
no	no	1 and 7	

START DATA TRAFFIC (SDT)
CLEAR (CLEAR)

Flow: From primary LU to secondary LU or from SSCP to PU|SSCP (Expedited) for SDT;
from primary LU to secondary LU (Expedited) for CLEAR

Principal FSMs:

FSM_DT_SEND_SDT_AND_CLEAR	(page 4-62)
FSM_DT_RCV_SDT_AND_CLEAR	(page 4-63)
FSM_DT_SEND_SDT	(page 4-64)
FSM_DT_RCV_SDT	(page 4-65)
FSM_DT_SEND_CLEAR	(page 4-66)
FSM_DT_RCV_CLEAR	(page 4-67)

SDT is sent by the primary session control to the secondary session control to enable both the sending and receiving of FMD and DFC requests and responses by both half-session TC.CPMGRs.

CLEAR is sent by the primary session control to reset the data traffic FSMs and the data traffic subtrees (e.g., brackets, pacing, sequence numbers) in the primary and secondary half-sessions. (For boundary function considerations, see "BF.TC," page 4-19.) CLEAR can be used after a catastrophic error as the first step in a data traffic recovery sequence.

Sending CLEAR precludes sending any further DFC or FMD requests or responses until a SDT is sucessfully processed. If SDT is not supported, the flow of FMD and DFC traffic is re-enabled when the RSP(CLEAR) is processed. All pending responses to DFC and FMD requests are discarded.

CLEAR is a valid request whenever the session is active. Any number of CLEARs may be outstanding at any one time. The CLEAR request and its response stay in order with other expedited requests and responses.

REQUEST RECOVERY (RQR)

Flow: From secondary LU to primary (Expedited)

Principal FSMs:

FSM_RQR_SEND	(page 4-67)
FSM_RQR_RCV	(page 4-68)

RQR is sent by the secondary to request the primary to initiate recovery for the session by sending CLEAR or to deactivate the session.

SET AND TEST SEQUENCE NUMBERS (STSN)

Flow: From primary LU to secondary LU (Expedited)

Principal FSMs: FSM_STSN_SEND (page 4-68)
FSM_STSN_RCV (page 4-69)

STSN is used by the sync point manager only after BIND has been sent and prior to the sending of SDT to resynchronize sync points following a session failure. The protocol associated with STSN requires that two versions of the normal-flow sequence numbers be kept. The first version is kept in both the primary and secondary half-sessions (see session control block in Appendix A); these are the half-session send and receive numbers. They correspond to the number of the last normal-flow request sent and the number of the last normal-flow request validly received by each half-session. The second version (the transaction processing program number) is kept by both the primary and secondary half-sessions' sync point managers. The sequence numbers kept by the sync point manager are not affected by any architecturally defined reset resulting from a session control request other than STSN.

STSN is sent by the primary half-session sync point manager to resynchronize the values of the half-session sequence numbers, for one or both of the normal flows at both ends of the session. Either or both sequence numbers (primary to secondary; or secondary to primary) can be "set," "sensed," or "set and tested." The sequence number values to be set are specified in the STSN request (see Appendix E for format details); they are set in each half-session associated with the session when the RU is processed by the half-session's associated TC.SC. If the action code in the request is "set," the secondary half-session's sync point manager is notified that its half-session sequence number has been changed. Testing or sensing is done only by the secondary half-session's sync point manager, not by TC.SC. Values to test or sense are asociated with a half-session by session name (see session name in the User Data field in BIND, Chapter 13). This allows correct restart even if network addresses change after a session failure and before restart. The restarted session retains the primary/secondary half-session polarity of the original session.

Half-session sequence number values are not affected by "sense" or "ignore" action codes.

CRYPTOGRAPHY VERIFICATION (CRV)

Flow: From primary LU to secondary LU (Expedited)

Principal FSMs: FSM_CRV_SEND (page 4-70)
FSM_CRV_RCV (page 4-71)

When session-level cryptography is specified in the BIND, CRV is sent by the primary LU session control to the secondary LU session control to enable sending and receiving of FMD requests by both half-sessions. CRV is a valid request only when session-level cryptography was selected in BIND. SDT can be sent only after +RSP(CRV) is received. CRV carries an 8-byte field (see Appendix E) that contains a transform (enciphered under the session cryptography key) of the deciphered value--the test value--received in +RSP(BIND); the transform in CRV is the test value with each bit of its first 4 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 value when enciphering/deciphering FMD RUs while the session is active.) The secondary TC.CMPGR obtains the returned test value by deciphering the aforementioned 8-byte field in CRV and inverting the first 4 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 cryptography seed.

BF.TC

Each secondary half-session within a peripheral node is given boundary function (BF) support within the adjacent subarea node. A general overview of BF is given in Chapter 1. The basic structure of BF is illustrated again in Figure 4-5. The details of BF.PC are given in Chapter 3. This section defines the TC aspects of BF.

A distinct BF.TC element is provided for each half-session receiving boundary function support, and is identified as SID.SEC.BF.TC. Whenever it is not ambiguous, the qualifying prefix, SID.SEC, will be omitted.

Each BF.TC consists of a send and a receive protocol machine (Figure 4-6). The receive protocol machine handles CLEAR processing and, for half-sessions in type 1 nodes, checks and assigns values carried in the Sequence Number field of the FID4 TH, since the FID3 TH has no such field. The send protocol machine provides boundary function support for pacing.

The boundary function has two TCCBs associated with it--one that is used for flows to and from the primary half-session and one that is used for flows to and from the secondary half-session.

The FSMs used to support each BF.TC protocol machine exist in a reset hierarchy described by BF.TC_RESET (page 4-52).

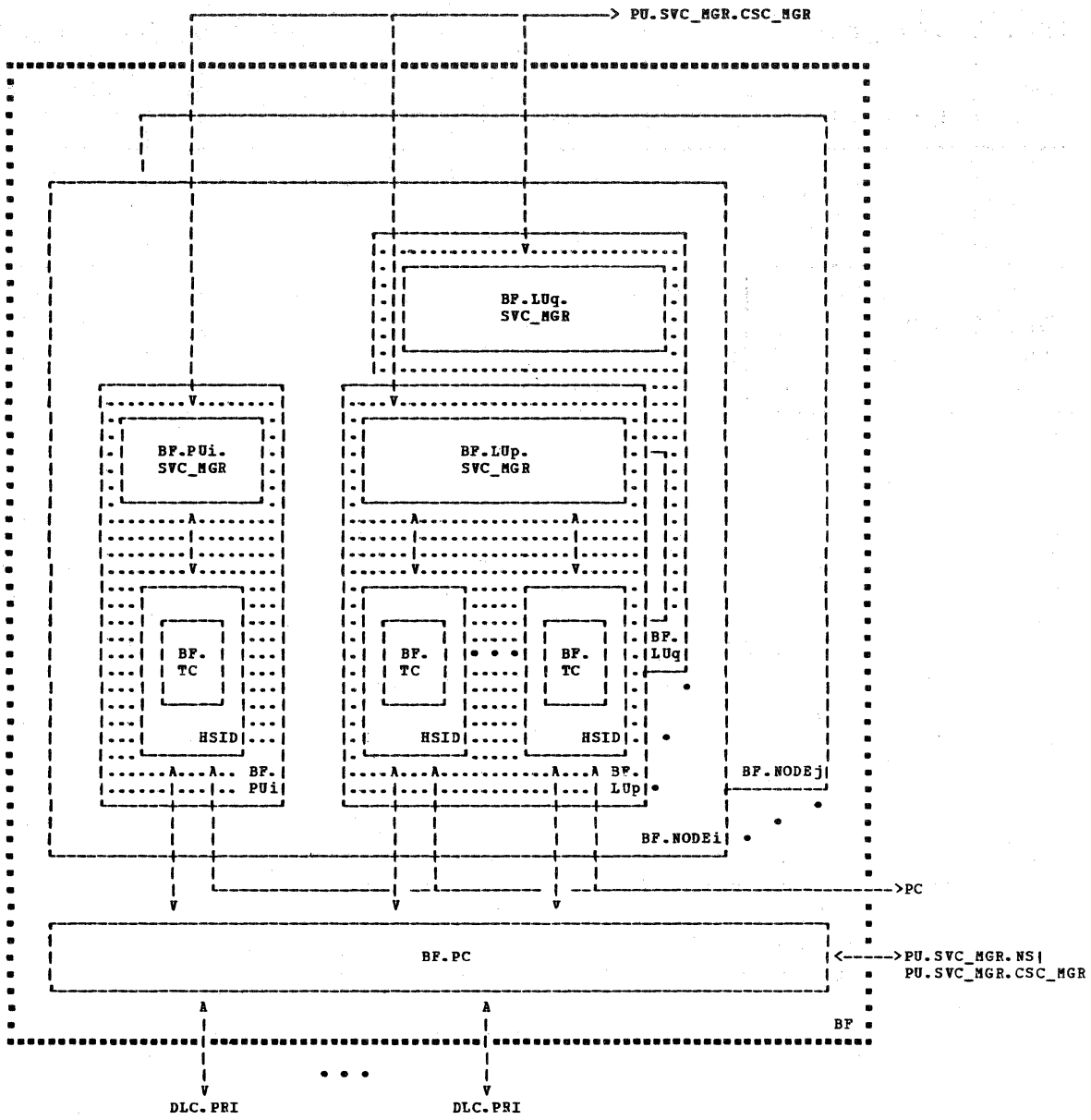
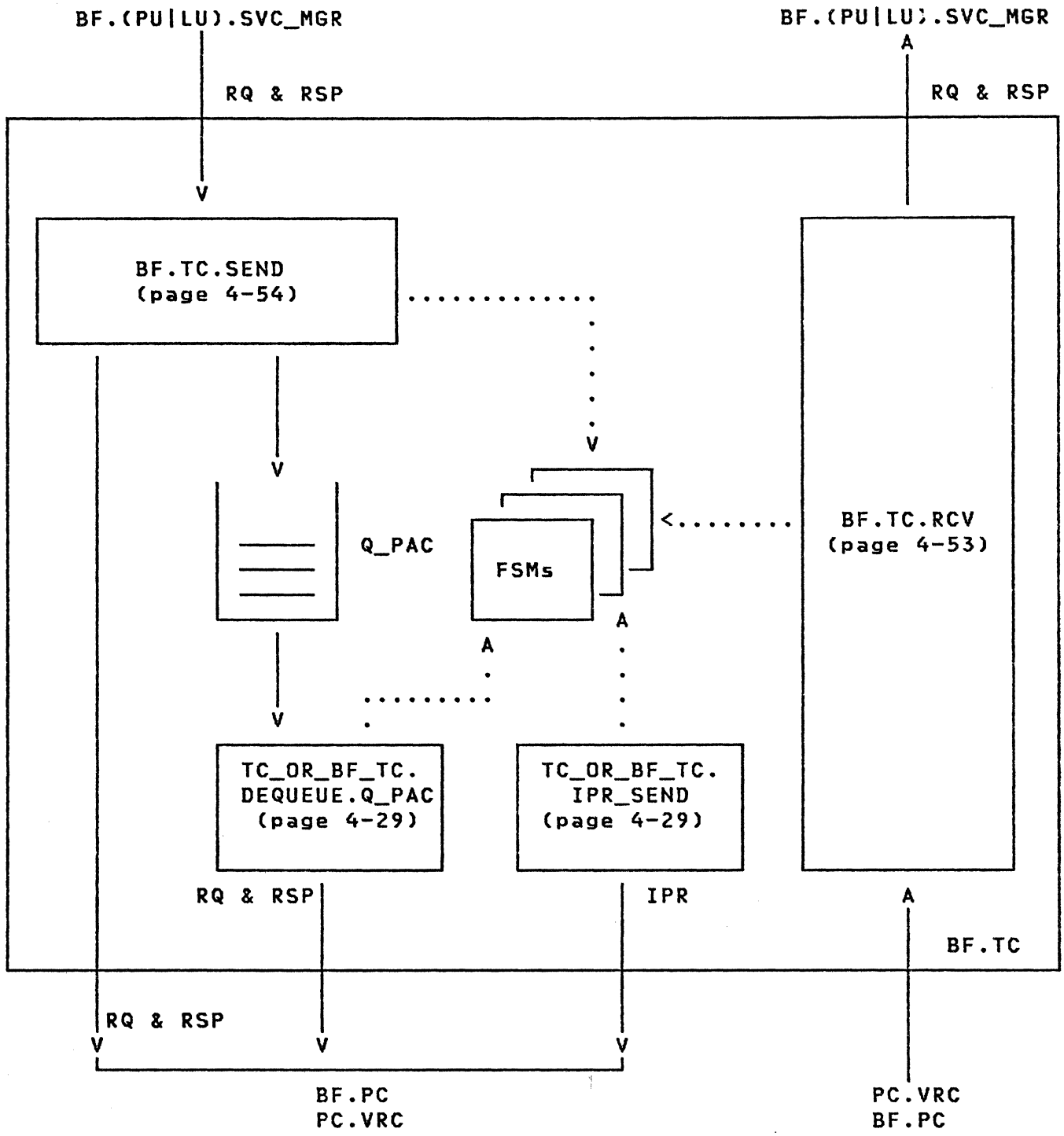


Figure 4-5. Boundary Function Structure



Note: TC_OR_BF_TC.DEQUEUE.Q_PAC and TC_OR_BF_TC.IPR_SEND are invoked by the higher-level scheduler.

Figure 4-6. Structure of BF.TC

BOUNDARY FUNCTION DATA TRAFFIC PROTOCOLS

CLEAR

The boundary function support for an LU-LU half-session processes both CLEAR and its response; all boundary function FSMs in the BF.TC_RESET hierarchy are reset when CLEAR or its response (whether positive or negative) is processed.

BOUNDARY FUNCTION CONSIDERATIONS FOR PACING

Pacing between a primary TC.CPMGR and a secondary TC.CPMGR, in a peripheral node supported by a boundary function, may occur in one stage (involving the primary and secondary TC.CPMGRs) or in two separate stages. One-stage pacing may be desirable if the primary LU and the boundary function are located in the same node; two-stage pacing may be more desirable otherwise.

If two stages are used, they are defined as follows:

- Stage 1--Primary (or secondary) TC.CPMGR to BF.TC: The purpose of this stage is to control the flow of requests from the primary (or secondary) TC.CPMGR to the BF.

- Stage 2--BF.TC to secondary (or primary) TC.CPMGR: The purpose of this stage is to control the flow of requests from the BF to the secondary (or primary) TC.CPMGR.

For flows that are paced, the window size (N) for each pacing stage is set at system definition or by a BIND parameter. The value of each N is independent of the others (see "Pacing" earlier in this chapter and the BIND RU specification in Appendix E).

If N is specified to be 0, then the associated stage is not paced. However, if N is specified to be 0 when the TS profile indicates that pacing may be used, and a request is received with the Pacing indicator on (PI=PAC), then the receiver must return either a pacing response or a negative response with sense code: Pacing Not Supported.

When the staging indicator for the primary TC.CPMGR to secondary TC.CPMGR flow is set indicating two-stage pacing, the primary TC.CPMGR send pacing count and the secondary TC.CPMGR receive pacing count do not have to be equal. If this staging indicator is set indicating one-stage pacing, the primary TC.CPMGR send pacing count is set equal to the secondary TC.CPMGR receive pacing count by the LU.SVC_MGR. The same is true for the secondary-to-primary direction.

The secondary LU may reduce the secondary TC.CPMGR receive pacing count suggested on a negotiable BIND; the primary TC.CPMGR send pacing count is set to the same value if pacing in this direction is to occur in one-stage. When two-stage pacing is indicated for a given direction and the request is a negotiable BIND, the boundary function may change the TC.CPMGR send pacing count for that direction. For a non-negotiable BIND, the boundary function can change the secondary TC.CPMGR send pacing count, if two-stage pacing is specified, but the primary TC.CPMGR send pacing count cannot be changed; if the primary TC.CPMGR send pacing count is unacceptable to BF, a negative response, Invalid Parameters (0821, 0832, 0833, or 0835), can be sent.

When one-stage pacing is used in one direction and two-stage pacing is used in the other direction, the boundary function passes the one-stage pacing request bit unaltered with the RH on which it was sent. However, the one-stage pacing response indicator cannot always be passed unaltered with the RH on which it was sent, because this RH can be delayed by normal-flow requests that are being held in Q_PAC awaiting a stage-2 pacing response. In order to avoid the delay, BF.TC.SEND (page 4-54) may generate an expedited-flow IPR and set PI=¬PAC in the original response.

BOUNDARY FUNCTION CONSIDERATIONS FOR SEGMENTING

Peripheral nodes may divide a normal-flow BIU into multiple BIU segments before sending it to the boundary function. The segments are passed on to their destination and assembled at the other end of the half-session. A subarea node sends only whole BIUs to the boundary function; BF.PC.SEND (Chapter 3) may segment the BIUs before sending them to the peripheral node.

```
SESSACT.TC_INITIALIZE: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  SETS UP SESSION PARAMETERS NEEDED BY TC.  THIS PROCEDURE IS EXECUTED │
 │                 WHEN THE SESSION IS BEING ACTIVATED.   A PROCEDURE IS CALLED TO FILL │
 │                 IN  THE SCB  DEPENDING ON  WHETHER THIS  IS A  PRIMARY OR  SECONDARY │
 │                 HALF-SESSION AND THE GENERIC VARIABLE PC IS ESTABLISHED DEPENDING ON │
 │                 THE NODE TYPE.                                       │
 │                                                                      │
 │      INPUT:     ON  CALL   FROM  PU.SVC_MGR.CSC_MGR,   THE  SCB_PTR   POINTS  TO   A │
 │                 HALF-SESSION SCB.                                    │
 │                                                                      │
 │      OUTPUT:    SCB IS UPDATED AND TCCB IS FILLED IN                 │
 │                                                                      │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
 │                 SESSACT.PRIMARY_INITIALIZE         PAGE 4-25         │
 │                 SESSACT.SECONDARY_INITIALIZE       PAGE 4-26         │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  TCCB_PTR = SCB.TC_CB_PTR;

  IF SCB.HALF_SESSION = PRIMARY THEN
    CALL SESSACT.PRIMARY_INITIALIZE;                    /* PAGE 4-25    */
  ELSE
    CALL SESSACT.SECONDARY_INITIALIZE;                  /* PAGE 4-26    */

  SELECT ANYORDER(NCB.PU_TYPE);
  . WHEN(PU_T1)
  .   #PC = PC_T1.SEND;                                 /* CHAPTER 3    */
  . WHEN(PU_T2)
  .   #PC = PC_T2.SEND;                                 /* CHAPTER 3    */
  . WHEN(PU_T4,PU_T5)
  .   #PC = PC.VRC.SEND;                                /* CHAPTER 3    */
  END;

  RETURN;

END SESSACT.TC_INITIALIZE;
```

```
SESSACT.PRIMARY_INITIALIZE: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  SETS UP SESSION PARAMETERS NEEDED BY A PRIMARY HALF-SESSION TC.  IT │
  │                 CALCULATES THE MAXIMUM RU SIZE THAT CAN BE SENT AND RECEIVED,  │
  │                 DETERMINES WHETHER OR NOT SESSION SEND AND RECEIVE PACING ARE USED,  │
  │                 AND SETS UP THE GENERIC FSM'S AND SVC_MGR VARIABLE.  THIS PROCEDURE  │
  │                 IS EXECUTED WHEN THE SESSION IS BEING ACTIVATED.             │
  │                                                                              │
  │      INPUT:     SCB_PTR POINTS AT A PRIMARY HALF-SESSION SCB AND TCCB_PTR IS  │
  │                 ESTABLISHED.                                                 │
  │                                                                              │
  │      OUTPUT:    UPDATED SCB AND TCCB                                         │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 SESSACT.TC_INITIALIZE              PAGE 4-24                 │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 DECODED                            PAGE 4-57                 │
  │                 FSM_CRV_SEND                       PAGE 4-70                 │
  │                 FSM_DT_SEND_CLEAR                  PAGE 4-66                 │
  │                 FSM_DT_SEND_SDT                    PAGE 4-64                 │
  │                 FSM_DT_SEND_SDT_AND_CLEAR          PAGE 4-62                 │
  │                 FSM_RQR_RCV                        PAGE 4-68                 │
  │                 FSM_STSN_SEND                      PAGE 4-68                 │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                                                /*
                    ┌──────────────────────────────────────┐
                    │        RU SIZES AND PACING COUNTS    │
                    └──────────────────────────────────────┘
                                                                                */
  IF SCB.PRI_SEND_MAX_RU_SIZE ¬= 0 THEN
    TCCB.MAX_SEND_RU_SIZE = DECODED(SCB.PRI_SEND_MAX_RU_SIZE);   /* PAGE 4-57   */
  ELSE
    TCCB.MAX_SEND_RU_SIZE = NOT_SPECIFIED;

  IF SCB.SEC_SEND_MAX_RU_SIZE ¬= 0 THEN
    TCCB.MAX_RCV_RU_SIZE = DECODED(SCB.SEC_SEND_MAX_RU_SIZE);    /* PAGE 4-57   */
  ELSE
    TCCB.MAX_RCV_RU_SIZE = NOT_SPECIFIED;

  IF SCB.PRI_SEND_PACING_CNT ¬= 0 THEN
    DO;
    . TCCB.SEND_PACING = YES;
    . TCCB.WINDOW_SIZE = SCB.PRI_SEND_PACING_CNT;
    . NEWLIST TCCB.Q_PAC ENTRY_NAME(MU) QUEUE;
    END;
  ELSE
    TCCB.SEND_PACING = NO;

  IF SCB.PRI_RCV_PACING_CNT ¬= 0 THEN
    TCCB.RCV_PACING = YES;
  ELSE
    TCCB.RCV_PACING = NO;
                                                                                /*
                    ┌──────────────────────────────────────┐
                    │              FSM'S                   │
                    └──────────────────────────────────────┘
                                                                                */
  IF SCB.SC_RQR = ALLOWED THEN
    #FSM_RQR = FSM_RQR_RCV;                                  /* PAGE 4-68   */
  ELSE
    #FSM_RQR = NO_OP;

  IF SCB.SC_STSN = ALLOWED THEN
    #FSM_STSN = FSM_STSN_SEND;                               /* PAGE 4-68   */
  ELSE
    #FSM_STSN = NO_OP;

  SELECT ANYORDER;
  . WHEN(SCB.SC_SDT = ALLOWED & SCB.SC_CLEAR = ALLOWED)
  .   #FSM_DT = FSM_DT_SEND_SDT_AND_CLEAR;                   /* PAGE 4-62   */
  . WHEN(SCB.SC_SDT = ALLOWED & SCB.SC_CLEAR ¬= ALLOWED)
  .   #FSM_DT = FSM_DT_SEND_SDT;                             /* PAGE 4-64   */
  . WHEN(SCB.SC_SDT ¬= ALLOWED & SCB.SC_CLEAR = ALLOWED)
  .   #FSM_DT = FSM_DT_SEND_CLEAR;                           /* PAGE 4-66   */
  . WHEN(SCB.SC_SDT ¬= ALLOWED & SCB.SC_CLEAR ¬= ALLOWED)
  .   #FSM_DT = NO_OP;
  END;

  IF SCB.SC_CRV = ALLOWED &
       SCB.CRYPTOGRAPHY_SESSION_LEVEL = (SELECTIVE | MANDATORY) THEN
    #FSM_CRV = FSM_CRV_SEND;                                 /* PAGE 4-70   */
  ELSE
    #FSM_CRV = NO_OP;

  RETURN;

END SESSACT.PRIMARY_INITIALIZE;
```

```
SESSACT.SECONDARY_INITIALIZE: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  SETS UP  SESSION PARAMETERS NEEDED  BY A SECONDARY  HALF-SESSION TC.  │
 │                 IT CALCULATES  THE MAXIMUM RU  SIZE THAT  CAN BE SENT  AND RECEIVED,  │
 │                 DETERMINES WHETHER OR NOT SEND AND RECEIVE PACING ARE USED, AND SETS  │
 │                 UP  THE  GENERIC FSM'S  AND  SVC_MGR  VARIABLE.  THIS  PROCEDURE  IS  │
 │                 EXECUTED WHEN THE SESSION IS BEING ACTIVATED.                         │
 │                                                                                       │
 │      INPUT:     SCB_PTR  POINTS AT  A  SECONDARY HALF-SESSION  SCB  AND TCCB_PTR  IS  │
 │                 ESTABLISHED.                                                          │
 │                                                                                       │
 │      OUTPUT:    UPDATED SCB AND TCCB                                                  │
 │                                                                                       │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                        │
 │                 SESSACT.TC_INITIALIZE              PAGE 4-24                          │
 │                                                                                       │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                                            │
 │                 DECODED                            PAGE 4-57                          │
 │                 FSM_CRV_RCV                        PAGE 4-71                          │
 │                 FSM_DT_RCV_CLEAR                   PAGE 4-67                          │
 │                 FSM_DT_RCV_SDT                     PAGE 4-65                          │
 │                 FSM_DT_RCV_SDT_AND_CLEAR           PAGE 4-63                          │
 │                 FSM_RQR_SEND                       PAGE 4-67                          │
 │                 FSM_STSN_RCV                       PAGE 4-69                          │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                                        /*
                 ┌──────────────────────────────────────────┐
                 │          RU SIZES AND PACING COUNTS      │
                 └──────────────────────────────────────────┘
                                                                        */
  IF SCB.SEC_SEND_MAX_RU_SIZE ¬= 0 THEN
    TCCB.MAX_SEND_RU_SIZE = DECODED(SCB.SEC_SEND_MAX_RU_SIZE);   /* PAGE 4-57   */
  ELSE
    TCCB.MAX_SEND_RU_SIZE = NOT_SPECIFIED;

  IF SCB.PRI_SEND_MAX_RU_SIZE ¬= 0 THEN
    TCCB.MAX_RCV_RU_SIZE = DECODED(SCB.PRI_SEND_MAX_RU_SIZE);    /* PAGE 4-57   */
  ELSE
    TCCB.MAX_RCV_RU_SIZE = NOT_SPECIFIED;

  IF SCB.SEC_SEND_PACING_CNT ¬= 0 THEN
    DO;
    . TCCB.SEND_PACING = YES;
    . TCCB.WINDOW_SIZE = SCB.SEC_SEND_PACING_CNT;
    . NEWLIST TCCB.Q_PAC ENTRY_NAME(MU) QUEUE;
    END;
  ELSE
    TCCB.SEND_PACING = NO;

  IF SCB.SEC_RCV_PACING_CNT ¬= 0 THEN
    TCCB.RCV_PACING = YES;
  ELSE
    TCCB.RCV_PACING = NO;
                                                                        /*
                 ┌──────────────────────────────────────────┐
                 │                FSM'S                     │
                 └──────────────────────────────────────────┘
                                                                        */
  IF SCB.SC_RQR = ALLOWED THEN
    #FSM_RQR = FSM_RQR_SEND;                                 /* PAGE 4-67   */
  ELSE
    #FSM_RQR = NO_OP;

  IF SCB.SC_STSN = ALLOWED THEN
    #FSM_STSN = FSM_STSN_RCV;                                /* PAGE 4-69   */
  ELSE
    #FSM_STSN = NO_OP;

  SELECT ANYORDER;
  . WHEN(SCB.SC_SDT = ALLOWED & SCB.SC_CLEAR = ALLOWED)
  .   #FSM_DT = FSM_DT_RCV_SDT_AND_CLEAR;                   /* PAGE 4-63   */
  . WHEN(SCB.SC_SDT = ALLOWED & SCB.SC_CLEAR ¬= ALLOWED)
  .   #FSM_DT = FSM_DT_RCV_SDT;                             /* PAGE 4-65   */
  . WHEN(SCB.SC_SDT ¬= ALLOWED & SCB.SC_CLEAR = ALLOWED)
  .   #FSM_DT = FSM_DT_RCV_CLEAR;                           /* PAGE 4-67   */
  . WHEN(SCB.SC_SDT ¬= ALLOWED & SCB.SC_CLEAR ¬= ALLOWED)
  .   #FSM_DT = NO_OP;
  END;

  IF SCB.SC_CRV = ALLOWED &
      SCB.CRYPTOGRAPHY_SESSION_LEVEL = (SELECTIVE | MANDATORY) THEN
    #FSM_CRV = FSM_CRV_RCV;                                  /* PAGE 4-71   */
  ELSE
    #FSM_CRV = NO_OP;

  RETURN;

END SESSACT.SECONDARY_INITIALIZE;
```

```
SESSACT.TC_RESET: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  RESETS ALL TC FSM'S IN THE DATA TRAFFIC SUBTREE, I.E., ALL TC FSM'S.
   │                THIS  ROUTINE IS  CALLED AS  A RESULT  OF RESETTING  A SUBTREE  THAT
   │                INCLUDES CRV, DT, AND ALL PACING OBJECTS.
   │
   │     INPUT:     RESET SIGNAL FROM A SERVICES MANAGER
   │
   │     OUTPUT:    FSM'S ARE RESET AND VARIABLES ARE SET TO THEIR INITIAL VALUES
   │
   │     REFERS TO THE FOLLOWING PROCEDURE(S):
   │                CPMGR_RESET                          PAGE 4-28
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  CALL #FSM_CRV('RESET');                                  /* PAGES 4-70 TO 4-71  */
  CALL #FSM_DT('RESET');                                   /* PAGES 4-62 TO 4-67  */

  CALL CPMGR_RESET;                                        /* PAGE 4-28           */

  RETURN;

END SESSACT.TC_RESET;



CLEAR_RESET: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  RESETS A HALF-SESSION WHEN A CLEAR IS BEING PROCESSED
   │
   │     INPUT:     CALLED BY AN FSM
   │
   │     OUTPUT:    FSM'S ARE RESET AND VARIABLES ARE SET TO THEIR INITIAL VALUES
   │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                FSM_DT_RCV_CLEAR                     PAGE 4-67
   │                FSM_DT_RCV_SDT_AND_CLEAR             PAGE 4-63
   │                FSM_DT_SEND_CLEAR                    PAGE 4-66
   │                FSM_DT_SEND_SDT                      PAGE 4-64
   │                FSM_DT_SEND_SDT_AND_CLEAR            PAGE 4-62
   │
   │     REFERS TO THE FOLLOWING PROCEDURE(S):
   │                CPMGR_RESET                          PAGE 4-28
   │                UPM_RESET_SPS                        PAGE 4-28
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  CALL CPMGR_RESET;                                        /* PAGE 4-28           */

  CALL SESSACT.DFC_RESET;                                  /* CHAPTER 5           */

  CALL UPM_RESET_SPS;                                      /* PAGE 4-28           */

  RETURN;

END CLEAR_RESET;
```

```
CPMGR_RESET: PROCEDURE;

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  RESETS ALL TC FSM'S  IN THE DATA TRAFFIC SUBTREE, EXCEPT  THE DT AND │
 │                 CRV FSM'S.  IT ALSO RESETS SESSION  PACING COUNT AND SEQUENCE NUMBER │
 │                 FIELDS.  THIS ROUTINE  IS CALLED AS A RESULT OF  RESETTING A SUBTREE │
 │                 THAT INCLUDES TC.                                            │
 │                                                                              │
 │      INPUT:     RESET SIGNAL FROM  A SERVICES MANAGER OR AN FSM  PROCESSING CLEAR OR │
 │                 SDT                                                          │
 │                                                                              │
 │      OUTPUT:    RESET FSM'S AND VARIABLES                                    │
 │                                                                              │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
 │                 CLEAR_RESET                        PAGE 4-27                 │
 │                 SESSACT.TC_RESET                   PAGE 4-27                 │
 │                                                                              │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
 │                 FSM_CNTL_IMMED_EXP                 PAGE 4-61                 │
 │                 FSM_PAC_RQ_RCV                     PAGE 4-61                 │
 │                 FSM_PAC_RQ_SEND                    PAGE 4-60                 │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │            ESTABLISH TCCB_PTR            │
                  └──────────────────────────────────────────┘
                                                                                */
  TCCB_PTR = SCB.TC_CB_PTR;
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │              RESET FSM'S                 │
                  └──────────────────────────────────────────┘
                                                                                */
  CALL #FSM_RQR('RESET');                                  /* PAGE 4-67        */
  CALL #FSM_STSN('RESET');                                 /* PAGE 4-68        */
  CALL FSM_PAC_RQ_SEND('RESET');                           /* PAGE 4-60        */
  CALL FSM_PAC_RQ_RCV('RESET');                            /* PAGE 4-61        */
  CALL FSM_CNTL_IMMED_EXP('RESET');                        /* PAGE 4-61        */
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │       EMPTY ALL TC RELATED QUEUES        │
                  └──────────────────────────────────────────┘
                                                                                */
  IF TCCB.SEND_PACING = YES THEN
    PURGE TCCB.Q_PAC;
  PURGE SCB.Q_TC_TO_DFC;
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │ RESET THE CURRENT SESSION  PACING RESIDUAL TO │
                  │ THE WINDOW SIZE                          │
                  └──────────────────────────────────────────┘
                                                                                */
  IF TCCB.SEND_PACING = YES THEN
    TCCB.PACING_COUNT = TCCB.WINDOW_SIZE;
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │  RESET NORMAL SEQUENCE NUMBER FIELDS TO ZERO │
                  └──────────────────────────────────────────┘
                                                                                */
  SCB.SQN_SEND_CNT = 0;
  SCB.SQN_RCV_CNT = 0;

  RETURN;

END CPMGR_RESET;




UPM_RESET_SPS: PROCEDURE;

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  RESETS  THAT  PART  OF  THE  HALF-SESSION  ASSOCIATED  WITH  SESSION │
 │                 PRESENTATION SERVICES (SNA LU-LU SESSION TYPES)               │
 │                                                                              │
 │      INPUT:     NONE                                                         │
 │                                                                              │
 │      OUTPUT:    FSM'S ARE RESET AND VARIABLES ARE SET TO THEIR INITIAL VALUES │
 │                                                                              │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
 │                 CLEAR_RESET                        PAGE 4-27                 │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  RETURN;

END UPM_RESET_SPS;
```

```
TC_OR_BF_TC.DEQUEUE.Q_PAC: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  DETERMINES IF IT IS  VALID TO REMOVE A MESSAGE UNIT  FROM Q_PAC.  IF │
  │                 VALID, REMOVES  PIU FROM Q_PAC AND  SENDS IT TO PATH  CONTROL.  THIS │
  │                 PROCEDURE MAY TURN PACING INDICATOR ON IN A RESPONSE         │
  │                                                                              │
  │      INPUT:     SIGNAL FROM HIGHER_LEVEL_SCHEDULER (APPENDIX C)              │
  │                                                                              │
  │      OUTPUT:    PIU TO PATH CONTROL (CHAPTER 3)                              │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 FSM_PAC_RQ_RCV                    PAGE 4-61                  │
  │                 FSM_PAC_RQ_SEND                   PAGE 4-60                  │
  │                 UPM_RESOURCES                     PAGE 4-59                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF TCCB.SEND_PACING = YES &                                  /* CAN ALWAYS SEND RSP &  */
       (TCCB.PACING_COUNT > 0 |                                /* CAN SEND RQ IF PACING  */
       FIRST_ENTRY(TCCB.Q_PAC)->RRI = RSP)  THEN               /* COUNT IS POSITIVE      */
    DO;
    . REMOVE FIRST(MU) FROM TCCB.Q_PAC;
    .
    . SELECT ANYORDER(RRI);
    . .
    . . WHEN(RQ)
    . .   DO;
    . .   . CALL FSM_PAC_RQ_SEND;                             /* PAGE 4-60              */
    . .   . TCCB.PACING_COUNT = TCCB.PACING_COUNT - 1;
    . .   END;
    . .
    . . WHEN(RSP)
    . .   DO;
    . .   . IF TCCB.RCV_PACING = YES &
    . .   .       UPM_RESOURCES = OK THEN                     /* PAGE 4-59              */
    . .   .    CALL FSM_PAC_RQ_RCV;                          /* PAGE 4-61              */
    . .   END;
    . END;
    .
    . IF SCB.SCB_TYPE = HALF_SESS THEN
    .   SEND MU TO #PC USING(ORIGIN = TC.CPMGR);             /* CHAPTER 3              */
    . ELSE
    .   SEND MU TO #PC USING(ORIGIN = BF.TC);                /* CHAPTER 3              */
    .
    END;

  RETURN;

END TC_OR_BF_TC.DEQUEUE.Q_PAC;



TC_OR_BF_TC.IPR_SEND: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  DETERMINES  IF   AN  IPR  MAY  BE   SENT  BASED  ON  THE   STATE  OF │
  │                 FSM_PAC_RQ_RCV (PAGE 4-61).  IF IT CAN BE SENT, GENERATES AN IPR AND │
  │                 SENDS IT TO PATH CONTROL.                                    │
  │                                                                              │
  │      INPUT:     SIGNAL FROM HIGHER_LEVEL_SCHEDULER (APPENDIX C)              │
  │                                                                              │
  │      OUTPUT:    ISOLATED PACING RESPONSE (IPR) TO PATH CONTROL (CHAPTER 3)    │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 CREATE_IPR                        PAGE 4-58                  │
  │                 FSM_PAC_RQ_RCV                    PAGE 4-61                  │
  │                 UPM_RESOURCES                     PAGE 4-59                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF TCCB.RCV_PACING = YES & FSM_PAC_RQ_RCV = PEND &           /* PAGE 4-61              */
       UPM_RESOURCES = OK THEN                                 /* PAGE 4-59              */
    DO;
    . CALL CREATE_IPR;                                       /* PAGE 4-58              */
    . EFI = EXPEDITED;                                       /* OPTIONAL               */
    . CALL FSM_PAC_RQ_RCV;                                   /* PAGE 4-61              */
    . IF SCB.SCB_TYPE = HALF_SESS THEN
    .   SEND MU TO #PC USING(ORIGIN = TC.CPMGR);             /* CHAPTER 3              */
    . ELSE
    .   SEND MU TO #PC USING(ORIGIN = BF.TC);                /* CHAPTER 3              */
    END;

  RETURN;

END TC_OR_BF_TC.IPR_SEND;
```

This page intentionally left blank

```
TC.CPMGR.SEND: PROCEDURE;
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  USAGE AND STATE CHECKS  ARE PERFORMED.  IN A TYPE 1  NODE, THE VALUE │
   │                 FROM THE SNF IS SAVED.  IF REQUIRED, THE MESSAGE UNIT IS ENCIPHERED. │
   │                 IF PACING IS SUPPORTED, THE MESSAGE UNIT MAY BE PLACED ON Q_PAC.     │
   │                                                                      │
   │      INPUT:     RQ|RSP FROM DFC.SEND|TC.SC|TC.CPMGR.RCV              │
   │                                                                      │
   │                 REQUESTS  CONTAIN  THE  FOLLOWING INFORMATION:  EFI,  SNF,  RRI=RQ,  │
   │                 RU_CTGY, FI,  SDI, BCI, ECI,  DR1I, DR2I,  ERI, QRI, BBI,  EBI, CDI,  │
   │                 CSI, EDI, RU                                         │
   │                                                                      │
   │                 RESPONSES  CONTAIN THE  FOLLOWING INFORMATION:  EFI, SNF,  RRI=RSP,  │
   │                 RU_CTGY, FI,  SDI (SAME SETTING AS  RTI), BCI, ECI, RI,  DR1I, DR2I,  │
   │                 QRI, RU                                              │
   │                                                                      │
   │      OUTPUT:    TH PARAMETERS AND BIU FOR RQ|RSP TO PC|Q_PAC         │
   │                                                                      │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
   │                 FSM_CNTL_IMMED_EXP                PAGE 4-61          │
   │                 TC.CPMGR.SEND_CHECKS              PAGE 4-32          │
   │                 TC.CPMGR.SEND_NORM_RQ             PAGE 4-33          │
   │                 TC.CPMGR.SEND_NORM_RSP            PAGE 4-33          │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL PACE BIT(1);
                                                                        /*
                   ┌──────────────────────────────────┐
                   │          ESTABLISH TCCB_PTR      │
                   └──────────────────────────────────┘
                                                                        */
  TCCB_PTR = SCB.TC_CB_PTR;
                                                                        /*
                   ┌──────────────────────────────────┐
                   │ OPTIONAL CHECKS.  NEED NOT BE DONE IF ALREADY  │
                   │ DONE IN A HIGHER LAYER OR COMPONENT            │
                   └──────────────────────────────────┘
                                                                        */
  IF  ¬DISPATCHED_BY(TC.SC*) &                          /* PAGE 4-47    */
      TC.CPMGR.SEND_CHECKS = NG THEN                    /* PAGE 4-32    */
    SEND SEND_CHECK TO SENDING_PROCEDURE;

  ELSE
    DO;
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      IN A  TYPE 1 NODE, THE  VALUE OF THE  SNF OF EACH  SENT REQUEST IS SAVED  TO BE  │
   │      INSERTED INTO THE RESPONSE WHEN IT ARRIVES SINCE THE FID3 DOES NOT HAVE AN SNF. │
   │      THIS IS A META-IMPLEMENTATION REQUIREMENT.                      │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
    . IF NCB.PU_TYPE = PU_T1 & RRI = RQ THEN
    .   IF EFI = NORMAL THEN
    .     SCB.SEND_NORM_SNF = SNF;
    .   ELSE
    .     SCB.SEND_EXP_SNF = SNF;
    .
    . MUCB.SEND_CHECK_SENSE = X'0000';                  /* RU IS GOOD   */
    .
    . SELECT ANYORDER;
    . .
    . . WHEN(RRI = RQ & EFI = EXPEDITED)
    . .   DO;
    . .   . CALL FSM_CNTL_IMMED_EXP;                   /* PAGE 4-61    */
    . .   . PACE = NO;                                  /* EXP RQ'S NOT PACED */
    . .   END;
    . .
    . . WHEN(RRI = RQ & EFI = NORMAL)
    . .   PACE = TC.CPMGR.SEND_NORM_RQ;                 /* PAGE 4-33    */
    . .
    . . WHEN(RRI = RSP & EFI = EXPEDITED)
    . .   PACE = NO;                                    /* EXP RSP'S NOT PACED */
    . .
    . . WHEN(RRI = RSP & EFI = NORMAL)
    . .   PACE = TC.CPMGR.SEND_NORM_RSP;                /* PAGE 4-33    */
    . END;
    .
    . SELECT INORDER;
    . .
    . . WHEN(MUCB.SEND_CHECK_SENSE ¬= X'0000')          /* ENCIPHERING FAILURE */
    . .   SEND SEND_CHECK TO SENDING_PROCEDURE;
    . .
    . . WHEN(PACE = YES)
    . .   INSERT MU LAST IN TCCB.Q_PAC;
    . .
    . . WHEN(PACE = NO)
    . .   SEND MU TO #PC;                               /* CHAPTER 3    */
    . .
    . END;
    .
    END;

  RETURN;
END TC.CPMGR.SEND;
```

```
TC.CPMGR.SEND_CHECKS: PROCEDURE RETURNS(BIT(1));

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE PERFORMS THE CONNECTION POINT MANAGER USAGE AND STATE
 │                 SEND ERROR CHECKS.
 │
 │      INPUT:     MU
 │
 │      OUTPUT:    IF  AN ERROR  IS FOUND,  A VALUE  OF NO  GOOD (NG)  IS RETURNED  AND
 │                 SEND_CHECK_SENSE IS SET; OTHERWISE, OK IS RETURNED.
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 TC.CPMGR.SEND                    PAGE 4-31
 │                 TC.SC.SEND                       PAGE 4-47
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 FSM_CNTL_IMMED_EXP               PAGE 4-61
 │                 UPM_Q_PAC_FULL                   PAGE 4-34
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  MUCB.SEND_CHECK_SENSE = X'0000';

  SELECT INORDER;
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │          SESSION NOT ACTIVE          │
                  └──────────────────────────────────────┘
                                                                        */
  . WHEN(#FSM_SESS ¬= ACTIVE)
  .   MUCB.SEND_CHECK_SENSE = X'8005';                  /* NO SESSION    */
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │            USAGE CHECKS              │
                  └──────────────────────────────────────┘
                                                                        */
  . WHEN(EFI = NORMAL &
  .      TCCB.MAX_SEND_RU_SIZE ¬= NOT_SPECIFIED &       /* LENGTH SPECIFIED   */
  .      DCF - RH_LENGTH > TCCB.MAX_SEND_RU_SIZE)       /* DCF-RH_LENGTH=RU LENGTH */
  .   MUCB.SEND_CHECK_SENSE = X'1002';                  /* INVALID RU SIZE    */
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │            STATE CHECKS              │
                  └──────────────────────────────────────┘
                                                                        */
  . WHEN(SEND_OR_RECEIVE_CHECK(FSM_CNTL_IMMED_EXP) |    /* PAGE 4-61          */
  .      SEND_OR_RECEIVE_CHECK(#FSM_DT) |               /* PAGES 4-62 TO 4-67 */
  .      SEND_OR_RECEIVE_CHECK(#FSM_CRV))               /* PAGES 4-70 TO 4-71 */
  .   ;                                                 /* SEND_CHECK_SENSE SET */
  .                                                     /* BY FSM'S           */
  .
  . WHEN(EFI = NORMAL &
  .       (RRI = RQ | (RRI = RSP & QRI = QR)) &
  .      UPM_Q_PAC_FULL = TRUE)                         /* PAGE 4-34          */
  .   MUCB.SEND_CHECK_SENSE = X'0812';                  /* PACING QUEUE IS FULL */
  .                                                     /* & MU IS TO BE PACED */
  .
  . OTHERWISE;
  .                                                     /* EVERYTHING OK      */
  .
  END;

  IF MUCB.SEND_CHECK_SENSE = X'0000' THEN
    RETURN(OK);
  ELSE
    RETURN(NG);

END TC.CPMGR.SEND_CHECKS;
```

```
TC.CPMGR.SEND_NORM_RQ: PROCEDURE RETURNS(BIT(1));

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  ENCIPHER A NORMAL-FLOW  REQUEST IF NECESSARY AND DETERMINE  IF IT IS
 │                 TO BE PACED
 │
 │      INPUT:     NORMAL RQ FROM CPMGR.SEND
 │
 │      OUTPUT:    RQ, ENCIPHERED IF NECESSARY
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 TC.CPMGR.SEND                      PAGE 4-31
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 RU_PAD                             PAGE 4-34
 │                 UPM_ENCIPHER                       PAGE 4-34
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                                        /*
                    ┌──────────────────────────────────┐
                    │      ENCIPHER IF NECESSARY       │
                    └──────────────────────────────────┘
                                                                        */
  IF RU_CTGY = FMD &
       DCF ¬= RH_LENGTH &
       SDI =  ¬SD &
       (SCB.CRYPTOGRAPHY_SESSION_LEVEL = MANDATORY |
       (SCB.CRYPTOGRAPHY_SESSION_LEVEL = SELECTIVE &          /* FOR SELECTIVE      */
       EDI = ED)) THEN                                        /* ENCIPHERING, EDI IS*/
                                                              /* SET BY THE END USER*/
                                                              /* TO INDICATE WHETHER*/
                                                              /* TO ENCIPHER        */
    DO;
    . CALL RU_PAD;                                            /* PAGE 4-34          */
    .                                                         /* ADDS PAD BYTES     */
    . IF UPM_ENCIPHER = NG THEN                               /* PAGE 4-34          */
    .   MUCB.SEND_CHECK_SENSE = X'0848';                      /* CRYPTOGRAPHY FUNCTION*/
    .                                                         /* INOPERATIVE        */
    END;
                                                                        /*
                    ┌──────────────────────────────────┐
                    │      DETERMINE IF PACED          │
                    └──────────────────────────────────┘
                                                                        */
  IF TCCB.SEND_PACING = YES THEN
    RETURN(YES);
  ELSE
    RETURN(NO);

END TC.CPMGR.SEND_NORM_RQ;



TC.CPMGR.SEND_NORM_RSP: PROCEDURE RETURNS(BIT(1));

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  PROCESS A  NORMAL-FLOW RESPONSE  BY DETERMINING  IF A  SESSION-LEVEL
 │                 PACING RESPONSE  SHOULD BE INCLUDED AND  IF THIS RESPONSE  SHOULD BE
 │                 PLACED ON THE PACING QUEUE
 │
 │      INPUT:     NORMAL RSP FROM CPMGR.SEND
 │
 │      OUTPUT:    RSP WITH PI POSSIBLY SET TO PAC
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 TC.CPMGR.SEND                      PAGE 4-31
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 FSM_PAC_RQ_RCV                     PAGE 4-61
 │                 UPM_RESOURCES                      PAGE 4-59
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF TCCB.SEND_PACING = YES THEN
    DO;
    . IF QRI =  ¬QR | EMPTY(TCCB.Q_PAC) THEN
    .   DO;
    .   . IF FSM_PAC_RQ_RCV = PEND &                          /* PAGE 4-61          */
    .   .       UPM_RESOURCES = OK THEN                       /* PAGE 4-59          */
    .   .   CALL FSM_PAC_RQ_RCV;                              /* PAGE 4-61          */
    .   . RETURN(NO);
    .   END;
    . ELSE
    .   RETURN(YES);
    END;
  ELSE
    RETURN(NO);

END TC.CPMGR.SEND_NORM_RSP;
```

```
RU_PAD: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  EXTEND THE RU TO A MULTIPLE OF 8  BYTES.  THE VALUE OF THE PAD BYTES
   │                 IS UNPREDICTABLE EXCEPT FOR THE LAST BYTE, WHICH CONTAINS THE NUMBER
   │                 OF PAD BYTES AS AN UNSIGNED NUMBER.
   │
   │      INPUT:     MU
   │
   │      OUTPUT:    MU WITH RU EXTENDED (IF NECESSARY) TO LENGTH THAT IS A MULTIPLE OF 8
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 TC.CPMGR.SEND_NORM_RQ              PAGE 4-33
   │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):
   │                 UPM_PAD                            PAGE 4-35
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL PAD FIXED(15) BIN;
  DCL PAD_ALIAS CHAR(2) BASED(ADDR(PAD));

  PAD = 8 - MODULO(DCF - RH_LENGTH,8);                          /* APPENDIX B    */
  IF PAD ¬= 8 THEN
    DO;
    . RU = RU(0:DCF - RH_LENGTH - 1)||UPM_PAD(PAD - 1)||PAD_ALIAS(1:1);  /* PAGE 4-35 */
    . DCF = DCF + PAD;
    . PDI = PD;
    END;
  ELSE
    PDI =  ¬PD;

  RETURN;

END RU_PAD;



UPM_Q_PAC_FULL: PROCEDURE RETURNS(BIT(1));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  DETERMINES IF A PACING QUEUE IS FULL
   │
   │      INPUT:     THE PACING QUEUE ASSOCIATED WITH THE SCB
   │
   │      OUTPUT:    TRUE IF IT IS FULL; OTHERWISE FALSE
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 TC.CPMGR.SEND_CHECKS               PAGE 4-32
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  RETURN(FALSE);

END UPM_Q_PAC_FULL;



UPM_ENCIPHER: PROCEDURE RETURNS(BIT(1));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  ENCIPHERS THE RU USING THE DES ALGORITHM
   │
   │      INPUT:     MU TO BE ENCIPHERED
   │
   │      OUTPUT:    OK OR NG.   IF OK, MU WITH  RU ENCIPHERED.  OTHERWISE, MU  AS IT WAS
   │                 PASSED
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 TC.CPMGR.SEND_NORM_RQ              PAGE 4-33
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  RETURN(OK);

END UPM_ENCIPHER;
```

```
UPM_PAD: PROCEDURE(LEN) RETURNS(CHARACTER(8) VARYING);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  GENERATES THE REQUIRED NUMBER OF UNPREDICTABLE CHARACTERS │
   │                                                                          │
   │      INPUT:     THE NUMBER OF BYTES REQUIRED, BETWEEN 1 AND 7 INCLUSIVE  │
   │                                                                          │
   │      OUTPUT:    A CHARACTER STRING OF THE REQUESTED LENGTH               │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 RU_PAD                                PAGE 4-34          │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL LEN FIXED(15) BIN;

  DCL PAD CHAR(8);

  PAD = '        '; /* AN IMPLEMENTATION SHOULD CHOOSE A PSEUDO-RANDOM VALUE   */
  RETURN(PAD(0:LEN - 1));

END UPM_PAD;
```

```
TC.CPMGR.RCV: PROCEDURE;
                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THE USAGE AND STATE CHECKS ARE MADE.  IF THE MESSAGE UNIT CONTAINS A │
│                 PACING RESPONSE,  IT IS  PROCESSED.  TYPE  1 NODES  HAVE SNF  ADDED. │
│                 REQUESTS AND RESPONSES ARE ROUTED AND PACING REQUESTS ARE PROCESSED. │
│                                                                          │
│      INPUT:     RQ|RSP FROM PC.  THE TH FIELDS AND BIU ARE THE SIGNIFICANT FIELDS. │
│                                                                          │
│      OUTPUT:    RQ|RSP TO DFC.RCV|TC.SC.RCV OR -RSP TO TC.CPMGR.SEND    │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 ADD_SNF_FOR_T1                  PAGE 4-40                │
│                 FSM_CNTL_IMMED_EXP              PAGE 4-61                │
│                 PAC_RSP_RCV                     PAGE 4-41                │
│                 TC.CPMGR.RCV.NORM_RQ            PAGE 4-40                │
│                 TC.CPMGR.RCV_CHECKS             PAGE 4-38                │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                                        /*
                ┌──────────────────────────────────────┐
                │           ESTABLISH TCCB             │
                └──────────────────────────────────────┘
                                                                        */
TCCB_PTR = SCB.TC_CB_PTR;
                                                                        /*
                ┌──────────────────────────────────────┐
                │       USAGE AND STATE CHECKS         │
                └──────────────────────────────────────┘
                                                                        */
SELECT ANYORDER(TC.CPMGR.RCV_CHECKS);                   /* PAGE 4-38   */
.
. WHEN(NEG_RSP)
.   DO;
.   . SEND MU TO TC.CPMGR.SEND;                         /* PAGE 4-31   */
.   . RETURN;
.   END;
.
. WHEN(DISCARD_MU)
.   DO;
.   . DISCARD MU;
.   . RETURN;
.   END;
.
. OTHERWISE                                             /* CONVERT_TO_EXR OR GOOD */
.
.   CALL FSM_CNTL_IMMED_EXP;                            /* PAGE 4-61   */
.
END;
                                                                        /*
                ┌──────────────────────────────────────┐
                │ CHECK FOR DFC OR FMD RU THAT WAS PASSED BY A │
                │                CLEAR                 │
                └──────────────────────────────────────┘
                                                                        */
IF RU_CTGY = (DFC | FMD) THEN
  DO;
  . CALL #FSM_DT;                                       /* PAGES 4-62 TO 4-67 */
  . IF RECEIVE_CHECK = DISCARD_MU THEN
  .   DO;
  .   . DISCARD MU;
  .   . RETURN;
  .   END;
  END;
                                                                        /*
                ┌──────────────────────────────────────┐
                │            DONE IF IPR               │
                └──────────────────────────────────────┘
                                                                        */
IF PAC_RSP_RCV = IPR_DISCARDED THEN                     /* PAGE 4-41   */
  RETURN;
                                                                        /*
                ┌──────────────────────────────────────┐
                │ META-IMPLEMENTATION  REQUIRES THAT  SNF'S  BE │
                │ REESTABLISHED IN A TYPE 1 NODE       │
                └──────────────────────────────────────┘
                                                                        */
IF NCB.PU_TYPE = PU_T1 THEN
  CALL ADD_SNF_FOR_T1;                                  /* PAGE 4-40   */
```

```
                                                                        /*
                        ROUTE
                                                                        */
SELECT ANYORDER(RU_CTGY);
.
. WHEN(SC)
.   SEND RU TO TC.SC_RCV;                          /* PAGE 4-44          */
.
. WHEN(DFC,FMD)
.   SELECT ANYORDER;
.   .
.   . WHEN(EFI = EXPEDITED)
.   .   SEND RU TO DFC.RCV;                        /* CHAPTER 5          */
.   .
.   . WHEN(EFI = NORMAL & RRI = RQ)
.   .   DO;
.   .   . CALL TC.CPMGR.RCV.NORM_RQ;               /* PAGE 4-40          */
.   .   . INSERT RU LAST IN SCB.Q_TC_TO_DFC;
.   .   END;
.   .
.   . WHEN(EFI = NORMAL & RRI = RSP)
.   .   IF QRI =  ¬QR THEN
.   .     SEND RU TO DFC.RCV;                      /* CHAPTER 5          */
.   .   ELSE
.   .     INSERT RU LAST IN SCB.Q_TC_TO_DFC;
.   .
.   END;
.
. WHEN(NC)
.   DO;
.   . IF RRI = RQ &  ¬RQR THEN                     /* APPENDIX B         */
.   .   DO;
.   .   . CALL CHANGE_RU_TO_NEG_RSP(X'1007');      /* APPENDIX B         */
.   .   .                                          /* CATEGORY NOT SUPPORTED */
.   .   . SEND RU TO TC.CPMGR.SEND;
.   .   END;
.   . ELSE
.   .   DISCARD RU;
.   END;
.
END;

  RETURN;

END TC.CPMGR.RCV;
```

```
TC.CPMGR.RCV_CHECKS: PROCEDURE RETURNS(BIT(2));
                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  USAGE CHECKS ARE MADE FOR VALID  RU LENGTH AND VALID │
 │                 ON  A NORMAL  FLOW  REQUEST.  IF CRYPTOGRAPHY IS TO  │
 │                 OPTIONAL CHECK IS MADE THAT EDI IS SET WHEN ENCIPHERING│
 │                 AND THE LENGTH OF THE RU IS CHECKED  FOR BEING A MULTIPLE│
 │                 SESSION ACTIVATION  STATE IS CHECKED AND  AN OPTIONAL │
 │                 FOR A MAU SERVICES MANAGER FAILURE.  THE PROCEDURE VERIFIES│
 │                 FSM'S ARE IN THE PROPER STATE.                       │
 │                                                                      │
 │      INPUT:     RQ|RSP FROM TC.CPMGR.RCV                             │
 │                                                                      │
 │      OUTPUT:    NEG_RSP, CONVERT_TO_EXR, DISCARD_MU, OR GOOD DEPENDING ON THE ERROR,│
 │                 IF ANY.   FOR A  NEGATIVE RESPONSE  OR EXR,  THE REQUEST  IS CHANGED│
 │                 BEFORE THE PROCEDURE RETURNS.                        │
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 TC.CPMGR.RCV                    PAGE 4-36            │
 │                                                                      │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
 │                 FSM_CNTL_IMMED_EXP              PAGE 4-61            │
 │                 UPM_MAU_INOPERATIVE             PAGE 4-42            │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                                        /*
                 ┌──────────────────────────────────────┐
                 │           ACTIVE SESSION             │
                 └──────────────────────────────────────┘
                                                                        */
IF #FSM_SESS ¬= ACTIVE THEN                          /* CHAPTER 13     */
  RETURN(DISCARD_MU);
                                                                        /*
                 ┌──────────────────────────────────────┐
                 │         SEQUENCE NUMBERS             │
                 └──────────────────────────────────────┘
                                                                        */
IF EFI = NORMAL & RRI = RQ &
     SCB.SQN_USAGE = SEQUENCE_NUMBERS &
     NCB.PU_TYPE = (PU_T2 | PU_T4 | PU_T5) THEN
  IF SNF = SCB.SQN_RCV_CNT + 1 THEN
    SCB.SQN_RCV_CNT = SCB.SQN_RCV_CNT + 1;
  ELSE
    DO;
    . CALL CHANGE_MU_TO_EXR(X'2001');                /* APPENDIX B     */
    .                                                /* SEQUENCE NUMBER*/
    . RETURN(CONVERT_TO_EXR);
    END;
                                                                        /*
                 ┌──────────────────────────────────────┐
                 │           USAGE CHECKS               │
                 └──────────────────────────────────────┘
                                                                        */
                                                                        /*
                 ┌──────────────────────────────────────┐
                 │         VALID RU LENGTH              │
                 └──────────────────────────────────────┘
                                                                        */
IF EFI = NORMAL &
     TCCB.MAX_RCV_RU_SIZE ¬= NOT_SPECIFIED &
     ((DCF - RH_LENGTH) > TCCB.MAX_RCV_RU_SIZE) THEN
  IF RRI = RQ THEN
    DO;
    . CALL CHANGE_MU_TO_EXR(X'1002');                /* APPENDIX B     */
    .                                                /* RU LENGTH ERROR*/
    . RETURN(CONVERT_TO_EXR);
    END;
  ELSE                                               /* RESPONSE       */
    DO;
    . CALL UPM_LOG('RU LENGTH ERROR');               /* APPENDIX B     */
    . RETURN(DISCARD_MU);
    END;
```

```
                                                                        /*
                    ┌──────────────────────────────────────────────┐
                    │ IF THE WINDOW SIZE IS SPECIFIED TO BE 0 WHEN │
                    │ THE TS PROFILE INDICATES THAT PACING MAY BE  │
                    │ USED, AND A REQUEST IS RECEIVED WITH PI=PAC, │
                    │ THEN THE RECEIVER RETURNS EITHER A PACING    │
                    │ RESPONSE OR A NEGATIVE RESPONSE WITH SENSE   │
                    │ CODE FOR PACING NOT SUPPORTED. THIS IS THE   │
                    │ OPTIONAL CHECK FOR RETURNING A NEGATIVE      │
                    │ RESPONSE.                                    │
                    └──────────────────────────────────────────────┘
                                                                        */
  IF RRI = RQ & EFI = NORMAL &
       PI = PAC & TCCB.RCV_PACING = NO &
       (SCB.TS_PROFILE = (PROFILE_2 | PROFILE_3 | PROFILE_4 | PROFILE_7 | PROFILE_17)) THEN
    DO;
    . IF ¬RQN THEN                                          /* APPENDIX B          */
    .   DO;
    .   . CALL CHANGE_RU_TO_NEG_RSP(X'4008');               /* APPENDIX B          */
    .   .                                                   /* PACING NOT SUPPORTED*/
    .   . RETURN(NEG_RSP);
    .   END;
    . ELSE
    .   RETURN(DISCARD_RU);
    END;
                                                                        /*
                    ┌──────────────────────────────────────────────┐
                    │          DECIPHERING FUNCTION CHECKS         │
                    └──────────────────────────────────────────────┘
                                                                        */
  IF RRI = RQ & EFI = NORMAL &
       (SCB.CRYPTOGRAPHY_SESSION_LEVEL = MANDATORY |
       (SCB.CRYPTOGRAPHY_SESSION_LEVEL = SELECTIVE & EDI = ED)) &
       RU_CTGY = FMD &
       DCF ¬= RH_LENGTH & SDI = ¬SD THEN
    DO;
    . IF EDI = ¬ED THEN                                     /* OPTIONAL CHECK FOR  */
    .                                                       /* MANDATORY ENCRYPTION*/
    .   DO;                                                 /* AND EDI NOT SET     */
    .   . CALL CHANGE_RU_TO_EXR(X'0809');                   /* APPENDIX B          */
    .   .                                                   /* MODE INCONSISTENCY  */
    .   . RETURN(CONVERT_TO_EXR);
    .   END;
    .
    . IF MODULO(DCF - RH_LENGTH,8) ¬= 0 THEN                /* APPENDIX B          */
    .   DO;
    .   . CALL CHANGE_RU_TO_EXR(X'1001');                   /* APPENDIX B          */
    .   .                                                   /* RU DATA ERROR       */
    .   . RETURN(CONVERT_TO_EXR);
    .   END;
    END;
                                                                        /*
                    ┌──────────────────────────────────────────────┐
                    │                STATE CHECKS                  │
                    └──────────────────────────────────────────────┘
                                                                        */
                                                                        /*
                    ┌──────────────────────────────────────────────┐
                    │     OPTIONAL CHECK FOR NAU SERVICES MANAGER   │
                    │                   FAILURE                    │
                    └──────────────────────────────────────────────┘
                                                                        */
  IF UPM_NAU_INOPERATIVE = TRUE THEN                        /* PAGE 4-42           */
    DO;
    . IF ¬RQN THEN                                          /* APPENDIX B          */
    .   DO;
    .   . CALL CHANGE_RU_TO_NEG_RSP(X'8003');               /* APPENDIX B          */
    .   .                                                   /* NAU INOPERATIVE     */
    .   . RETURN(NEG_RSP);
    .   END;
    . ELSE
    .   RETURN(DISCARD_RU);
    END;
                                                                        /*
                    ┌──────────────────────────────────────────────┐
                    │ COMPLIANCE WITH THE IMMEDIATE REQUEST MODE    │
                    │ PROTOCOL IS CHECKED AND IT IS CHECKED THAT    │
                    │ DATA TRAFFIC AND CRV FSM'S ARE PROPERLY       │
                    │ ESTABLISHED FOR HIGHER LEVEL RU'S TO FLOW.    │
                    │ THE DATA TRAFFIC FSM CHECK IS REQUIRED ONLY   │
                    │ IN A PRIMARY HALF-SESSION THAT ALLOWS THE     │
                    │ SENDING OF CLEAR AND THE CRYPTOGRAPHY FSM     │
                    │ CHECK IS OPTIONAL IN A SECONDARY.             │
                    └──────────────────────────────────────────────┘
                                                                        */
  IF SEND_OR_RECEIVE_CHECK(FSM_CNTL_IMMED_EXP) |            /* PAGE 4-61            */
       SEND_OR_RECEIVE_CHECK(#FSM_DT) |                     /* PAGES 4-62 TO 4-67  */
       SEND_OR_RECEIVE_CHECK(#FSM_CRV) THEN                 /* PAGES 4-70 TO 4-71  */
    RETURN(RECEIVE_CHECK);

  RETURN(GOOD);

END TC.CPMGR.RCV_CHECKS;
```

```
TC.CPMGR.RCV.NORM_RQ: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  DECIPHER A NORMAL-FLOW REQUEST IF NECESSARY AND UPDATE PACING FSM
  │
  │      INPUT:     NORMAL-FLOW REQUEST
  │
  │      OUTPUT:    NORMAL-FLOW REQUEST OR EXR AS RETURNED BY DECIPHER
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 TC.CPMGR.RCV                    PAGE 4-36
  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):
  │                 DECIPHER                        PAGE 4-42
  │                 FSM_PAC_RQ_RCV                  PAGE 4-61
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF (SCB.CRYPTOGRAPHY_SESSION_LEVEL = MANDATORY |
      (SCB.CRYPTOGRAPHY_SESSION_LEVEL = SELECTIVE & EDI = ED)) &
      RU_CTGY = FMD & DCF ¬= RH_LENGTH & SDI = ¬SD THEN
    CALL DECIPHER;                                      /* PAGE 4-42        */

  IF TCCB.RCV_PACING = YES THEN
    CALL FSM_PAC_RQ_RCV;                                /* PAGE 4-61        */

  RETURN;

END TC.CPMGR.RCV.NORM_RQ;



ADD_SNF_FOR_T1: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  CREATE SNF VALUES FOR DFC TO PROCESS
  │
  │      INPUT:     RQ|RSP
  │
  │      OUTPUT:    RQ|RSP WITH SNF FILLED IN
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 TC.CPMGR.RCV                    PAGE 4-36
  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):
  │                 UPM_ID_EXP                      PAGE 4-42
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SELECT ANYORDER;
  .
  . WHEN(EFI = EXPEDITED & RRI = RQ)
  .   SNF = UPM_ID_EXP;                                 /* PAGE 4-42        */
  .
  . WHEN(EFI = EXPEDITED & RRI = RSP)
  .   SNF = SCB.SEND_EXP_SNF;
  .
  . WHEN(EFI = NORMAL & RRI = RQ)
  .   DO;
  .   . SCB.SQN_RCV_CNT = SCB.SQN_RCV_CNT + 1;
  .   . SNF = SCB.SQN_RCV_CNT;
  .   END;
  .
  . WHEN(EFI = NORMAL & RRI = RSP)
  .   SNF = SCB.SEND_NORM_SNF;
  END;

  RETURN;

END ADD_SNF_FOR_T1;
```

```
PAC_RSP_RCV: PROCEDURE RETURNS(BIT(1));

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  IF MESSAGE UNIT IS AN IPR OR  RESPONSE WITH PI=PAC, THE RECEIPT OF A
 │                 PAC_RSP IS NOTED.   IF THE MESSAGE UNIT  IS AN IPR, IT  IS DISCARDED
 │                 AND THE  RETURN CODE  IS SET TO  INDICATE THIS ACTION.   IF IT  IS A
 │                 RESPONSE WITH PI  = PAC, PI IS SET  TO ¬PAC AND THE  PIU IS RETURNED
 │                 FOR FURTHER PROCESSING.
 │
 │      INPUT:     RQ|RSP
 │
 │      OUTPUT:    RQ|RSP OR IPR_DISCARDED INDICATION
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 TC.CPMGR.RCV                        PAGE 4-36
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 FSM_PAC_RQ_SEND                     PAGE 4-60
 │                 IPR_CHECK                           PAGE 4-58
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF TCCB.SEND_PACING = YES THEN
    DO;
    . IF RRI = RSP & PI = PAC THEN
    .   DO;
    .   . CALL FSM_PAC_RQ_SEND;                          /* PAGE 4-60          */
    .   .
    .   . IF IPR_CHECK = YES THEN                        /* PAGE 4-58          */
    .   .   DO;
    .   .   . DISCARD MU;
    .   .   . RETURN(IPR_DISCARDED);
    .   .   END;
    .   . ELSE
    .   .   RETURN(¬IPR_DISCARDED);
    .   END;
    END;

  ELSE                                                   /* OPTIONAL CHECK FOR IPR */
                                                         /* WHEN PACING NOT IN USE */
    IF IPR_CHECK = YES THEN                              /* PAGE 4-58          */
      DO;
      . DISCARD MU;
      . RETURN(IPR_DISCARDED);
      END;

  RETURN(¬IPR_DISCARDED);

END PAC_RSP_RCV;
```

```
DECIPHER: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO DECIPHER AN ENCRYPTED MESSAGE                             │
  │                                                                              │
  │      INPUT:     ENCIPHERED MU                                                │
  │                                                                              │
  │      OUTPUT:    DECIPHERED MU OR AN EXR                                      │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 TC.CPMGR.RCV.NORM_RQ              PAGE 4-40                  │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 UPM_DECIPHER                      PAGE 4-43                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL PAD_COUNT FIXED BIN(15);
  DCL PAD_COUNT_BYTES CHAR(2) BASED(ADDR(PAD_COUNT));

  IF UPM_DECIPHER = NG THEN                                  /* PAGE 4-43       */
    DO;
    . CALL CHANGE_MU_TO_EXR(X'0848');                        /* APPENDIX B      */
    .                                                        /* CRYPTOGRAPHY FUNCTION */
    .                                                        /* INOPERATIVE     */
    . RETURN;
    END;

  IF PDI = PD THEN
    DO;
    . PAD_COUNT = 0;                                         /* THESE LINES OF CODE */
    . PAD_COUNT_BYTES(0:0) = RU(DCF - 4:DCF - 4);            /* EXTRACT THE PAD COUNT */
    .                                                        /* FROM THE LAST BYTE OF */
    .                                                        /* THE RU & ASSIGN IT TO */
    .                                                        /* PAD_COUNT       */
    . IF PAD_COUNT = 0 | PAD_COUNT > 7 THEN
    .   CALL CHANGE_MU_TO_EXR(X'1001');                      /* APPENDIX B      */
    .                                                        /* RU DATA ERROR   */
    . ELSE
    .   DCF = DCF - PAD_COUNT;
    END;

  RETURN;

END DECIPHER;



UPM_NAU_INOPERATIVE: PROCEDURE RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  DETERMINES IF A NAU SERVICES MANAGER HAS FAILED              │
  │                                                                              │
  │      INPUT:     NONE                                                         │
  │                                                                              │
  │      OUTPUT:    TRUE IF THE NAU SERVICES MANAGER IS INOPERATIVE; OTHERWISE FALSE │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 TC.CPMGR.RCV_CHECKS               PAGE 4-38                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  RETURN(FALSE);

END UPM_NAU_INOPERATIVE;



UPM_ID_EXP: PROCEDURE RETURNS(FIXED BIN(16));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  GENERATES A UNIQUE 16-BIT ID FOR THE SESSION                 │
  │                                                                              │
  │      INPUT:     NONE                                                         │
  │                                                                              │
  │      OUTPUT:    16-BIT ID                                                    │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 ADD_SNF_FOR_T1                    PAGE 4-40                  │
  │                 TC.SC.SEND                        PAGE 4-47                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SCB.RCV_EXP_SNF = SCB.RCV_EXP_SNF + 1;   /* IMPLEMENTATIONS MAY GENERATE ANY UNIQUE VALUE */

  RETURN(SCB.RCV_EXP_SNF);

END UPM_ID_EXP;
```

```
UPM_DECIPHER: PROCEDURE RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  DECIPHERS THE RU USING THE DES ALGORITHM                     │
  │                                                                              │
  │      INPUT:     MU TO BE DECIPHERED                                          │
  │                                                                              │
  │      OUTPUT:    OK OR NG.  IF OK, DECIPHERED MU; OTHERWISE, MU AS PASSED TO IT│
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 DECIPHER                                  PAGE 4-42          │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  RETURN(OK);

END UPM_DECIPHER;
```

```
TC.SC.RCV: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  CHECKS  THAT THE  FUNCTION  IS SUPPORTED,  AND  MAKES STATE  RECEIVE │
   │                 CHECKS.   IF THE  CHECKS  FAIL, THE  MESSAGE  UNIT  IS DISCARDED  OR │
   │                 RETURNED AS A -RSP.  OTHERWISE, MESSAGE UNIT IS ROUTED TO THE FSM'S. │
   │                                                                      │
   │      INPUT:     RQ|RSP FROM TC.CPMGR.RCV                             │
   │                                                                      │
   │      OUTPUT:    RQ|RSP TO NAU.SVC_MGR                                │
   │                                                                      │
   │                 WHEN IT IS SENT ON, A REQUEST HAS THE FOLLOWING FIELDS SET:  SESSION │
   │                 IDENTIFICATION (SCB_PTR), SNF (IDENTIFIER), RRI=RQ, RU_CTGY=SC, RU.  │
   │                                                                      │
   │                 A RESPONSE  HAS THE  FOLLOWING  FIELDS  SET:  SESSION  IDENTIFICATION │
   │                 (SCB_PTR),  SNF (IDENTIFIER),  RRI=RSP, RTI,  SDI  (SAME SETTING  AS │
   │                 RTI), RU                                             │
   │                                                                      │
   │      NOTE:      THE  MU  IS  SENT  TO THE  APPROPRIATE  SERVICES  MANAGER  FOR  THIS │
   │                 HALF-SESSION:  LU.SVC_MGR, PU.SVC_MGR (CHAPTER  11), OR SSCP.SVC_MGR │
   │                 (CHAPTER 7).                                         │
   │                                                                      │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
   │                 TC.SC.RCV_CHECKS                    PAGE 4-45        │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  SELECT ANYORDER(TC.SC.RCV_CHECKS);                    /* PAGE 4-45      */
  .
  . WHEN(NEG_RSP)
  .   SEND MU TO TC.CPMGR.SEND;                         /* PAGE 4-31      */
  .
  . WHEN(DISCARD_MU)
  .   DISCARD MU;
  .
  . WHEN(GOOD)
  .   DO;
  .   . CALL #FSM_DT;                                   /* PAGES 4-62 TO 4-67 */
  .   . CALL #FSM_STSN;                                 /* PAGE 4-68      */
  .   . CALL #FSM_RQR;                                  /* PAGE 4-67      */
  .   . CALL #FSM_CRV;                                  /* PAGES 4-70 TO 4-71 */
  .   . SEND MU TO #SVC_MGR;                           /* SEE NOTE       */
  .   END;
  .
  END;

  RETURN;

END TC.SC.RCV;
```

```
TC.SC.RCV_CHECKS: PROCEDURE RETURNS(BIT(2));

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  VERIFIES  THAT   THE  FUNCTION  REQUESTED   IS  SUPPORTED   BY  THIS │
 │                 HALF-SESSION AND  THAT ALL  FSM'S ARE  IN THE  PROPER STATE  FOR THE │
 │                 MESSAGE UNIT TO BE PROCESSED                                 │
 │                                                                              │
 │      INPUT:     MU                                                           │
 │                                                                              │
 │      OUTPUT:    NEG_RSP, DISCARD_MU, OR GOOD.  IF  AN INVALID STATE CONDITION EXISTS │
 │                 AND THE MU  IS A REQUEST, IT  IS CHANGED TO A  NEGATIVE RESPONSE AND │
 │                 NEG_RSP IS RETURNED.   IF AN INVALID STATE CONDITION  EXISTS AND THE │
 │                 MU IS A  RESPONSE, DISCARD_MU IS RETURNED.  IF  ALL STATE CONDITIONS │
 │                 ARE VALID, GOOD IS RETURNED.                                 │
 │                                                                              │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
 │                 TC.SC.RCV                          PAGE 4-44                 │
 │                                                                              │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
 │                 TC.SC_FORMAT_CHECK                 PAGE 4-46                 │
 │                 TC.SC_FUNCTION_SUPPORTED           PAGE 4-50                 │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF TC.SC_FUNCTION_SUPPORTED = NG THEN                    /* PAGE 4-50        */
    DO;
    . IF RRI = RQ THEN
    .   DO;
    .   . CALL CHANGE_MU_TO_NEG_RSP(X'1003');              /* APPENDIX B       */
    .   . RETURN(NEG_RSP);
    .   END;
    . ELSE
    .   RETURN(DISCARD_MU);
    END;

  IF TC.SC_FORMAT_CHECK = NG |                             /* PAGE 4-46        */
       SEND_OR_RECEIVE_CHECK(#FSM_STSN) |                  /* PAGE 4-68        */
       SEND_OR_RECEIVE_CHECK(#FSM_DT) |                    /* PAGES 4-62 TO 4-67 */
       SEND_OR_RECEIVE_CHECK(#FSM_CRV) |                   /* PAGES 4-70 TO 4-71 */
       SEND_OR_RECEIVE_CHECK(#FSM_RQR) THEN                /* PAGE 4-67        */
    RETURN(RECEIVE_CHECK);
  ELSE
    RETURN(GOOD);

END TC.SC.RCV_CHECKS;
```

```
TC.SC_FORMAT_CHECK: PROCEDURE RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  CHECKS THE RH BITS OF THE REQUEST OR RESPONSE.               │
  │                                                                              │
  │      INPUT:     SC RQ&RSP                                                    │
  │                                                                              │
  │      OUTPUT:    OK IF ALL  BITS ARE PROPERLY SET;  OTHERWISE, NG.  IF OK,  THE RQ OR
  │                 RSP IS RETURNED AS IT WAS RECEIVED  BY THIS PROCEDURE; IF NG, THE RQ
  │                 IS CHANGED TO -RSP(4001).                                    │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 TC.SC.RCV_CHECKS                    PAGE 4-45                │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SELECT ANYORDER(RRI);
  .
  . WHEN(RQ)
  .   IF RU_CTGY = SC &
  .        FI = B'1' &
  .        SDI =  ¬SD &
  .        BCI = BC &
  .        ECI = EC &
  .        DR1I = DR1 &
  .        DR2I =  ¬DR2 &
  .        ERI =  ¬ER &
  .        QRI =  ¬QR &
  .        PI =  ¬PAC &
  .        BBI =  ¬BB &
  .        EBI =  ¬EB &
  .        CDI =  ¬CD &
  .        CSI = CODE0 &
  .        EDI =  ¬ED &
  .        PDI =  ¬PD THEN
  .     RETURN(OK);
  .   ELSE
  .     DO;
  .     . IF  ¬RQN THEN                                  /* APPENDIX B          */
  .     .   DO;
  .     .   . CALL CHANGE_RU_TO_NEG_RSP(X'4001');        /* APPENDIX B          */
  .     .   .                                            /* INVALID SC RH       */
  .     .   . RECEIVE_CHECK = NEG_RSP;
  .     .   END;
  .     . ELSE
  .     .   RECEIVE_CHECK = DISCARD_RU;
  .     . RETURN(NG);
  .     END;
  .
  . WHEN(RSP)
  .   IF RU_CTGY = SC &
  .        FI = B'1' &
  .        BCI = BC &
  .        ECI = EC &
  .        DR1I = DR1 &
  .        DR2I =  ¬DR2 &
  .        QRI =  ¬QR &
  .        PI =  ¬PAC THEN
  .     RETURN(OK);
  .   ELSE
  .     DO;
  .     . RECEIVE_CHECK = DISCARD_RU;
  .     . RETURN(NG);
  .     END;
  .
  END;

END TC.SC_FORMAT_CHECK;
```

```
TC.SC.SEND: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  CHECKS THAT THE  FUNCTION IS SUPPORTED AND MAKES  STATE SEND CHECKS.
│                 IF THE CHECKS FAIL,  A SEND-CHECK SENSE DATA IS SENT  TO THE SENDING
│                 PROCEDURE, A  NAU.SVC_MGR.  OTHERWISE, THE  MESSAGE UNIT IS  SENT TO
│                 THE PROPER FSM.   AFTER AN SNF IS FILLED IN  FOR EXPEDITED REQUESTS,
│                 THE MU IS SENT ON.
│
│      INPUT:     RQ|RSP FROM NAU.SVC_MGR
│
│                 REQUESTS HAVE THE FOLLOWING FIELDS SET:  RRI=RQ, RU.
│
│                 RESPONSES HAVE  THE FOLLOWING  FIELDS SET:   SNF, RRI=RSP,  RTI, SDI
│                 (SAME SETTING AS RTI), RU
│
│      OUTPUT:    RQ|RSP TO CPMGR.SEND
│
│      REFERS TO THE FOLLOWING PROCEDURE(S):
│                 SC_FORMAT_SET                    PAGE 4-49
│                 TC.CPMGR.SEND_CHECKS             PAGE 4-32
│                 TC.SC.SEND_CHECKS                PAGE 4-48
│                 UPM_ID_EXP                       PAGE 4-42
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  MUCB.SEND_CHECK_SENSE = X'0000';

  IF TC.SC.SEND_CHECKS = NG |                                  /* PAGE 4-48   */
       TC.CPMGR.SEND_CHECKS = NG THEN                          /* PAGE 4-32   */
    DO;
    . IF MUCB.SEND_CHECK_SENSE ¬= X'0000' THEN
    .   SEND SEND_CHECK TO SENDING_PROCEDURE;
    . ELSE
    .   DISCARD MU;
    END;

  ELSE
    DO;
                                                                        /*
                  ┌──────────────────────────────────┐
                  │          UPDATE FSM'S            │
                  └──────────────────────────────────┘
                                                                        */
    . CALL #FSM_DT;                                            /* PAGES 4-62 TO 4-67 */
    . CALL #FSM_STSN;                                          /* PAGE 4-68   */
    . CALL #FSM_RQR;                                           /* PAGE 4-67   */
    . CALL #FSM_CRV;                                           /* PAGES 4-70 TO 4-71 */
                                                                        /*
                  ┌──────────────────────────────────┐
                  │ ASSIGN VALUE TO SNF FOR REQUESTS AND SAVE THE  │
                  │ SNF VALUE IF IT IS A CLEAR REQUEST             │
                  └──────────────────────────────────┘
                                                                        */
    . IF RRI = RQ THEN
    .   DO;
    .   .
    .   . SNF = UPM_ID_EXP;                                    /* PAGE 4-42   */
    .   .
    .   . IF RQ_CODE = CLEAR THEN
    .   .   SCB.LAST_CLEAR_SNF = SNF;
    .   END;
                                                                        /*
                  ┌──────────────────────────────────┐
                  │          SET RH BITS             │
                  └──────────────────────────────────┘
                                                                        */
    . CALL SC_FORMAT_SET;                                      /* PAGE 4-49   */
    .
    . SEND MU TO TC.CPMGR.SEND;                                /* PAGE 4-31   */
    .
    END;

  RETURN;

END TC.SC.SEND;
```

```
TC.SC.SEND_CHECKS: PROCEDURE RETURNS(BIT(1));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  VERIFIES  THAT   THE  FUNCTION  REQUESTED   IS  SUPPORTED  BY  THIS  │
   │                 HALF-SESSION AND THAT THE APPROPRIATE FSM'S  ARE IN THE PROPER STATE  │
   │                 FOR THE MESSAGE UNIT TO BE PROCESSED                     │
   │                                                                          │
   │      INPUT:     MU                                                       │
   │                                                                          │
   │      OUTPUT:    NG IF  AN INVALID  STATE CONDITION  EXISTS; OTHERWISE,  OK.  IF  NG,  │
   │                 SEND_CHECK_SENSE IS SET.                                 │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 TC.SC.SEND                        PAGE 4-47              │
   │                                                                          │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
   │                 FSM_CNTL_IMMED_EXP                PAGE 4-61              │
   │                 TC.SC_FUNCTION_SUPPORTED          PAGE 4-50              │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF TC.SC_FUNCTION_SUPPORTED = NG THEN                    /* PAGE 4-50       */
    DO;
    . MUCB.SEND_CHECK_SENSE = X'1003';                     /* FUNCTION NOT SUPPORTED */
    . RETURN(NG);
    END;

  IF #FSM_SESS ¬= ACTIVE THEN
    DO;
    . MUCB.SEND_CHECK_SENSE = X'8005';                     /* NO SESSION      */
    . RETURN(NG);
    END;

  IF SEND_OR_RECEIVE_CHECK(FSM_CNTL_IMMED_EXP) |           /* PAGE 4-61       */
       SEND_OR_RECEIVE_CHECK(#FSM_DT) |                    /* PAGES 4-62 TO 4-67 */
       SEND_OR_RECEIVE_CHECK(#FSM_CRV) |                   /* PAGES 4-70 TO 4-71 */
       SEND_OR_RECEIVE_CHECK(#FSM_STSN) |                  /* PAGE 4-68       */
       SEND_OR_RECEIVE_CHECK(#FSM_RQR) THEN                /* PAGE 4-67       */
    RETURN(NG);
  ELSE
    RETURN(OK);

END TC.SC.SEND_CHECKS;
```

```
SC_FORMAT SET: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  SETS THE RH BITS OF THE REQUEST OR RESPONSE.          │
   │                                                                      │
   │     INPUT:     SC RQ|RSP                                             │
   │                                                                      │
   │     OUTPUT:    SC RQ|RSP WITH RH BITS PROPERLY SET                   │
   │                                                                      │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
   │                TC.SC.SEND                          PAGE 4-47         │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  SELECT ANYORDER(RRI);
  .
  . WHEN(RQ)
  .   DO;
  .   . EFI = EXPEDITED;
  .   . RU_CTGY = SC;
  .   . FI = B'1';
  .   . SDI =  ¬SD;
  .   . BCI = BC;
  .   . ECI = EC;
  .   . DR1I = DR1;
  .   . DR2I =  ¬DR2;
  .   . ERI =  ¬ER;
  .   . QRI =  ¬QR;
  .   . PI =  ¬PAC;
  .   . BBI =  ¬BB;
  .   . EBI =  ¬EB;
  .   . CDI =  ¬CD;
  .   . CSI = CODE0;
  .   . EDI =  ¬ED;
  .   . PDI =  ¬PD;
  .   END;
  .
  . WHEN(RSP)
  .   DO;
  .   . EFI = EXPEDITED;
  .   . RU_CTGY = SC;
  .   . FI = B'1';
  .   . BCI = BC;
  .   . ECI = EC;
  .   . DR1I = DR1;
  .   . DR2I =  ¬DR2;
  .   . QRI =  ¬QR;
  .   . PI =  ¬PAC;
  .   END;
  .
  END;

END SC_FORMAT_SET;
```

```
TC.SC_FUNCTION_SUPPORTED: PROCEDURE RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  VERIFIES  THAT   THE  FUNCTION  REQUESTED   IS  SUPPORTED   BY  THIS  │
  │                 HALF-SESSION.                                                │
  │                                                                              │
  │      INPUT:     MU                                                           │
  │                                                                              │
  │      OUTPUT:    NG IF NOT SUPPORTED; OTHERWISE, OK.                          │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 TC.SC.RCV_CHECKS                  PAGE 4-45                  │
  │                 TC.SC.SEND_CHECKS                 PAGE 4-48                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │        DETERMINE IF RQ_CODE IS SUPPORTED │
                  └──────────────────────────────────────────┘
                                                                                */
  IF (RQ_CODE = CLEAR & SCB.SC_CLEAR ¬= ALLOWED) |
       (RQ_CODE = SDT & SCB.SC_SDT ¬= ALLOWED) |
       (RQ_CODE = STSN & SCB.SC_STSN ¬= ALLOWED) |
       (RQ_CODE = CRV & (SCB.SC_CRV ¬= ALLOWED |
         SCB.CRYPTOGRAPHY_SESSION_LEVEL = NONE)) |
       (RQ_CODE = RQR & SCB.SC_RQR ¬= ALLOWED) THEN
    RETURN(NG);
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │  DETERMINE IF  THIS HALF-SESSION  CAN SEND  OR  │
                  │  RECEIVE  THE  RQ  OR  RSP  FOR  THE  RQ_CODE.  │
                  │  PRIMARY HALF-SESSIONS SEND  CLEAR, SDT, STSN,  │
                  │  AND CRV.  SECONDARY HALF-SESSIONS SEND RQR     │
                  └──────────────────────────────────────────┘
                                                                                */
  SELECT ANYORDER;
  .
  . WHEN((SCB.HALF_SESSION = PRIMARY & MUCB.DIRECTION = RECEIVE) |
  .      (SCB.HALF_SESSION = SECONDARY & MUCB.DIRECTION = SEND))
  .
  .   SELECT ANYORDER;
  .   .
  .   . WHEN(RRI = RQ & RQ_CODE = RQR)
  .   .   RETURN(OK);
  .   .
  .   . WHEN(RRI = RQ & RQ_CODE = (CLEAR | SDT | STSN | CRV))
  .   .   RETURN(NG);
  .   .
  .   . WHEN(RRI = RSP & RQ_CODE = RQR)
  .   .   RETURN(NG);
  .   .
  .   . WHEN(RRI = RSP & RQ_CODE = (CLEAR | SDT | STSN | CRV))
  .   .   RETURN(OK);
  .   .
  .   END;
  .
  . WHEN((SCB.HALF_SESSION = PRIMARY & MUCB.DIRECTION = SEND) |
  .      (SCB.HALF_SESSION = SECONDARY & MUCB.DIRECTION = RECEIVE))
  .
  .   SELECT ANYORDER;
  .   .
  .   . WHEN(RRI = RQ & RQ_CODE = RQR)
  .   .   RETURN(NG);
  .   .
  .   . WHEN(RRI = RQ & RQ_CODE = (CLEAR | SDT | STSN | CRV))
  .   .   RETURN(OK);
  .   .
  .   . WHEN(RRI = RSP & RQ_CODE = RQR)
  .   .   RETURN(OK);
  .   .
  .   . WHEN(RRI = RSP & RQ_CODE = (CLEAR | SDT | STSN | CRV))
  .   .   RETURN(NG);
  .   .
  .   END;
  .
  END;

END TC.SC_FUNCTION_SUPPORTED;
```

```
BF.SESSACT.TC.INITIALIZE: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  SETS UP SESSION PARAMETERS THAT ARE NEEDED BY BF.TC.  THIS PROCEDURE
  │                 IS EXECUTED WHEN THE SESSION IS BEING ACTIVATED.
  │
  │      INPUT:     SCB_PTR IS ESTABLISHED
  │
  │      OUTPUT:    UPDATES SCB AND TCCB'S FOR BOUNDARY FUNCTION
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                                                /*
                    ┌──────────────────────────────────────┐
                    │  SET #PC -- THE PATH CONTROL PROCEDURE THAT IS
                    │                      SENT TO
                    └──────────────────────────────────────┘
                                                                                */
  SCB.TC_CB_PTR->TCCB.#PC = PC.VRC.SEND;                    /* CHAPTER 3      */
  SCB.SEC_TO_BF_TC_CB_PTR->TCCB.#PC = BF.PC.SEND;           /* CHAPTER 3      */

  SELECT ANYORDER(SCB.TYPE_OF_SESSION);
  .
  . WHEN(SSCP_PU,SSCP_LU)
  .   DO;
  .   . SCB.TC_CB_PTR->TCCB.SEND_PACING = NO;
  .   . SCB.TC_CB_PTR->TCCB.RCV_PACING = NO;
  .   . SCB.SEC_TO_BF_TC_CB_PTR->TCCB.SEND_PACING = NO;
  .   . SCB.SEC_TO_BF_TC_CB_PTR->TCCB.RCV_PACING = NO;
  .   END;
  .
  . WHEN(LU_LU)
  .   DO;
                                                                                /*
                    ┌──────────────────────────────────────┐
                    │          SECONDARY TO PRIMARY PACING │
                    └──────────────────────────────────────┘
                                                                                */
  .   . IF BIND_RSP.SEC_TO_PRI_STAGING_IND = SEC_TO_PRI_TWO THEN
  .   .   DO;
  .   .   . IF BIND_RSP.SEC_SEND_PACING_CNT ¬= 0 THEN
  .   .   .   SCB.SEC_TO_BF_TC_CB_PTR->TCCB.RCV_PACING = YES;
  .   .   . ELSE
  .   .   .   SCB.SEC_TO_BF_TC_CB_PTR->TCCB.RCV_PACING = NO;
  .   .   .
  .   .   . IF BIND_RSP.PRI_RCV_PACING_CNT ¬= 0 THEN
  .   .   .   DO;
  .   .   .   . SCB.TC_CB_PTR->TCCB.SEND_PACING = YES;
  .   .   .   . SCB.TC_CB_PTR->TCCB.WINDOW_SIZE = BIND_RSP.PRI_RCV_PACING_CNT;
  .   .   .   . NEWLIST SCB.TC_CB_PTR->TCCB.Q_PAC ENTRY_NAME(MU) QUEUE;
  .   .   .   END;
  .   .   . ELSE
  .   .   .   SCB.TC_CB_PTR->TCCB.SEND_PACING = NO;
  .   .   END;
  .   .
  .   . ELSE
  .   .   DO;
  .   .   . SCB.SEC_TO_BF_TC_CB_PTR->TCCB.RCV_PACING = NO;
  .   .   . SCB.TC_CB_PTR->TCCB.SEND_PACING = NO;
  .   .   END;
                                                                                /*
                    ┌──────────────────────────────────────┐
                    │          PRIMARY TO SECONDARY PACING │
                    └──────────────────────────────────────┘
                                                                                */
  .   . IF BIND_RSP.PRI_TO_SEC_STAGING_IND = PRI_TO_SEC_TWO THEN
  .   .   DO;
  .   .   . IF BIND_RSP.SEC_RCV_PACING_CNT ¬= 0 THEN
  .   .   .   DO;
  .   .   .   . SCB.SEC_TO_BF_TC_CB_PTR->TCCB.SEND_PACING = YES;
  .   .   .   . SCB.SEC_TO_BF_TC_CB_PTR->TCCB.WINDOW_SIZE = BIND_RSP.SEC_RCV_PACING_CNT;
  .   .   .   . NEWLIST SCB.SEC_TO_BF_TC_CB_PTR->TCCB.Q_PAC ENTRY_NAME(MU) QUEUE;
  .   .   .   END;
  .   .   . ELSE
  .   .   .   SCB.SEC_TO_BF_TC_CB_PTR->TCCB.SEND_PACING = NO;
  .   .   .
  .   .   . IF BIND_RSP.PRI_SEND_PACING_CNT ¬= 0 THEN
  .   .   .   SCB.TC_CB_PTR->TCCB.RCV_PACING = YES;
  .   .   . ELSE
  .   .   .   SCB.TC_CB_PTR->TCCB.RCV_PACING = NO;
  .   .   END;
  .   .
  .   . ELSE
  .   .   DO;
  .   .   . SCB.SEC_TO_BF_TC_CB_PTR->TCCB.SEND_PACING = NO;
  .   .   . SCB.TC_CB_PTR->TCCB.RCV_PACING = NO;
  .   .   END;
  .   .
  .   END;
  .
  END;

  RETURN;

END BF.SESSACT.TC.INITIALIZE;
```

```
BF.TC.RESET: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  RESETS ALL BF.TC FSM'S, PURGES  BF.TC QUEUES, AND RE-INITIALIZES THE
   │                 SESSION-LEVEL  PACING  COUNT  AND THE  NORMAL-FLOW  SEQUENCE  NUMBER
   │                 FIELDS IN THE SCB
   │
   │      INPUT:     SCB_PTR IS ESTABLISHED
   │
   │      OUTPUT:    RESET TCCB'S AND SCB
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 BF.TC.RCV                          PAGE 4-53
   │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):
   │                 FSM_PAC_RQ_RCV                     PAGE 4-61
   │                 FSM_PAC_RQ_SEND                    PAGE 4-60
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                                        /*
                    ┌──────────────────────────────────┐
                    │           RESET FSM'S            │
                    └──────────────────────────────────┘
                                                                        */
  CALL SCB.TC_CB_PTR->FSM_PAC_RQ_SEND('RESET');              /* PAGE 4-60  */
  CALL SCB.TC_CB_PTR->FSM_PAC_RQ_RCV('RESET');               /* PAGE 4-61  */
  CALL SCB.SEC_TO_BF_TC_CB_PTR->FSM_PAC_RQ_SEND('RESET');    /* PAGE 4-60  */
  CALL SCB.SEC_TO_BF_TC_CB_PTR->FSM_PAC_RQ_RCV('RESET');     /* PAGE 4-61  */
                                                                        /*
                    ┌──────────────────────────────────┐
                    │       EMPTY ALL BF.TC QUEUES     │
                    └──────────────────────────────────┘
                                                                        */
  IF SCB.TC_CB_PTR->TCCB.SEND_PACING = YES THEN
    PURGE SCB.TC_CB_PTR->TCCB.Q_PAC;
  IF SCB.SEC_TO_BF_TC_CB_PTR->TCCB.SEND_PACING = YES THEN
    PURGE SCB.SEC_TO_BF_TC_CB_PTR->TCCB.Q_PAC;
                                                                        /*
                    ┌──────────────────────────────────┐
                    │ RESET THE SESSION-LEVEL PACING  COUNTS TO THE
                    │ CORRESPONDING WINDOW SIZES
                    └──────────────────────────────────┘
                                                                        */
  IF SCB.TC_CB_PTR->TCCB.SEND_PACING = YES THEN
    SCB.TC_CB_PTR->TCCB.PACING_COUNT = SCB.TC_CB_PTR->TCCB.WINDOW_SIZE;
  IF SCB.SEC_TO_BF_TC_CB_PTR->TCCB.SEND_PACING = YES THEN
    SCB.SEC_TO_BF_TC_CB_PTR->TCCB.PACING_COUNT = SCB.SEC_TO_BF_TC_CB_PTR->TCCB.WINDOW_SIZE;
                                                                        /*
                    ┌──────────────────────────────────┐
                    │ RESET THE NORMAL-FLOW SEQUENCE NUMBER FIELDS.
                    │ THE SEND NORMAL-FLOW SNF IS RESET TO 1 IN THE
                    │ BOUNDARY  FUNCTION  BECAUSE OF  THE  NEED  TO
                    │ INCREMENT ON  EBIU TO  AVOID SEQUENCE  NUMBER
                    │ PROBLEMS IF THE NORMAL SEGMENTING SEQUENCE IS
                    │ INTERRUPTED   BY  A   FORWARD  ABORT.    (SEE
                    │ "SEGMENTING"  IN  CHAPTER  3  FOR  ADDITIONAL
                    │ DISCUSSION.)
                    └──────────────────────────────────┘
                                                                        */
  SCB.SQN_SEND_CNT = 1;
  SCB.SQN_RCV_CNT = 0;

  RETURN;

END BF.TC.RESET;
```

```
BF.TC.RCV: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  CHECKS  SESSION  ACTIVATION.   WHEN RECEIVING  FROM  THE  SECONDARY,  │
│                 INSERTS CORRECT SEQUENCE NUMBER OR ID  IN SNF OF TH.  WHEN RECEIVING  │
│                 A REQUEST FROM THE PRIMARY, SAVES  THE CORRECT SEQUENCE NUMBER OR ID  │
│                 FROM THE SNF OF TH.  PROCESSES CLEAR                     │
│                                                                          │
│      INPUT:     RQ|RSP FROM BF.PC OR PC                                  │
│                                                                          │
│      OUTPUT:    RQ|RSP TO BF.NAU                                         │
│                                                                          │
│      NOTE:      #SVC_MGR IS SET BY CSC (CHAPTER  13).  WITHIN THE BOUNDARY FUNCTION,  │
│                 IT IS SET TO EITHER BF.PU.SVC_MGR OR BF.LU.SVC_MGR.      │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 BF.TC.ADD_SNF                    PAGE 4-55               │
│                 BF.TC.RESET                      PAGE 4-52               │
│                 BF.TC.SAVE_SNF                   PAGE 4-56               │
│                 FSM_PAC_RQ_SEND                  PAGE 4-60               │
│                 IPR_CHECK                        PAGE 4-58               │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │     CHECK THAT SESSION IS ACTIVE     │
                    └──────────────────────────────────────┘
                                                                        */
  IF #FSM_SESS ¬= ACTIVE THEN                          /* CHAPTER 13     */
    DO;
    . DISCARD MU;
    . RETURN;
    END;
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │          ESTABLISH TCCB_PTR          │
                    └──────────────────────────────────────┘
                                                                        */
  IF DISPATCHED_BY(BF.PC*) THEN
    TCCB_PTR = SCB.SEC_TO_BF_TC_CB_PTR;
  ELSE
    TCCB_PTR = SCB.TC_CB_PTR;
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │  PROCESS IPR THAT IS DIRECTED HERE   │
                    └──────────────────────────────────────┘
                                                                        */
  IF TCCB.RCV_PACING = YES & IPR_CHECK = YES THEN      /* PAGE 4-58      */
    DO;
    . CALL FSM_PAC_RQ_SEND;                            /* PAGE 4-60      */
    . DISCARD MU;
    . RETURN;
    END;
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │ IF SUPPORTING A TYPE 1 NODE, PROCESS SEQUENCE │
                    │              NUMBERS                 │
                    └──────────────────────────────────────┘
                                                                        */
  IF SCB.SUPPORTED_NODE_TYPE = T1 THEN
    DO;
    . IF DISPATCHED_BY(BF.PC*) THEN
    .   CALL BF.TC.ADD_SNF;                            /* PAGE 4-55      */
    . ELSE
    .   CALL BF.TC.SAVE_SNF;                           /* PAGE 4-56      */
    END;
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │           PROCESS CLEAR              │
                    └──────────────────────────────────────┘
                                                                        */
  IF RU_CTGY = SC & RQ_CODE = CLEAR &
       SCB.SC_CLEAR = ALLOWED &
       ((RRI = RSP & DISPATCHED_BY(BF.PC*)) |
       (RRI = RQ & DISPATCHED_BY(PC.VRC*))) THEN
    CALL BF.TC.RESET;                                  /* PAGE 4-52      */

  SEND MU TO #SVC_MGR;                                 /* SEE NOTE       */

  RETURN;

END BF.TC.RCV;
```

```
BF.TC.SEND: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  ENFORCES PACING PROTOCOLS IF APPLICABLE.                 │
│                                                                          │
│      INPUT:     RQ|RSP FROM BF.PU_OR_LU.SVC_MGR TCCB_PTR IS ESTABLISHED. │
│                                                                          │
│      OUTPUT:    RQ|RSP TO PC OR Q_PAC                                    │
│                                                                          │
│      NOTES:     1. SEGMENTING IS ONLY VALID ON FLOWS FROM THE SECONDARY TO THE
│                    PRIMARY.  ON FLOWS FROM THE PRIMARY TO SECONDARY BBIUI WILL
│                    ALWAYS BE SET TO BBIU.                                │
│                                                                          │
│                 2. #PC IS SET IN BF.TC.RCV TO EITHER BF.PC.SEND OR PC.VRC.SEND.
│                    WHEN THE FLOW IS FROM PRIMARY TO SECONDARY, IT IS SET TO
│                    BF.PC.SEND; WHEN THE FLOW IS FROM SECONDARY TO PRIMARY, IT IS SET
│                    TO PC.VRC.SEND.  #PC IS CARRIED IN THE TCCB WHICH IS CARRIED
│                    THROUGHOUT THE THREAD AND THEREFORE PROPERLY ESTABLISHED WHEN
│                    THIS PROCEDURE EXECUTES.                              │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 CREATE_IPR                      PAGE 4-58                │
│                 FSM_PAC_RQ_RCV                  PAGE 4-61                │
│                 FSM_PAC_RQ_SEND                 PAGE 4-60                │
│                 UPM_RESOURCES                   PAGE 4-59                │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
SELECT INORDER;
.
. WHEN(BBIUI =  ¬BBIU | (RRI = RQ & EFI = NORMAL))          /* SEE NOTE 1   */
.   DO;
                                                                        /*
                ┌──────────────────────────────────────────┐
                │ IF THE WINDOW SIZE IS SPECIFIED TO BE 0 WHEN │
                │ THE TS PROFILE INDICATES THAT PACING MAY BE │
                │ USED, AND A REQUEST IS RECEIVED WITH PI=PAC, │
                │ THEN THE RECEIVER RETURNS EITHER A PACING │
                │ RESPONSE OR A NEGATIVE RESPONSE WITH SENSE │
                │ CODE FOR PACING NOT SUPPORTED. THIS IS THE │
                │ OPTIONAL CHECK FOR RETURNING A NEGATIVE │
                │ RESPONSE.                                │
                └──────────────────────────────────────────┘
                                                                        */
.   . IF PI = PAC & TCCB.RCV_PACING = NO &
.   .      SCB.TS_PROFILE = (PROFILE_2 | PROFILE_3 | PROFILE_4 | PROFILE_7) THEN
.   .   DO;
.   .   . IF ¬RQN THEN                                      /* APPENDIX B   */
.   .   .   DO;
.   .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'4008');          /* APPENDIX B   */
.   .   .   . SEND MU TO #PC;                               /* CHAPTER 3    */
.   .   .   END;
.   .   . ELSE
.   .   .   DISCARD MU;
.   .   END;
.   .
.   . ELSE
.   .   DO;
.   .   . IF TCCB.SEND_PACING = YES THEN
.   .   .   DO;
.   .   .   . CALL FSM_PAC_RQ_SEND;                         /* PAGE 4-60    */
.   .   .   . INSERT MU LAST IN TCCB.Q_PAC;
.   .   .   END;
.   .   . ELSE
.   .   .   SEND MU TO #PC;                                 /* CHAPTER 3    */
.   .   END;
.   END;
.
. WHEN(EFI = EXPEDITED)
.   SEND MU TO TCCB.#PC;                                    /* CHAPTER 3    */
```

```
. WHEN(RRI = RSP & EFI = NORMAL)
.   DO;
.   . IF TCCB.RCV_PACING = YES &
.   .      UPM_RESOURCES = OK THEN                          /* PAGE 4-59          */
.   .   CALL FSM_PAC_RQ_RCV;                                /* PAGE 4-61          */
                                                                                    /*
                  ┌──────────────────────────────────────────────┐
                  │ WITHIN THIS SECTION THE PACING INDICATOR     │
                  │ COULD BE SET FOR EITHER ONE-STAGE OR         │
                  │ TWO-STAGE PACING.                            │
                  └──────────────────────────────────────────────┘
                                                                                    */
.   . IF TCCB.SEND_PACING = YES THEN
.   .   SELECT ANYORDER;
.   .   .
.   .   . WHEN(QRI = QR)
.   .   .   SEND MU TO #PC;                                 /* CHAPTER 3          */
.   .   .
.   .   . WHEN(QRI = ¬QR & PI = ¬PAC)
.   .   .   INSERT MU LAST IN TCCB.Q_PAC;
.   .   .
.   .   . WHEN(QRI = ¬QR & PI = PAC)
.   .   .   DO;
.   .   .   .
.   .   .   . PI = PAC;
.   .   .   . INSERT MU LAST IN TCCB.Q_PAC;
.   .   .   .
.   .   .   . CALL CREATE_IPR;                             /* PAGE 4-58          */
.   .   .   . EFI = EXPEDITED;
.   .   .   . SEND MU TO #PC;                               /* CHAPTER 3          */
.   .   .   .
.   .   .   END;
.   .   .
.   .   END;
.   .
.   . ELSE
.   .   SEND MU TO #PC;                                     /* CHAPTER 3          */
.   .
.   END;
  END;

  RETURN;

END BF.TC.SEND;
```

```
BF.TC.ADD_SNF: PROCEDURE;
                                                                                    /*
  ┌──────────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  ADDS APPROPRIATE SNF TO AN RU COMING FROM A TYPE 1 NODE          │
  │                                                                                  │
  │      INPUT:     RQ|RSP FROM BF.TC.RCV                                            │
  │                                                                                  │
  │      OUTPUT:    RQ|RSP UPDATED WITH SNF                                          │
  │                                                                                  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                   │
  │                 BF.TC.RCV                          PAGE 4-53                     │
  │                                                                                  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                       │
  │                 UPM_ID_ASSIGN                      PAGE 4-56                     │
  │                 UPM_ID_NORM                        PAGE 4-56                     │
  └──────────────────────────────────────────────────────────────────────────────────┘
                                                                                    */
  SELECT ANYORDER;
  .
  . WHEN(EFI = EXPEDITED & RRI = RQ)
  .   SNF = UPM_ID_ASSIGN;                                  /* PAGE 4-56          */
  .
  . WHEN(EFI = EXPEDITED & RRI = RSP)
  .   SNF = SCB.SEND_EXP_SNF;
  .
  . WHEN(EFI = NORMAL & RRI = RQ)
  .   DO;
  .   . SNF = SCB.SQN_RCV_CNT;
  .   . IF EBIUI = EBIU THEN                                /* TEST ONLY REQUIRED IF */
  .   .                                                     /* UNASSEMBLED SEGMENTS  */
  .   .                                                     /* PASSED BY THE BF      */
  .   .   IF SCB.SQN_USAGE = SEQUENCE_NUMBERS THEN
  .   .     SCB.SQN_RCV_CNT = SCB.SQN_RCV_CNT + 1;
  .   .   ELSE
  .   .     SCB.SQN_RCV_CNT = UPM_ID_NORM;                  /* PAGE 4-56          */
  .   END;
  .
  . WHEN(EFI = NORMAL & RRI = RSP)
  .   SNF = SCB.SEND_NORM_SNF;
  .
  END;

  RETURN;

END BF.TC.ADD_SNF;
```

```
BF.TC.SAVE_SNF: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  SAVES SQN OR ID OF LAST REQUEST GOING TO A TYPE 1 NODE   │
   │                                                                          │
   │      INPUT:     RQ|RSP FROM BF.TC.RCV                                    │
   │                                                                          │
   │      OUTPUT:    RQ, RSP OR EXR AND UPDATED SCB CONTAINING THE SNF        │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 BF.TC.RCV                          PAGE 4-53             │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SELECT ANYORDER;
  .
  . WHEN(RRI = RQ & EFI = EXPEDITED)
  .   SCB.RCV_EXP_SNF = SNF;
  .
  . WHEN(RRI = RQ & EFI = NORMAL)
  .   IF SNF ¬= (SCB.SQN_RCV_CNT + 1) THEN
  .     CALL CHANGE_MU_TO_EXR(X'2001');                    /* APPENDIX B      */
  .                                                        /* SEQUENCE NUMBER */
  .   ELSE
  .     SCB.SQN_RCV_CNT = SCB.SQN_RCV_CNT + 1;
  .
  . OTHERWISE                                              /* RRI = RSP       */
  .   ;
  .
  END;

  RETURN;

END BF.TC.SAVE_SNF;




UPM_ID_NORM: PROCEDURE RETURNS(FIXED BIN(16));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  GENERATES A UNIQUE 16-BIT ID FOR THE SESSION             │
   │                                                                          │
   │      INPUT:     NONE                                                     │
   │                                                                          │
   │      OUTPUT:    16-BIT ID                                                │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 BF.TC.ADD_SNF                      PAGE 4-55             │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SCB.SQN_RCV_CNT = SCB.SQN_RCV_CNT + 1;   /* IMPLEMENTATIONS MAY ASSIGN ANY UNIQUE VALUE    */
  RETURN(SCB.SQN_RCV_CNT);

END UPM_ID_NORM;




UPM_ID_ASSIGN: PROCEDURE RETURNS(FIXED BIN(16));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  GENERATES A UNIQUE 16-BIT ID FOR THE SESSION             │
   │                                                                          │
   │      INPUT:     NONE                                                     │
   │                                                                          │
   │      OUTPUT:    16-BIT ID                                                │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 BF.TC.ADD_SNF                      PAGE 4-55             │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SCB.SEND_EXP_SNF = SCB.SEND_EXP_SNF + 1; /* IMPLEMENTATIONS MAY ASSIGN ANY UNIQUE VALUE    */

  RETURN(SCB.SEND_EXP_SNF);

END UPM_ID_ASSIGN;
```

```
DECODED: PROCEDURE(SIZE) RETURNS(BIT(32));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  CONVERTS MAX_RU_SIZE FROM ITS ENCODED FORM TO AN INTEGER VALUE │
  │                                                                              │
  │      INPUT:     ENCODED VALUE X'AB'                                          │
  │                                                                              │
  │      OUTPUT:    A*(2**B)                                                     │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 SESSACT.PRIMARY_INITIALIZE          PAGE 4-25                │
  │                 SESSACT.SECONDARY_INITIALIZE        PAGE 4-26                │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL SIZE BIT(8);

  DCL EXPONENT FIXED BIN(31);
  DCL EXPONENT_BITS BIT(32) BASED(ADDR(EXPONENT));

  DCL INTSIZE BIT(32);
                                                                                /*
                    ┌──────────────────────────────────┐
                    │      CONVERT EXPONENT INTO INTEGER      │
                    └──────────────────────────────────┘
                                                                                */
  EXPONENT = 0;
  EXPONENT_BITS(28:31) = SIZE(4:7);
                                                                                /*
                    ┌──────────────────────────────────┐
                    │    PLACE MANTISSA IN CORRECT LOCATION   │
                    └──────────────────────────────────┘
                                                                                */
  INTSIZE = ALL_ZEROES;
  INTSIZE(28 - EXPONENT:31 - EXPONENT) = SIZE(0:3);

  RETURN(INTSIZE);

END DECODED;
```

```
CREATE_IPR: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  GENERATES AN ISOLATED PACING RESPONSE (IPR) WITH RH=X'830100'
   │
   │      INPUT:     NONE
   │
   │      OUTPUT:    NORMAL-FLOW IPR
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 BF.TC.SEND                          PAGE 4-54
   │                 TC_OR_BF_TC.IPR_SEND                PAGE 4-29
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  CREATE MU;
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │          SET RH VALUES FOR RESPONSE  │
                    └──────────────────────────────────────┘
                                                                        */
  RRI = RSP;
  BCI = BC;
  ECI = EC;
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │            SET RH VALUES FOR IPR     │
                    └──────────────────────────────────────┘
                                                                        */
  RU_CTGY =  FMD;
  FI      =  B'0';
  SDI     = ¬SD;
  DR1I    = ¬DR1;
  DR2I    = ¬DR2;
  RTI     =  POSITIVE;
  QRI     = ¬QR;
  PI      =  PAC;
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │  SET TH VALUES NEEDED FOR LENGTH AND BOUNDARY
                    │                FUNCTION              │
                    └──────────────────────────────────────┘
                                                                        */
  BBIUI = BBIU;
  EBIUI = EBIU;
  DCF = RH_LENGTH;
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │      DIRECTION BIT FOR META-IMPLEMENTATION
                    └──────────────────────────────────────┘
                                                                        */
  MUCB.DIRECTION = SEND;

  RETURN;

END CREATE_IPR;




IPR_CHECK: PROCEDURE RETURNS(BIT(1));

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  DETERMINES IF MESSAGE UNIT IS AN ISOLATED PACING RESPONSE
   │
   │      INPUT:     RQ|RSP
   │
   │      OUTPUT:    RQ|RSP AND IPR INDICATION
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 BF.TC.RCV                           PAGE 4-53
   │                 PAC_RSP_RCV                         PAGE 4-41
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF RRI = RSP & PI = PAC & DR1I =  ¬DR1 & DR2I =  ¬DR2 THEN
    RETURN(YES);
  ELSE
    RETURN(NO);

END IPR_CHECK;
```

```
UPM_RESOURCES: PROCEDURE RETURNS(BIT(1));
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  DETERMINES  WHETHER THERE  ARE  ENOUGH RESOURCES  TO  SEND A  PACING  │
   │                RESPONSE                                              │
   │                                                                      │
   │     INPUT:     NONE                                                  │
   │                                                                      │
   │     OUTPUT:    OK, IF OK TO SEND A PACING RESPONSE.  OTHERWISE, NG    │
   │                                                                      │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
   │                BF.TC.SEND                          PAGE 4-54         │
   │                TC.CPMGR.SEND_NORM_RSP              PAGE 4-33         │
   │                TC_OR_BF_TC.DEQUEUE.Q_PAC           PAGE 4-29         │
   │                TC_OR_BF_TC.IPR_SEND                PAGE 4-29         │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  RETURN(OK);

END UPM_RESOURCES;
```

```
FSM_PAC_RQ_SEND: FSM_DEFINITION CONTEXT(TCCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     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_PAC_RSP INDICATES THAT  A PACING REQUEST HAS  BEEN SENT BUT  │
│                NO PACING RESPONSE HAS BEEN RECEIVED.                     │
│                                                                          │
│                THIS FSM ALSO APPEARS IN BF.TC.                           │
│                                                                          │
│     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_PAC_RSP  │
│                STATE.  WHEN THE  FSM ENTERS THE AWAITING_PAC_RSP  STATE, THE PACING  │
│                COUNT IS SET TO ONE LESS THAN  THE WINDOW SIZE.  THE PACING COUNT IS  │
│                ONLY INCREASED WHEN A PACING RESPONSE IS RECEIVED, AT WHICH TIME THE  │
│                FSM RETURNS TO THE RESET STATE.                           │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                BF.TC.RCV                          PAGE 4-53              │
│                BF.TC.RESET                        PAGE 4-52              │
│                BF.TC.SEND                         PAGE 4-54              │
│                CPMGR_RESET                        PAGE 4-28              │
│                PAC_RSP_RCV                        PAGE 4-41              │
│                TC_OR_BF_TC.DEQUEUE.Q_PAC          PAGE 4-29              │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----------> / INPUTS	RESET 1	AWAITING PAC_RSP 2
S, RQ, FIRST_IN_WINDOW	2(PACRQ)	/NOTE
S, RQ, ¬FIRST_IN_WINDOW	-(NOPAC)	-(NOPAC)
R, RSP, PAC	>(PACERR)	1(PACRSP)
'RESET'	-	1

OUTPUT CODE	FUNCTION
PACRQ	PI = PAC;
NOPAC	PI = ¬PAC;
PACERR	PI = ¬PAC; CALL UPM_LOG ('UNEXPECTED PACING RSP RECEIVED'); /* APPENDIX B */
PACRSP	PI = ¬PAC; TCCB.PACING_COUNT = TCCB.PACING_COUNT + TCCB.WINDOW_SIZE;

```
END FSM_PAC_RQ_SEND;
```

```
FSM_PAC_RQ_RCV: FSM_DEFINITION CONTEXT(TCCB);
```

```
/*
     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.

                THIS FSM ALSO APPEARS IN BF.TC.

     REFERENCED BY THE FOLLOWING PROCEDURE(S):
                BF.TC.RESET                          PAGE 4-52
                BF.TC.SEND                           PAGE 4-54
                CPMGR_RESET                          PAGE 4-28
                TC.CPMGR.RCV.NORM_RQ                 PAGE 4-40
                TC.CPMGR.SEND_NORM_RSP               PAGE 4-33
                TC_OR_BF_TC.DEQUEUE.Q_PAC            PAGE 4-29
                TC_OR_BF_TC.IPR_SEND                 PAGE 4-29
*/
```

INPUTS \ STATE NAMES ---->	RESET 1	PEND 2
R, RQ, PAC	2	>(PACERR)
R, RQ, ¬PAC	-	-
S, RSP	-(NOPAC)	1(PAC)
'RESET'	-	1

OUTPUT CODE	FUNCTION
PAC	PI = PAC;
NOPAC	PI = ¬PAC;
PACERR	PI = ¬PAC; CALL UPM_LOG ('UNEXPECTED PACING RQ RECEIVED'); /* APPENDIX B */

```
END FSM_PAC_RQ_RCV;
```

```
FSM_CNTL_IMMED_EXP: FSM_DEFINITION CONTEXT(SCB);
```

```
/*
     FUNCTION:  ENFORCES IMMEDIATE  REQUEST MODE FOR EXPEDITED  REQUESTS.  IMMEDIATE
                REQUEST MODE  IS IN  EFFECT FOR  ALL DFC  AND SC  EXPEDITED REQUESTS
                EXCEPT  RQR AND  CLEAR, WHICH  CAN BE  SENT WITHOUT  WAITING FOR  AN
                OUTSTANDING RESPONSE.

                ALL DFC AND SC EXPEDITED REQUESTS ARE SENT RQD.

                IN  RESET STATE,  ANY REQUEST  CAN BE  SENT.  IN  BLOCK_RQ STATE,  A
                RESPONSE NEEDS  TO BE  RECEIVED BEFORE  A REQUEST  OBEYING IMMEDIATE
                REQUEST MODE CAN BE SENT.

     REFERENCED BY THE FOLLOWING PROCEDURE(S):
                CPMGR_RESET                          PAGE 4-28
                TC.CPMGR.RCV                         PAGE 4-36
                TC.CPMGR.RCV_CHECKS                  PAGE 4-38
                TC.CPMGR.SEND                        PAGE 4-31
                TC.CPMGR.SEND_CHECKS                 PAGE 4-32
                TC.SC.SEND_CHECKS                    PAGE 4-48
*/
```

INPUTS \ STATE NAMES ---->	RESET 1	BLOCK_RQ 2
S, RQ, EXP, ¬(CLEAR|RQR)	2	>(S200A)
R, RSP, EXP, ¬(CLEAR|RQR)	>(DISC)	1
'RESET'	-	1

OUTPUT CODE	FUNCTION
DISC	RECEIVE_CHECK = DISCARD_MU;
S200A	MUCB.SEND_CHECK_SENSE = X'200A'; /* IMMEDIATE REQUEST MODE ERROR */

```
END FSM_CNTL_IMMED_EXP;
```

```
FSM_DT_SEND_SDT_AND_CLEAR: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  RECORDS THE ABILITY FOR DATA TO FLOW  IN A SESSION.  THIS VERSION OF
│                THE DATA TRAFFIC FSM HANDLES SESSIONS THAT ALLOW SDT AND CLEAR TO BE
│                SENT.   THIS FSM  APPEARS  ONLY IN  PRIMARY  HALF-SESSIONS USING  TS
│                PROFILES 3 AND 4.
│
│                RESET  MEANS THAT  NO  DFC OR  FMD  TRAFFIC  CAN FLOW.   PEND_ACTIVE
│                INDICATES THAT AN SDT IS OUTSTANDING.  ACTIVE MEANS THAT ALL TRAFFIC
│                CAN FLOW, AND PEND_RESET MEANS THAT A CLEAR IS OUTSTANDING.
│
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                SESSACT.PRIMARY_INITIALIZE          PAGE 4-25
│
│     REFERS TO THE FOLLOWING PROCEDURE(S):
│                CLEAR_RESET                         PAGE 4-27
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES---------> / INPUTS	RESET 1	PEND ACTIVE 2	ACTIVE 3	PEND RESET 4
S, RQ, SDT	2	>(S0809)	>(S0809)	>(S0809)
R, +RSP, SDT	>(DISC)	3	>(DISC)	>(DISC)
R, -RSP, SDT, 2007	>(DISC)	3	>(DISC)	>(DISC)
R, -RSP, SDT, ¬2007	>(DISC)	1	>(DISC)	>(DISC)
S, RQ, CLEAR	4(RESETC)	4(RESETC)	4(RESETC)	-(RESETC)
R, RSP, CLEAR, LAST_CLEAR	>(DISC)	>(DISC)	>(DISC)	1
R, RSP, CLEAR, ¬LAST_CLEAR	>(DISC)	>(DISC)	>(DISC)	-
S, RQ, STSN	-	>(S2007)	>(S2007)	>(S2007)
S, RQ, ¬(SDT|CLEAR|STSN|CRV)	>(S2005)	>(S2005)	-	>(S2005)
S, RSP	>(S2005)	>(S2005)	-	>(S2005)
R, RQ, DFC|FMD	>(R2005)	>(R2005)	-	-(DISC)
R, RSP, DFC|FMD	>(DISC)	>(DISC)	-	-(DISC)
'RESET'	-	1	1	1

OUTPUT CODE	FUNCTION
S0809	MUCB.SEND_CHECK_SENSE = X'0809'; /* MODE INCONSISTENCY */
S2005	MUCB.SEND_CHECK_SENSE = X'2005'; /* DATA TRAFFIC RESET */
S2007	MUCB.SEND_CHECK_SENSE = X'2007'; /* DATA TRAFFIC NOT RESET */

R2005	/* OPTIONAL CHECK */
	IF ¬RQN THEN /* APPENDIX B */
	DO;
	CALL CHANGE_MU_TO_NEG_RSP(X'2005'); /* APPENDIX B */
	/* DATA TRAFFIC RESET */
	RECEIVE_CHECK = NEG_RSP;
	END;
	ELSE
	RECEIVE_CHECK = DISCARD_MU;

DISC	RECEIVE_CHECK = DISCARD_MU;
RESETC	CALL CLEAR_RESET; /* PAGE 4-27 */

```
END FSM_DT_SEND_SDT_AND_CLEAR;
```

```
FSM_DT_RCV_SDT_AND_CLEAR: FSM_DEFINITION CONTEXT(SCB);
                                                                        /*
   FUNCTION:  RECORDS THE ABILITY FOR DATA TO FLOW  IN A SESSION.  THIS VERSION OF
              THE DATA TRAFFIC FSM HANDLES SESSIONS THAT ALLOW SDT AND CLEAR TO BE
              SENT.  THIS  FSM APPEARS  ONLY IN  SECONDARY HALF-SESSIONS  USING TS
              PROFILES 3 AND 4.

              RESET  MEANS THAT  NO  DFC OR  FMD  TRAFFIC  CAN FLOW.   PEND_ACTIVE
              INDICATES THAT  AN SDT IS BEING  PROCESSED BY THE  SERVICES MANAGER.
              ACTIVE MEANS THAT ALL TRAFFIC CAN  FLOW, AND PEND_RESET MEANS THAT A
              CLEAR IS BEING PROCESSED BY THE SERVICES MANAGER.

   NOTE:      WHEN A DUPLICATE SDT IS SENT,  THE SERVICES MANAGER MAY RESPOND WITH
              EITHER A +RSP(SDT) OR -RSP(SDT,2007).

   REFERENCED BY THE FOLLOWING PROCEDURE(S):
              SESSACT.SECONDARY_INITIALIZE        PAGE 4-26

   REFERS TO THE FOLLOWING PROCEDURE(S):
              CLEAR_RESET                         PAGE 4-27
                                                                        */
```

STATE NAMES----------> / INPUTS	RESET 1	PEND ACTIVE 2	ACTIVE 3	PEND RESET 4
R, RQ, SDT	2	>(R0809)	-	>(R0809)
S, +RSP, SDT	>	3	-	>
S, -RSP, SDT, 2007	>	>(S2009)	-	>
S, -RSP, SDT, ¬2007	>	1	>(S2009)	>
R, RQ, CLEAR	4(SETCL)	4(SETCL)	4(SETCL)	-(SETCL)
S, RSP, CLEAR	>(S2009)	>(S2009)	-	1(LASTCL)
S, RQ, ¬RQR	>(S2005)	>(S2005)	-	>(S2005)
S, RSP, ¬(SDT|CLEAR|CRV)	>(S2005)	>(S2005)	-	>(S2005)
R, RQ, DFC|FMD	>(R2005)	>(R2005)	-	-(DISC)
R, RSP, DFC|FMD	>(RSPERR)	>(RSPERR)	-	-(DISC)
R, RQ, STSN	-	>(R2007)	>(R2007)	>(R2007)
'RESET'	-	1	1	1

OUTPUT CODE	FUNCTION
LASTCL	SNF = SCB.LAST_CLEAR_SNF; CALL CLEAR_RESET; /* PAGE 4-27 */
SETCL	SCB.LAST_CLEAR_SNF = SNF;
R0809	CALL CHANGE_MU_TO_NEG_RSP(X'0809'); /* APPENDIX B */ /* MODE INCONSISTENCY */ RECEIVE_CHECK = NEG_RSP;

R2005	IF ¬RQR THEN /* APPENDIX B */
	DO;
	CALL CHANGE_MU_TO_NEG_RSP(X'2005'); /* APPENDIX B */
	/* DATA TRAFFIC RESET */
	RECEIVE_CHECK = NEG_RSP;
	END;
	ELSE
	RECEIVE_CHECK = DISCARD_MU;

R2007	/* OPTIONAL CHECK */ CALL CHANGE_MU_TO_NEG_RSP(X'2007'); /* APPENDIX B */ /* DATA TRAFFIC NOT RESET */ RECEIVE_CHECK = NEG_RSP;
S2009	MUCB.SEND_CHECK_SENSE = X'2009'; /* SESSION CONTROL PROTOCOL VIOLATION */
S2005	MUCB.SEND_CHECK_SENSE = X'2005'; /* DATA TRAFFIC RESET */
DISC	RECEIVE_CHECK = DISCARD_MU;
RSPERR	CALL UPM_LOG ('UNEXPECTED RSP RECEIVED'); /* APPENDIX B */ RECEIVE_CHECK = DISCARD_MU;

```
END FSM_DT_RCV_SDT_AND_CLEAR;
```

```
FSM_DT_SEND_SDT: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  RECORDS THE ABILITY FOR DATA TO FLOW  IN A SESSION.  THIS VERSION OF
│                THE DATA  TRAFFIC FSM HANDLES  SESSIONS THAT  ALLOW SDT TO  BE SENT.
│                THIS FSM APPEARS  ONLY IN PRIMARY HALF-SESSIONS USING  TS PROFILES 5
│                AND 17.
│
│                RESET MEANS THAT NO DFC OR  FMD TRAFFIC CAN FLOW.  PEND_ACTIVE STATE
│                IS ENTERED  WHEN AN SDT REQUEST  IS OUTSTANDING.  ACTIVE  MEANS THAT
│                ALL TRAFFIC CAN FLOW.
│
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                SESSACT.PRIMARY_INITIALIZE          PAGE 4-25
│
│     REFERS TO THE FOLLOWING PROCEDURE(S):
│                CLEAR_RESET                         PAGE 4-27
└──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----------> / INPUTS	RESET 1	PEND ACTIVE 2	ACTIVE 3
S, RQ, SDT	2	>(S0809)	>(S0809)
R, +RSP, SDT	>(DISC)	3(RESETC)	>(DISC)
R, -RSP, SDT, 2007	>(DISC)	3(RESETC)	>(DISC)
R, -RSP, SDT, ¬2007	>(DISC)	1	>(DISC)
S, RQ, ¬SDT	>(S2005)	>(S2005)	-
S, RSP	>(S2005)	>(S2005)	-
R, RQ, DFC|FMD	>(R2005)	>(R2005)	-
R, RSP, DFC|FMD	>(DISC)	>(DISC)	-
'RESET'	-	1	1

OUTPUT CODE	FUNCTION
S0809	MUCB.SEND_CHECK_SENSE = X'0809'; /* MODE INCONSISTENCY */
S2005	MUCB.SEND_CHECK_SENSE = X'2005'; /* DATA TRAFFIC RESET */

R2005	/* OPTIONAL CHECK */
	IF ¬RQN THEN /* APPENDIX B */
	DO;
	CALL CHANGE_MU_TO_NEG_RSP(X'2005'); /* APPENDIX B */
	/* DATA TRAFFIC RESET */
	RECEIVE_CHECK = NEG_RSP;
	END;
	ELSE
	RECEIVE_CHECK = DISCARD_MU;

DISC	RECEIVE_CHECK = DISCARD_MU;
RESETC	CALL CLEAR_RESET; /* PAGE 4-27 */

```
END FSM_DT_SEND_SDT;
```

```
FSM_DT_RCV_SDT: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  RECORDS THE ABILITY FOR DATA TO FLOW  IN A SESSION.  THIS VERSION OF
│                THE DATA  TRAFFIC FSM HANDLES  SESSIONS THAT  ALLOW SDT TO  BE SENT.
│                THIS FSM APPEARS ONLY IN SECONDARY HALF-SESSIONS USING TS PROFILES 5
│                AND 17.
│
│                RESET MEANS THAT NO DFC OR  FMD TRAFFIC CAN FLOW.  PEND_ACTIVE STATE
│                IS ENTERED  WHEN THE SERVICES MANAGER  IS PROCESSING A  SDT.  ACTIVE
│                MEANS THAT ALL TRAFFIC CAN FLOW.
│
│     NOTE:      WHEN A DUPLICATE SDT IS SENT,  THE SERVICES MANAGER MAY RESPOND WITH
│                EITHER A +RSP(SDT)  OR -RSP(SDT,2007).
│
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                SESSACT.SECONDARY_INITIALIZE        PAGE 4-26
└──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----------> / INPUTS	RESET 1	PEND ACTIVE 2	ACTIVE 3
R, RQ, SDT	2	>(R0809)	-
S, RSP, SDT	>(S2009)	3	-
S, RQ, ¬RQR	>(S2005)	>(S2005)	-
S, RSP, ¬SDT	>(S2005)	>(S2005)	-
R, RQ, DFC|FMD	>(R2005)	>(R2005)	-
R, RSP, DFC|FMD	>(RSPERR)	>(RSPERR)	-
'RESET'	-	1	1

OUTPUT CODE	FUNCTION
R0809	CALL CHANGE_MU_TO_NEG_RSP(X'0809'); /* APPENDIX B */ /* MODE INCONSISTENCY */ RECEIVE_CHECK = NEG_RSP;

R2005	IF ¬RQN THEN /* APPENDIX B */
	DO;
	CALL CHANGE_MU_TO_NEG_RSP(X'2005'); /* APPENDIX B */
	/* DATA TRAFFIC RESET */
	RECEIVE_CHECK = NEG_RSP;
	END;
	ELSE
	RECEIVE_CHECK = DISCARD_MU;

S2009	MUCB.SEND_CHECK_SENSE = X'2009'; /* SESSION CONTROL PROTOCOL VIOLATION */
S2005	MUCB.SEND_CHECK_SENSE = X'2005'; /* DATA TRAFFIC RESET */
RSPERR	CALL UPM_LOG ('UNEXPECTED RSP RECEIVED'); /* APPENDIX B */ RECEIVE_CHECK = DISCARD_MU;

```
END FSM_DT_RCV_SDT;
```

```
FSM_DT_SEND_CLEAR: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  RECORDS THE ABILITY FOR DATA TO FLOW  IN A SESSION.  THIS VERSION OF │
│                THE DATA  TRAFFIC FSM HANDLES SESSIONS  THAT ALLOW ONLY CLEAR  TO BE │
│                SENT.   THIS FSM  APPEARS  ONLY IN  PRIMARY  HALF-SESSIONS USING  TS │
│                PROFILE 2.                                                │
│                                                                          │
│                ACTIVE  STATE MEANS  THAT  ALL TRAFFIC  CAN  FLOW.   THE PEND  STATE │
│                INDICATES AN OUTSTANDING CLEAR.                           │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                SESSACT.PRIMARY_INITIALIZE          PAGE 4-25             │
│                                                                          │
│     REFERS TO THE FOLLOWING PROCEDURE(S):                                │
│                CLEAR_RESET                         PAGE 4-27             │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

┌──────────────────────────────┬──────────────┬──────────────┐
│              STATE NAMES---------------->│    ACTIVE    │     PEND     │
│ INPUTS                       │      1       │      2       │
├──────────────────────────────┼──────────────┼──────────────┤
│ S, RQ, CLEAR                 │ 2(RESETC)    │ -(RESETC)    │
│ R, RSP, CLEAR, LAST_CLEAR    │ >(RSPERR)    │ 1            │
│ R, RSP, CLEAR, ¬LAST_CLEAR   │ >(RSPERR)    │ -            │
├──────────────────────────────┼──────────────┼──────────────┤
│ S, RQ, ¬CLEAR                │ -            │ >(S2005)     │
│ S, RSP                       │ -            │ >(S2005)     │
├──────────────────────────────┼──────────────┼──────────────┤
│ R, DFC|FMD                   │ -            │ -(DISC)      │
├──────────────────────────────┼──────────────┼──────────────┤
│ 'RESET'                      │ -            │ 1            │
└──────────────────────────────┴──────────────┴──────────────┘

┌──────────┬──────────────────────────────────────────────────────────┐
│ OUTPUT   │ FUNCTION                                                 │
│  CODE    │                                                          │
├──────────┼──────────────────────────────────────────────────────────┤
│ RESETC   │ CALL CLEAR_RESET;                  /* PAGE 4-27      */ │
├──────────┼──────────────────────────────────────────────────────────┤
│ RSPERR   │ CALL UPM_LOG ('UNEXPECTED RSP RECEIVED'); /* APPENDIX B */ │
│          │ RECEIVE_CHECK = DISCARD_MU;                              │
├──────────┼──────────────────────────────────────────────────────────┤
│ S2005    │ MUCB.SEND_CHECK_SENSE = X'2005';   /* DATA TRAFFIC RESET */ │
├──────────┼──────────────────────────────────────────────────────────┤
│ DISC     │ RECEIVE_CHECK = DISCARD_MU;                              │
└──────────┴──────────────────────────────────────────────────────────┘

END FSM_DT_SEND_CLEAR;
```

```
FSM_DT_RCV_CLEAR: FSM_DEFINITION CONTEXT(SCB);
```

```
/*
┌──────────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  RECORDS THE ABILITY FOR DATA TO FLOW  IN A SESSION.  THIS VERSION OF │
│                THE DATA  TRAFFIC FSM  HANDLES SESSIONS  USING TS  PROFILE 2,  WHICH │
│                ALLOW ONLY  CLEAR TO BE  SENT.  THIS  FSM APPEARS ONLY  IN SECONDARY │
│                HALF-SESSIONS USING TS PROFILE 2.                             │
│                                                                              │
│                ACTIVE  STATE MEANS  THAT DATA  TRAFFIC  CAN FLOW.   THE PEND  STATE │
│                INDICATES THAT THE SERVICES MANAGER IS PROCESSING CLEAR.      │
│                                                                              │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                                │
│                SESSACT.SECONDARY_INITIALIZE       PAGE 4-26                  │
│                                                                              │
│     REFERS TO THE FOLLOWING PROCEDURE(S):                                    │
│                CLEAR_RESET                        PAGE 4-27                  │
└──────────────────────────────────────────────────────────────────────────────┘
*/
```

STATE NAMES----------> / INPUTS	ACTIVE 1	PEND 2
R, RQ, CLEAR S, RSP, CLEAR	2(SETCL) >(S2009)	-(SETCL) 1(LASTCL)
S, RQ, ¬RQR S, RSP, ¬CLEAR	- -	>(S2005) >(S2005)
R, DFC|FMD	-	-(DISC)
'RESET'	-	1

OUTPUT CODE	FUNCTION
LASTCL	SNF = SCB.LAST_CLEAR_SNF; CALL CLEAR_RESET; /* PAGE 4-27 */
SETCL	SCB.LAST_CLEAR_SNF = SNF;
DISC	RECEIVE_CHECK = DISCARD_MU;
S2009	MUCB.SEND_CHECK_SENSE = X'2009'; /* SESSION CONTROL PROTOCOL VIOLATION */
S2005	MUCB.SEND_CHECK_SENSE = X'2005'; /* DATA TRAFFIC RESET */

```
END FSM_DT_RCV_CLEAR;
```

```
FSM_RQR_SEND: FSM_DEFINITION CONTEXT(SCB);
```

```
/*
┌──────────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  RECORDS THE SENDING  OF A REQUEST RECOVERY (RQR).   THIS FSM APPEARS │
│                ONLY IN SECONDARY HALF-SESSIONS THAT SUPPORT RQR.             │
│                                                                              │
│                RESET STATE  MEANS THAT  THERE IS  NO OUTSTANDING  RQR.  PEND  STATE │
│                INDICATES AN OUTSTANDING RQR.                                 │
│                                                                              │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                                │
│                SESSACT.SECONDARY_INITIALIZE       PAGE 4-26                  │
└──────────────────────────────────────────────────────────────────────────────┘
*/
```

STATE NAMES----------> / INPUTS	RESET 1	PEND 2
S, RQ, RQR R, RSP, RQR	2 >(RSPERR)	>(S0809) 1
'RESET'	-	1

OUTPUT CODE	FUNCTION
S0809	MUCB.SEND_CHECK_SENSE = X'0809'; /* MODE INCONSISTENCY */
RSPERR	CALL UPM_LOG ('UNEXPECTED RESPONSE RECEIVED'); /* APPENDIX B */ RECEIVE_CHECK = DISCARD_MU;

```
END FSM_RQR_SEND;
```

```
FSM_RQR_RCV: FSM_DEFINITION CONTEXT(SCB);
```

```
                                                                        /*
┌──────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  RECORDS THE RECEIPT  OF A REQUEST RECOVERY (RQR).     │
│                THIS FSM APPEARS ONLY IN PRIMARY HALF-SESSIONS THAT   │
│                SUPPORT RQR.                                          │
│                                                                      │
│                RESET STATE  MEANS THAT  THERE IS  NO OUTSTANDING     │
│                RQR.  PEND  STATE INDICATES THAT THE SERVICES         │
│                MANAGER IS PROCESSING A RQR.                          │
│                                                                      │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
│                SESSACT.PRIMARY_INITIALIZE        PAGE 4-25          │
└──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----------> / INPUTS	RESET 1	PEND 2
R, RQ, RQR	2	>(R0809)
S, RSP, RQR	>(S0809)	1
'RESET'	-	1

OUTPUT CODE	FUNCTION
R0809	CALL CHANGE_MU_TO_NEG_RSP(X'0809'); /* APPENDIX B */ /* MODE INCONSISTENCY */ RECEIVE_CHECK = NEG_RSP;
S0809	MUCB.SEND_CHECK_SENSE = X'0809'; /* MODE INCONSISTENCY */

```
END FSM_RQR_RCV;
```

```
FSM_STSN_SEND: FSM_DEFINITION CONTEXT(SCB);
```

```
                                                                        /*
┌──────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  RECORDS THE SENDING OF A SET  AND TEST SEQUENCE       │
│                NUMBER (STSN).  THIS FSM APPEARS ONLY IN PRIMARY      │
│                HALF-SESSIONS THAT SUPPORT STSN.                      │
│                                                                      │
│                RESET STATE  MEANS THAT  THERE IS NO  OUTSTANDING     │
│                STSN.  PEND STATE INDICATES AN OUTSTANDING STSN.      │
│                                                                      │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
│                SESSACT.PRIMARY_INITIALIZE        PAGE 4-25          │
└──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----------> / INPUTS	RESET 1	PEND 2
S, RQ, STSN	2(SET)	>(S0809)
R, RSP, STSN	>(DISC)	1
'RESET'	-	1

OUTPUT CODE	FUNCTION
DISC	RECEIVE_CHECK = DISCARD_MU;

SET	IF STSN_RQ.ACTION_CODE_SEC_TO_PRI = (SET | SET_AND_TEST) THEN
	SCB.SQN_RCV_CNT = STSN_RQ.SEC_TO_PRI_SQN;
	IF STSN_RQ.ACTION_CODE_PRI_TO_SEC = (SET | SET_AND_TEST) THEN
	SCB.SQN_SEND_CNT = STSN_RQ.PRI_TO_SEC_SQN;

S0809	MUCB.SEND_CHECK_SENSE = X'0809'; /* MODE INCONSISTENCY */

```
END FSM_STSN_SEND;
```

```
FSM_STSN_RCV: FSM_DEFINITION CONTEXT(SCB);
                                                                        /*
   FUNCTION:  RECORDS THE RECEIPT OF A SET  AND TEST SEQUENCE NUMBER (STSN).  THIS
              FSM APPEARS ONLY IN SECONDARY HALF-SESSIONS THAT SUPPORT STSN.

              RESET STATE  MEANS THAT  THERE IS NO  OUTSTANDING STSN.   PEND STATE
              INDICATES THAT THE SERVICES MANAGER IS PROCESSING A STSN.

   REFERENCED BY THE FOLLOWING PROCEDURE(S):
              SESSACT.SECONDARY_INITIALIZE       PAGE 4-26
                                                                        */
```

STATE NAMES----------> / INPUTS	RESET 1	PEND 2
R, RQ, STSN S, RSP, STSN	2(SET) >(S0809)	>(R0809) 1
'RESET'	-	1

OUTPUT CODE	FUNCTION

SET	IF STSN_RQ.ACTION_CODE_SEC_TO_PRI = (SET | SET_AND_TEST) THEN
	  SCB.SQN_SEND_CNT = STSN_RQ.SEC_TO_PRI_SQN;
	IF STSN_RQ.ACTION_CODE_PRI_TO_SEC = (SET | SET_AND_TEST) THEN
	  SCB.SQN_RCV_CNT = STSN_RQ.PRI_TO_SEC_SQN;

R0809	CALL CHANGE_MU_TO_NEG_RSP(X'0809'); /* APPENDIX B */ /* MODE INCONSISTENCY */ RECEIVE_CHECK = NEG_RSP;
S0809	MUCB.SEND_CHECK_SENSE = X'0809'; /* MODE INCONSISTENCY */

```
END FSM_STSN_RCV;
```

```
FSM_CRV_SEND: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  RECORDS THE ABILITY FOR ENCIPHERED DATA  TO FLOW IN A SESSION.  THIS
   │                FSM APPEARS ONLY IN PRIMARY HALF-SESSIONS THAT SUPPORT CRV.
   │
   │                RESET MEANS THAT  CRV HAS NOT YET BEEN  SENT.  PEND_ACTIVE INDICATES
   │                AN OUTSTANDING CRV.  ACTIVE MEANS THAT CRYPTOGRAPHY IS FUNCTIONAL.
   │
   │     NOTE:      ON RECEIPT  OF A NEGATIVE RESPONSE  TO CRV, THE LU.SVC_MGR  SETS THE
   │                SESSION CRYPTOGRAPHY  KEY AND SESSION  CRYPTOGRAPHY SEED TO  0'S AND
   │                CAUSES AN UNBIND TO BE SENT.
   │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                SESSACT.PRIMARY_INITIALIZE          PAGE 4-25
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

   ┌──────────────────────┬──────────┬──────────┬──────────┐
   │ STATE NAMES--------->│  RESET   │   PEND   │  ACTIVE  │
   │                      │          │  ACTIVE  │          │
   │ INPUTS               │    1     │    2     │    3     │
   ├──────────────────────┼──────────┼──────────┼──────────┤
   │ S, RQ, CRV           │ 2        │ >(S0809) │ >(S0809) │
   │ R, +RSP, CRV         │ >(DISC)  │ 3        │ >(DISC)  │
   │ R, -RSP, CRV         │ >(DISC)  │ 1        │ >(DISC)  │
   ├──────────────────────┼──────────┼──────────┼──────────┤
   │ S, RQ, ¬CRV          │ >(S2009) │ >(S2009) │ -        │
   │ S, RSP               │ >(S2009) │ >(S2009) │ -        │
   ├──────────────────────┼──────────┼──────────┼──────────┤
   │ R, DFC|FMD           │ >(R2009) │ >(R2009) │ -        │
   ├──────────────────────┼──────────┼──────────┼──────────┤
   │ 'RESET'              │ -        │ 1        │ 1        │
   └──────────────────────┴──────────┴──────────┴──────────┘

   ┌──────────┬──────────────────────────────────────────────────────────┐
   │ OUTPUT   │ FUNCTION                                                 │
   │  CODE    │                                                          │
   ├──────────┼──────────────────────────────────────────────────────────┤
   │ S0809    │ MUCB.SEND_CHECK_SENSE = X'0809';                         │
   │          │                                   /* MODE INCONSISTENCY */ │
   ├──────────┼──────────────────────────────────────────────────────────┤
   │ S2009    │ MUCB.SEND_CHECK_SENSE = X'2009';  /* SESSION CONTROL PROTOCOL VIOLATION */ │
   ├──────────┼──────────────────────────────────────────────────────────┤
   │ R2009    │ IF ¬RQN THEN                                 /* APPENDIX B */ │
   │          │   DO;                                                    │
   │          │     CALL CHANGE_MU_TO_NEG_RSP(X'200900C0');  /* APPENDIX B */ │
   │          │                                  /* SESSION CONTROL PROTOCOL VIOLATION */ │
   │          │     RECEIVE_CHECK = NEG_RSP;                             │
   │          │   END;                                                   │
   │          │ ELSE                                                     │
   │          │   RECEIVE_CHECK = DISCARD_MU;                            │
   ├──────────┼──────────────────────────────────────────────────────────┤
   │ DISC     │ RECEIVE_CHECK = DISCARD_MU;                              │
   └──────────┴──────────────────────────────────────────────────────────┘

END FSM_CRV_SEND;
```

```
FSM_CRV_RCV: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │     FUNCTION:  RECORDS THE ABILITY FOR ENIPHERED DATA TO FLOW IN A SESSION.  THIS
 │                FSM APPEARS ONLY IN SECONDARY HALF-SESSIONS THAT SUPPORT CRV.
 │
 │                RESET MEANS THAT CRV HAS NOT YET BEEN RECEIVED.  PEND_ACTIVE
 │                INDICATES THAT THE LU SERVICES MANAGER IS PROCESSING A CRV.  ACTIVE
 │                MEANS THAT CRYPTOGRAPHY IS FUNCTIONAL.
 │
 │     NOTE:      WHEN THE LU SERVICES MANAGER SENDS A NEGATIVE RESPONSE TO CRV, IT
 │                SETS THE SESSION CRYPTOGRAPHY KEY AND SESSION CRYPTOGRAPHY SEED TO
 │                0'S.
 │
 │     REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                SESSACT.SECONDARY_INITIALIZE       PAGE 4-26
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */

┌──────────────────────┬──────────┬──────────┬──────────┐
│ STATE NAMES---------->│  RESET   │   PEND   │  ACTIVE  │
│                      │          │  ACTIVE  │          │
│ INPUTS               │    1     │    2     │    3     │
├──────────────────────┼──────────┼──────────┼──────────┤
│ R, RQ, CRV           │ 2        │ >(R0809) │ >(R0809) │
│ S, +RSP, CRV         │ >(RSPERR)│ 3        │ >(RSPERR)│
│ S, -RSP, CRV         │ >(RSPERR)│ 1        │ >(RSPERR)│
├──────────────────────┼──────────┼──────────┼──────────┤
│ S, RQ                │ >(S2009) │ >(S2009) │ -        │
│ S, RSP, ¬CRV         │ >(S2009) │ >(S2009) │ -        │
├──────────────────────┼──────────┼──────────┼──────────┤
│ R, RQ, SDT           │ >(R2009) │ >(R2009) │ -        │
│ R, DFC|FMD           │ >(R2009) │ >(R2009) │ -        │
├──────────────────────┼──────────┼──────────┼──────────┤
│ 'RESET'              │ -        │ 1        │ 1        │
└──────────────────────┴──────────┴──────────┴──────────┘

┌──────────┬──────────────────────────────────────────────────────────────┐
│ OUTPUT   │ FUNCTION                                                     │
│  CODE    │                                                              │
├──────────┼──────────────────────────────────────────────────────────────┤
│ R0809    │ CALL CHANGE_MU_TO_NEG_RSP(X'0809');          /* APPENDIX B  */
│          │                                              /* MODE INCONSISTENCY */
│          │ RECEIVE_CHECK = NEG_RSP;                                     │
├──────────┼──────────────────────────────────────────────────────────────┤
│ R2009    │ IF ¬RQN THEN                                 /* APPENDIX B  */
│          │   DO;                                                        │
│          │     CALL CHANGE_MU_TO_NEG_RSP(X'200900C0') ; /* APPENDIX B */
│          │                                  /* SESSION CONTROL PROTOCOL VIOLATION */
│          │     RECEIVE_CHECK = NEG_RSP;                                │
│          │   END;                                                       │
│          │ ELSE                                                         │
│          │   RECEIVE_CHECK = DISCARD_MU;                                │
├──────────┼──────────────────────────────────────────────────────────────┤
│ S2009    │ MUCB.SEND_CHECK_SENSE = X'2009';  /* SESSION CONTROL PROTOCOL VIOLATION */
├──────────┼──────────────────────────────────────────────────────────────┤
│ RSPERR   │ MUCB.SEND_CHECK_SENSE = X'4001';             /* INVALID SC RH */
└──────────┴──────────────────────────────────────────────────────────────┘

END FSM_CRV_RCV;
```

```
FSM_INPUT_DEFINITION:
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      THE SYMBOLS  USED IN  THE INPUTS  COLUMN OF  THE STATE-TRANSITION  MATRICES ARE  │
   │      DEFINED BELOW.                                                  │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

   CLEAR                    RU_CTGY = SC & RQ_CODE = CLEAR;
   CLEAR|RQR                RU_CTGY = SC & RQ_CODE = (CLEAR|RQR);
   CRV                      RU_CTGY = SC & RQ_CODE = CRV;
   DFC|FMD                  RU_CTGY = (DFC | FMD);
   EXP                      EFI = EXPEDITED;
   FIRST_IN_WINDOW          TCCB.PACING_COUNT = TCCB.WINDOW_SIZE;
   LAST_CLEAR               SNF = SCB.LAST_CLEAR_SNF;
   PAC                      PI = PAC;
   R                        MUCB.DIRECTION = RECEIVE;
   'RESET'                  FSMINPUT = 'RESET';
   RQ                       RRI = RQ;
   RQR                      RU_CTGY = SC & RQ_CODE = RQR;
   RSP                      RRI = RSP;
   +RSP                     RRI = RSP & RTI = POS;
   -RSP                     RRI = RSP & RTI = NEG;
   S                        MUCB.DIRECTION = SEND;
   SDT                      RU_CTGY = SC & RQ_CODE = SDT;
   SDT|CLEAR|CRV            RU_CTGY = SC & RQ_CODE = (SDT | CLEAR | CRV);
   SDT|CLEAR|STSN|CRV       RU_CTGY = SC & RQ_CODE = (SDT | CLEAR | STSN | CRV);
   STSN                     RU_CTGY = SC & RQ_CODE = STSN;
   2007                     SNC = X'2007';

END FSM_INPUT_DEFINITION;
```

```
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │                                                                      │
   │                                                                      │
   │                          GLOBAL CHAPTER VARIABLES                    │
   │                                                                      │
   │                                                                      │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

```
DCL RECEIVE_CHECK BIT(2);  /* INDICATES CORRECT DISPOSITION OF A RECEIVED MU */
                           /*    B'00' = GOOD,    B'01' = DISCARD_MU,        */
                           /*    B'10' = NEG_RSP, B'11' = CONVERT_TO_EXR     */
```

INTRODUCTION

GENERAL DESCRIPTION

The function of the data flow control (DFC) layer (Figure 5-1) is to control the flow of FMD requests and responses between FMDS pairs within sessions. DFC handles only FMD and DFC requests; network control and session control requests do not flow through DFC.

A distinct DFC element is provided for each half-session (identified uniquely by HSID) supported in the node. The qualifying half-session prefix, HSID, is always implied for the DFC layer. A distinct memory--the session control block (SCB)--exists for each HSID. This memory contains, for DFC, its states, tables, and other local fields. (See Appendix A for the detailed format.)

BRIEF DESCRIPTION OF DFC FUNCTIONS

- Request/Response Formatting: DFC enforces correct RH parameter settings for FMD and DFC requests and responses.

- Chaining Protocol: Chaining is enforced and checked to provide a means of sending or receiving a sequence of requests as an error recovery entity.

- Request/Response Correlation: DFC correlates responses with their associated requests. The sequence number field on requests is also assigned by DFC.

- Request/Response Mode Protocols: Immediate and delayed request/response modes are enforced by DFC.

- Send/Receive Mode Protocols: The normal-flow send/receive modes (full-duplex, half-duplex contention, half-duplex flip-flop) specify a particular form of coordination between sending and receiving of normal-flow requests and responses. DFC checks that this is done correctly.

- Brackets Protocol: Bracket protocols are enforced to provide a means of sending or receiving a sequence of chains as a delimited transaction entity.

- Error Recovery Protocol: When a negative response is sent to a normal-flow request and the session protocol allows more than one chain to be sent before a response is received, the beginning of error recovery is delayed until the extra chains have been completely received.

- Stop-bracket-initiation, Quiesce, and Shutdown Protocols: Normal-flow traffic may be suspended using various DFC requests; DFC enforces suspension rules following quiescing or shutdown of the normal flows.

- Queued Response Protocol: The queueing of responses to normal-flow requests (on the Q_TC_TO_DFC queue) is regulated by DFC.

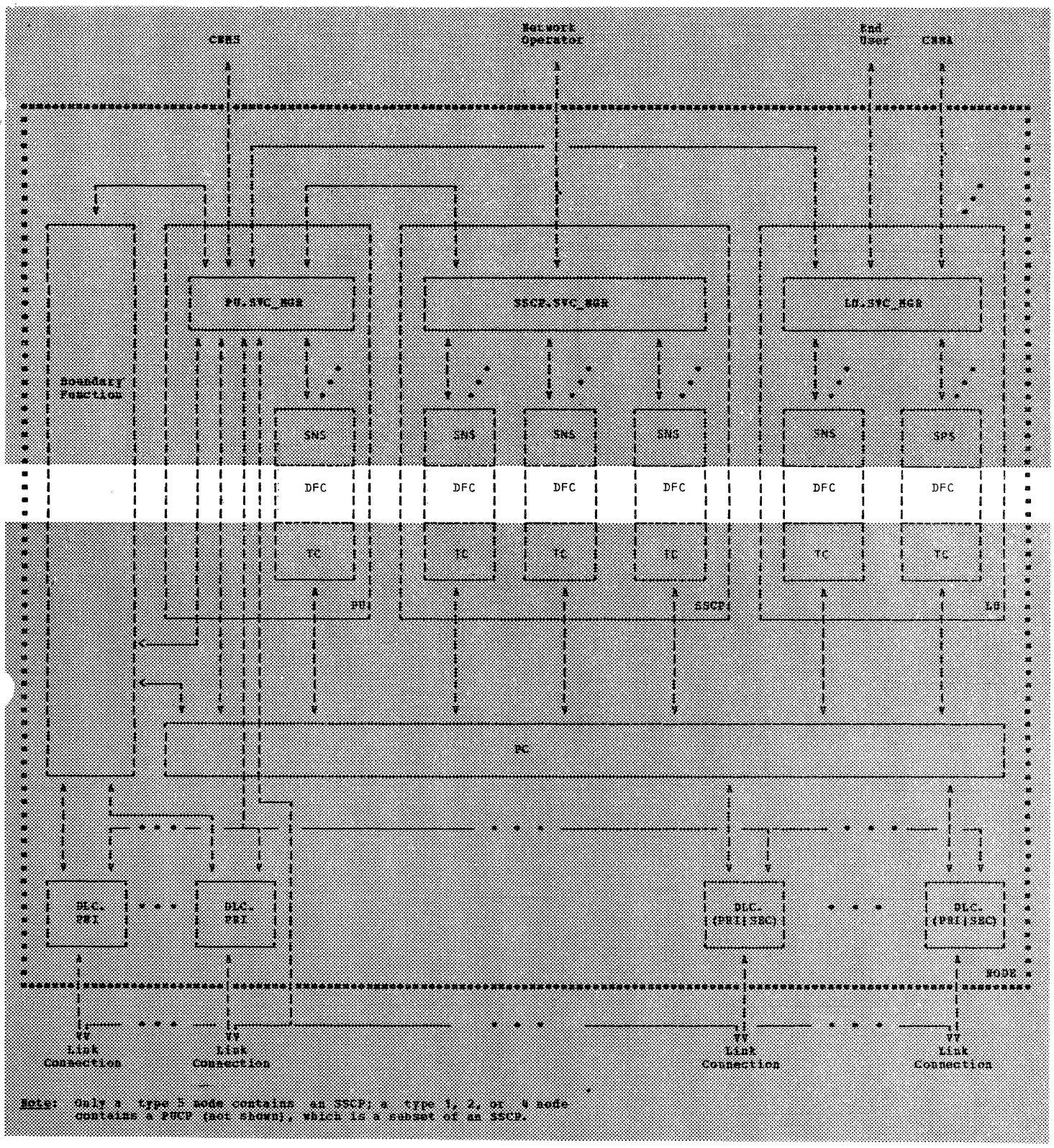


Figure 5-1. Structural Overview of a Node

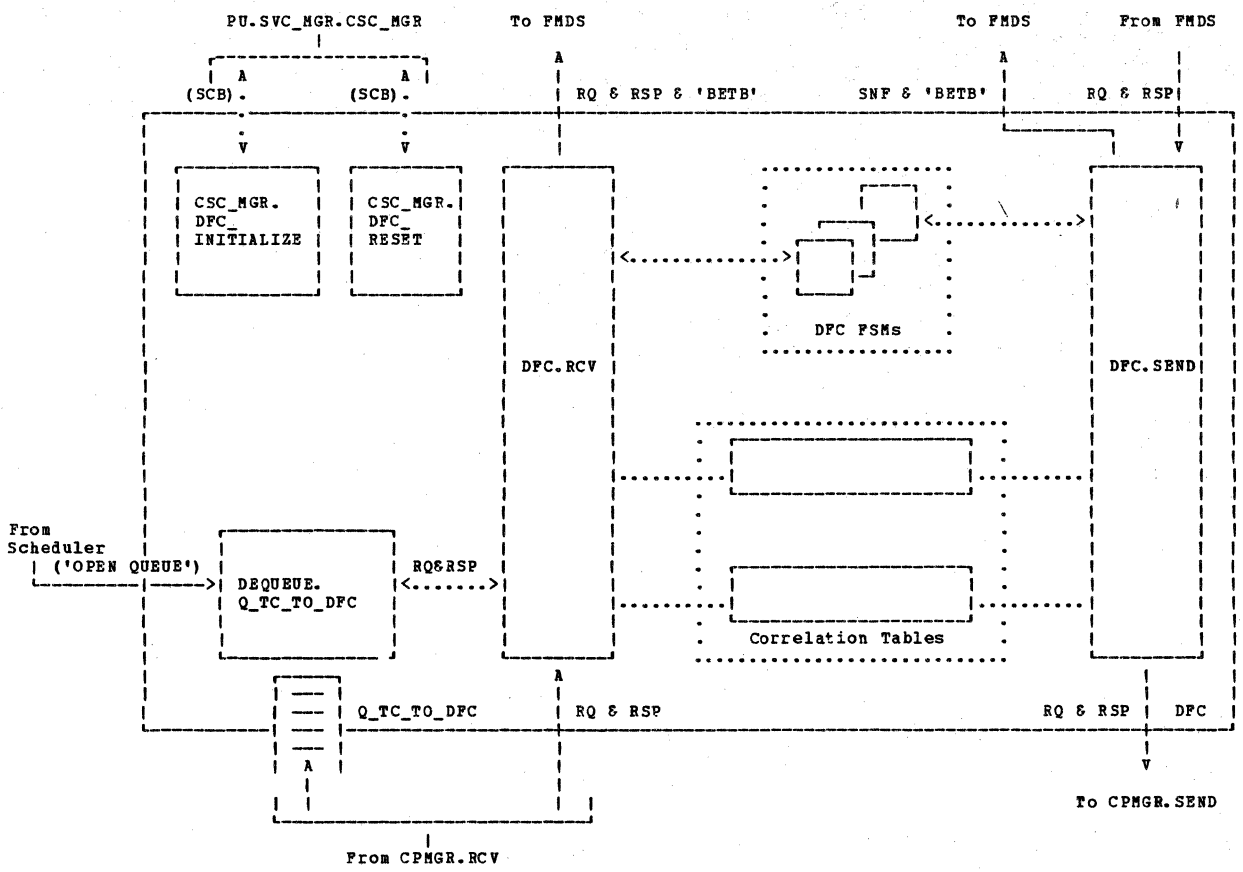


Figure 5-2. Structure of DFC

DFC STRUCTURE

DFC COMPONENTS

Initialization

The DFC initialization component (CSC_MGR.DFC_INITIALIZE, page 5-31) is called by the common session control manager in the PU services component (PU.SVC_MGR.CSC_MGR, Chapter 13) at the activation of each session. It initializes FSMs, DFC request usage, and other protocol related parameters to be used during the session. These are based on the FM profile and its associated parameters used to activate the session.

Reset

The DFC reset component (CSC_MGR.DFC_RESET, page 5-38) is called by the common session control manager in the PU services component (PU.SVC_MGR.CSC_MGR, Chapter 13) at the activation of each session. It is also called as a result of resetting a subtree that includes DFC. Its function is to reset FSMs, correlation tables, and other DFC related fields.

Dequeue

The DFC dequeue component (DEQUEUE.Q_TC_TO_DFC, Page 5-40) is invoked by the higher level scheduler (see Appendix C for description of the scheduler) to dequeue a BIU from the Q_TC_TO_DFC queue. Each half-session has a Q_TC_TO_DFC queue. Valid normal-flow FMD or DFC BIUs received by a half-session are handled first by the half-session's TC element, where they may be temporarily queued in Q_TC_TO_DFC prior to being passed to the DFC.RCV procedure. There are conditions under which the received BIUs are passed immediately to the DFC.RCV procedure. For example, expedited-flow BIUs always bypass Q_TC_TO_DFC and go directly to DFC.RCV. The BIUs in the Q_TC_TO_DFC are dequeued, one at a time, under control of the scheduler. When the scheduler decides to dequeue, it invokes the DEQUEUE.Q_TC_TO_DFC procedure by sending (using the SEND function and the dispatcher) it an OPEN_QUEUE signal. This procedure determines if it is allowable, at the time, to dequeue a BIU from Q_TC_TO_DFC. If a dequeue is allowable, a dequeue is done and the DFC receive procedure (DFC.RCV, page 5-50) is called to process the dequeued BIU. If a dequeue is not allowable, at the time, DEQUEUE.Q_TC_TO_DFC returns control to the dispatcher.

In general, the rules for dequeuing are:

- First speakers are, by definition, contention winners. Contention winners may always dequeue. The node scheduler for half-sessions of this type may cause the Q_TC_TO_DFC queue to become transparent by doing a dequeue immediately after each enqueue.

- Bidders and contention losers may dequeue only when the states of the send/receive mode FSMs or bracket FSMs allow it.

- Responses may always be dequeued.

- Half-sessions using the full-duplex send/receive protocol may always dequeue.

Two conditions require special attention by the scheduler in order to avoid deadlocks:

(1) A half-session's Q_PAC is full and it is waiting for a pacing response.

(2) A half-session has sent an RQD request and is waiting for the response.

If both paired half-sessions were concurrently to have one of the above conditions existing, i.e., both had (1), or both had (2), or one had (1) while the other had (2), the result would be a deadlock--each half-session would be waiting for an event that would never occur.

To avoid these deadlocks, the scheduler forces a dequeue from Q_TC_TO_DFC if either condition (1) or (2) exists for a half-session that is a contention winner or whose session parameters indicate HDX-FF with symmetric error recovery and no brackets.

Send

The DFC send component (DFC.SEND procedure, page 5-41) handles sending requests and responses. It receives a request or response from the layer above it FMDS.SEND (Chapter 6), processes it (if error free), and sends it on to the next lower layer, CPMGR.SEND (Chapter 4). If an error is found in the request or response, a reject is sent to the next higher layer (the one that sent the request or response in error). If no errors are found, DFC send processing consists mainly of updating the states of the DFC FSMs.

Detailed protocol boundary information is specified in the prologue for DFC.SEND (page 5-41).

Receive

The DFC receive component (DFC.RCV procedure, page 5-50) handles receiving requests and responses. It receives a request or response from either the DFC dequeue component (DEQUEUE.Q_TC_TO_DFC procedure, page &CA5P001) or the next lower layer, CPMGR.RCV (Chapter 4). DFC.RCV optionally may check for receive error conditions. These are conditions that occur only when the other half-session has violated the architecture. If a receive error condition is detected the action taken by DFC.RCV is not architected. The suggested courses of action are described in an unarchitected procedure UPM_RECEIVE_ERROR_PROCESS (page 5-57). If no receive error conditions are detected, the processing consists mainly of updating the states of the DFC FSMs. After DFC.RCV finishes processing, the request or response is sent to the next higher layer, FMDS.RCV (Chapter 6).

Detailed protocol boundary information is specified in the prologue for DFC.RCV (page 5-50).

CONTROL BLOCKS

The chief control block used by DFC is the session control block (SCB). Each time a DFC component is entered, an SCB is implicity passed as one of its parameters. DFC uses the SCB for:

- Referencing session activation parameters, such as FM profile and, chaining usage.
- Anchor points for correlation tables.
- Memory for DFC FSM states.
- Setting up fields in at session activation time, and referencing them throughout the session.

A special section in the session activation parameters section of the SCB is reserved for fields used exclusively by DFC. See Appendix A for more information about the SCB.

PROTOCOL BOUNDARY

The protocol boundary information for DFC is given in the prologues for the DFC.RCV (page 5-50) and DFC.SEND (page 5-41) procedures.

DETAILED DESCRIPTION OF DFC FUNCTIONS

REQUEST/RESPONSE FORMATTING

DFC enforces that the RH and RU request code fields for requests and responses 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 rules established at session activation are not violated, e.g., BBI=BB or EBI=EB and the session is not using brackets, or an FM profile 18 half-session tries to send a QEC DFC request.

Format checks do not involve the use of finite-state machines (FSMs). DFC does not allow any BIU with a format error to be sent.

Format checks are done before state checks. State checks are those that involve FSMs and FSMs require the BIU 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. There is at most one response sent per chain. Following an error, further requests on a chain are rejected.

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 proper chaining of requests is enforced for each half-session by DFC.SEND, using the FSM, FSM_CHAIN_SEND (page 5-72). The checking of received requests for proper chaining is provided for each half-session by DFC.RCV, using the FSM, FSM_CHAIN_RCV (page 5-72).

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:

- No-response chain: Each request in the chain is marked no-response.
- Exception-response chain: Each request in the chain is marked exception-response.
- Definite-response chain: The last request in the chain is marked definite-response; all other requests in the chain are marked exception-response.

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. Furthermore, the Form of Response Requested bits in the last request in a chain or in a request in error need be examined only when the half-session activation parameter, Chain Response Protocol, indicates that both definite-response and exception-response chains may be received. When the Chain Response Protocol parameter indicates that (1) only definite-response chains, (2) only exception-response chains, or (3) only no-response chains will be received, the setting of the Form of Response Requested bits on last-in-chain may be assumed (without checking) by the receiver.

The only normal-flow DFC request that can be sent while sending a normal-flow, multiple-request chain is CANCEL, which terminates the chain. The chain indicators in CANCEL are always set to (BC,EC).

If a chain sender is notified of an error in a chain being sent, the chain FSMs are reset by the sender's issuing either 1) an EC FMD request, 2) CANCEL (which carries EC), or 3) a higher-level reset command (e.g., CLEAR or UNBIND).

REQUEST/RESPONSE CORRELATION

When a response is received, DFC must know which request the response is for, and what information was on that request. Since many requests may be sent before a response is received, a method is needed to correlate the response to the request. The sequence number field (SNF) in the TH is used for this purpose. This field may contain a sequence number or an ID. Each FMD or DFC request sent has an SNF

value that is assigned by DFC. Each response sent contains (in its SNF) the SNF value of the request it is responding to. The SNF values of all outstanding (not yet responded to) requests must be unique for unambiguous response/request correlation.

DFC also has the responsibility to enforce that responses are formatted correctly with respect to the associated request.

In order to perform the functions specified above, DFC uses correlation tables. They are:

- CT_RCV_RQ_EXP (used for sending responses to requests received on the expedited flow)
- CT_SEND_RQ_EXP (used for receiving responses to requests sent on the expedited flow)
- CT_RCV_RQ_NORM (used for sending responses to requests received on the normal flow)
- CT_SEND_RQ_NORM (used for receiving responses to requests sent on the normal flow)

Each correlation table is composed of a variable number of entries. An entry corresponds to a chain (see section on chaining in this chapter for definition of a chain). New entries are added to the end of the table. Entries may be deleted from any part of the table. Entries are added to a correlation table when the first RU in a chain is sent or received. Entries are deleted when the chain is responded to and the complete chain has been sent or received. If an entry is responded to before it is completely sent or received the entry is deleted when the last RU of the chain is sent or received.

Each entry in a correlation table represents a chain of RUs. Information in a correlation table entry may contain:

- Selected RH indicators needed by DFC, such as BBI, EBI, and CDI.
- Entry type. Information is kept pertaining to the type of chain this entry represents. The entry types are:

  --Complete chain with no CANCEL

  --Complete chain with CANCEL (the CANCEL RU ended this chain)

  --Partial chain (this chain has not yet ended)

--CANCEL only (the CANCEL RU is the only RU in the chain--this condition should not occur if DFC send checks are correctly supported)

- Sequence number range for this chain. The beginning (first RU of a chain) and ending (last RU of a chain) sequence numbers are kept for each chain. A response to this chain is one having a sequence number falling within this range.

- Response sent/received. The correlation table entry records whether or not a response has been sent or received for a chain.

- DFC request code.

Depending upon the particular correlation table, entries may or may not have all the above information. A complete description of correlation table entries can be found in the "DFC Correlation Table Entity Declarations" section of this chapter (page 5-95).

Some examples of how the correlation table is used are:

- When sending a normal-flow response, a check is made to see if the RU_CTGY (RU category) in the response is equal to CT_RU_CTGY (RU category of the received request that was saved in the correlation table). If not, the response is rejected with a format error.

- In order to find out whether certain indicators were set on a request currently being responded to, the FSM_BSM_FSP (the first speaker's bracket state manager) has an input entry specified as follows:

  S,+RSP,FMD|LUSTAT,CT(BB,¬EB,CD)

  This means sending a positive response to an FMD or LUSTAT request that had specified BBI=BB, EBI=¬EB, and CDI=CD. The correlation table (CT) contains this information.

The number of entries in a correlation table may be limited if certain protocols are used. For example, if all chains are sent RQD (asking for definite response) and immediate request mode is used (only one RQD request may be waiting for a response at a time), the number of entries in the correlation table will never exceed one. This is because the response to the RQD chain will always delete that entry from the correlation table.

REQUEST/RESPONSE MODE PROTOCOLS

DFC enforces the following request/response protocols:

- Immediate request mode
- Delayed request mode
- Immediate response mode
- Delayed response mode

These protocols apply only to the normal flow. The expedited flow uses a separate protocol, which is enforced by the TC layer (Chapter 4.). Prose descriptions of both the normal- and expedited-flow protocols are in Chapter 4.

The immediate request mode protocol, allowing a maximum of one outstanding RQD request, is enforced by FSM_IMM_RQ_MODE_SEND (page 5-86) and FSM_IMM_RQ_MODE_RCV (page 5-86).

The delayed request mode protocol, because it allows multiple RQD requests to be outstanding, does not require any enforcing; therefore, there are no FSMs associated with this protocol.

The immediate response protocol, causing responses to be returned in the same order as the received requests, is enforced by using the correlation table. The oldest entry (oldest request received) in the correlation table must be the next entry responded to.

The delayed response protocol, allowing responses to be returned in any order except for the response to CHASE, is also enforced by using the correlation table. The response to CHASE can be sent only after all previous outstanding responses are sent.

SEND/RECEIVE MODE PROTOCOLS

The DFC.SEND and DFC.RCV protocol boundary with FMDS can be either half-duplex (HDX) or full-duplex (FDX). This attribute is referred to as the normal-flow send/receive mode. Informally, the boundary is half-duplex if it is incapable of concurrently passing normal-flow request chains in both directions, and full-duplex if it can.

Sessions can run (1) half-duplex flip-flop (HDX-FF)--with some variation in protocols depending on whether bracket protocols are also being used, (2) half-duplex-contention (HDX-CONT), or (3) full-duplex (FDX). The details of the session protocols for these modes are provided in this chapter; the following remarks also apply:

- HDX-FF (not using bracket protocol): Each half-session has a half-duplex DFC.SEND and DFC.RCV protocol boundary with FMDS; at session activation, one half-session is designated first sender, and the other, first receiver. 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 on the last request sent.

- HDX-FF (using bracket protocol): Each half-session has a half-duplex DFC.SEND and DFC.RCV protocol boundary with FMDS; at session activation, one half-session is designated HDX flip-flop bidder, and the other, HDX flip-flop first speaker. Using bracket protocol with HDX-FF protocol requires a synchronization between the two half-sessions. When between brackets, each half-session is in contention state: either may send. The contention winner is always the first speaker. When not between brackets, the half-sessions are subject to the protocol described above for HDX-FF not using brackets. See the section, "Bracket Protocols," for additional details.

- HDX-CONT: Each half-session has a half-duplex DFC.SEND and DFC.RCV protocol boundary with FMDS; at session activation, one half-session is designated the contention winner, and the other, the contention loser. The designated loser uses the queue (Q_TC_TO_DFC) for buffering normal-flow requests received while sending. Initially, both winner and loser are in the contention state, and either one may independently begin sending normal-flow requests.

  Normal-flow requests arriving at the loser, if it is sending, are queued; normal-flow requests arriving at the winner, if it is sending, may be temporarily queued or may be rejected with an appropriate negative response. Valid normal-flow requests, arriving at a nonsending half-session, place the half-session in a receiving state.

  The contention winner or loser reverts to contention state after sending or receiving the last request of a chain.

  Upon reverting to the contention state, a contention loser, or a contention winner that queues received BIUs, may dequeue any requests (and responses).

  Contention can be avoided through end user protocols or by use of the Change Direction indicator.

- FDX: The primary and secondary half-session DFC.SEND and DFC.RCV protocol boundaries with FMDS are full-duplex. The normal-flow request and response flows in each direction are independent; any correlation between flows is done at a level of control above that supplied by DFC.

The normal-flow send/receive mode protocols are enforced by DFC.SEND and DFC.RCV for each half-session as follows:

- If running half-duplex flip-flop, with or without brackets, the DFC.SEND and DFC.RCV protocol machines each use FSM_HDX_FF (page 5-84) to enforce the protocols.

- If running half-duplex contention, the contention winner uses FSM_HDX_CONT_WINNER (page 5-83) and the contention loser uses FSM_HDX_CONT_LOSER (page 5-82). When running with brackets the contention winner is always the bracket protocol first speaker and the contention loser is always the bracket protocol bidder.

- If running full-duplex, no FSM is used to enforce the protocol.

The Change Direction indicator (CDI) is used in the HDX-FF protocols, and may be used in the HDX-CONT 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 EB.

When running in half-duplex mode, a normal-flow request is not sent if DFC.SEND has not yet processed a required response for a previously received request.

BRACKETS PROTOCOL

A bracket is a sequence of normal-flow request chains and their responses, exchanged in either or both directions between two half-sessions. Bracket protocols allow half-sessions to contend for activating a bracket, and assist in resolving the race condition that can result from that contention. BIND parameters specify whether a bracket protocol is to be used in a session.

The rules for brackets regulate the initiation and termination of a bracket.

A bracket is delimited by use of Begin Bracket (BB) in the first request of the first chain, and End Bracket (EB) in the first request of the last chain in the bracket.

If brackets are used in a session, the BIND parameters specify one of the half-sessions as first speaker and the other as bidder. The first speaker has the freedom to begin a bracket without requesting permission from the other half-session to do so. The bidder must request and receive permission from the first speaker to begin a bracket.

The bracket protocols are enforced by DFC.SEND (page 5-41), DFC.RCV (page 5-50), and an appropriate bracket state manager (BSM) in each half-session. If the half-session is a bracket first speaker, FSM_BSM_FSP (page 5-70) is used; if a bracket bidder, FSM_BSM_BIDDER (page 5-68) is used.

Expedited requests and responses are not affected by bracket indicators on normal-flow requests, nor by the states of the BSMs.

BID is a normal-flow DFC request issued by the bidder to request permission to begin a bracket. A positive response to BID indicates that the first speaker will not begin a bracket, but will wait for the bidder to begin a bracket.

A negative response to BID 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. If the first speaker will send RTR later, the sense code with the negative response to BID is 0814 (Bracket Bid Reject--RTR Forthcoming). The bidder has the option of waiting for RTR or sending BID again. If the RTR will not be sent, the sense code is 0813 (Bracket Bid Reject--No RTR Forthcoming). In the latter case, the bidder must send BID again, if it still wants to begin a bracket. FSM_RTR_FSP records that the first speaker should transmit an RTR; FSM_RTR_BIDDER records that the bidder can expect (but need not await) an RTR.

Instead of sending BID followed by FMD request with BB, the bidder may attempt to initiate a bracket by simply sending an FMD request with BB, RQD. The first speaker grants the attempt (via positive response) or refuses it (via negative response indicating either 0814 (Bracket Bid Reject--RTR Forthcoming) or 0813 (Bracket Bid Reject--No RTR Forthcoming)). However, if the bidder terminates the chain of FMD requests that carries BB by sending CANCEL, then, regardless of response, the bracket is not initiated.

RTR may be issued by the first speaker to grant permission to the bidder to begin a bracket, or to find out if the bidder wants to begin a bracket. A positive response to RTR indicates that the bidder will initiate the next bracket. If the bidder does not want to initiate a bracket, it issues a negative response with the sense code, RTR Not Required.

The first speaker does not have to be granted permission (via a positive response) to begin a bracket. Any request sent by the first speaker carrying BB will begin a bracket. The first speaker does not send BID.

The following rules apply to the bracket indicators:

- BB may be set only on the first (or only) request of a chain.

- EB may be set only on the first (or only) request of a chain or on CANCEL. It indicates the last chain in the bracket. If EBI is set, CDI may not be set because EB overrides CD.

- BB and EB may occur on the same request of the same chain.

- BB or EB may be issued by either half-session, unless a BIND parameter or a private end-user protocol limits issuance.

- BB or EB may be set on FMD requests. EB may be set on any normal-flow DFC request except BID, BIS, or RTR. BB may not be set on any DFC requests except LUSTAT. Neither BB nor EB may be set on responses or on expedited requests.

- When the bidder is in the state BETB (i.e., FSM_BSM_BIDDER=BETB), it may send BB, without EB, only on the first (or only) request of a definite-response chain; it may send (BB,EB) on the first request of a definite-response chain, an exception-response chain, or a no-response chain.

- After sending a positive response to BID or receiving a positive response to RTR (i.e., FSM_BSM_FSP=PEND_BB), the first speaker must wait for the bidder to send an FMD request with BB. (When FSM_BSM_FSP=PEND_BB, the first speaker cannot send FMD requests or RTR.)

One of the following bracket termination rules is specified for the session, in the BIND parameters:

- Bracket Termination Rule 1 (Conditional Termination): Bracket termination is controlled by the form of response requested (definite, exception, or no-response) for the chain containing (-BB,EB). 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. A negative response to any but the last request in the chain allows the option of terminating or continuing

the bracket. The sender of the chain may end the bracket by sending CANCEL with EB, or by ending the chain with a request specifying exception response or no-response. Alternatively, the sender of the chain may continue the bracket by sending CANCEL without EB or by ending the chain with a request specifying definite response.

If the chain requests exception response or no-response, the bracket is terminated unconditionally when the last request of the chain that has EB in its first request is processed.

If BB and EB appear on the same chain, the bracket is unconditionally terminated when the last request of that chain is processed, regardless of the form of response requested.

- Bracket Termination Rule 2 (Unconditional Termination): A bracket is terminated unconditionally when the last request of the chain that has EB in its first request is processed, regardless of the form of response requested.

No more than one BB can be outstanding from a half-session.

The DFC requests--CANCEL, CHASE, LUSTAT, and QC--may flow both in and between brackets. Each of these four normal-flow DFC requests may carry EB, but only LUSTAT may carry BB. When brackets are used, only those FMD requests carrying BB may flow between brackets.

When CANCEL with EB is sent or received to terminate a chain that does not carry EB, then the bracket is terminated.

A bracket may contain one or more sync points (committed units of work), but units of work do not span brackets. The table below shows the meaning of EB as it relates to units of work:

Request	Meaning
EB,RQD1 | EB,RQE1	End, do not commit since no protected resources were changed
EB,RQD2|3	End, request commit
EB,RQE2|3	Not allowed, since request commit must have a response.
EB,LUSTAT(0824),RQD1 EB,LUSTAT(0824),RQE1	Abort current unit of work. See also FM header 7 (SNA LU-LU Session Types) and sense codes 0866, 0867, 0868.

Three types of error conditions are associated with the management of brackets:

- Violations detected at the sender. DFC.SEND rejects any attempt to transmit in violation of bracket protocols; e.g., the first speaker attempts to send BB while a bracket is in process (FSM_BSM_FSP=INB). The mechanism for passing this error information is an implementation option.

- Bracket protocol errors detected at the receiver due to sender error. These errors are receive errors and the receiver action is not specifically architected. Possible actions are suggested in the UPM procedure (see page 5-57) that is called when a receive error is detected.

- Errors detected at the receiver and caused by race conditions. The appropriate action is for the receiver to send the Bracket Race Error sense code on a negative response to the other half-session. This condition implies that a retry of the operation may be necessary.

ERROR RECOVERY PROTOCOL

Sessions operating HDX use one of two error recovery procedures:

- Contention loser responsible for recovery: The contention loser half-session assumes an HDX sending state at an appropriate moment after error detection;

it initiates the sending of requests to attempt recovery. Correspondingly, the contention winner half-session assumes an HDX receiving state and awaits recovery requests from the contention loser.

- Symmetric recovery: The half-session that sent a request found to be in error (by the receiver) assumes an HDX sending state at an appropriate moment; it initiates the sending of requests to attempt recovery. Correspondingly, the half-session that received the request in error assumes an HDX receiving state and awaits recovery requests from the sender.

The recovery management of an HDX FSM in a half-session that receives or sends a request chain containing an error is described in the HDX protocol machines. For sessions that limit the number of outstanding chains to one, the transition to the HDX recovery state (i.e., sending or receiving) is made after the last RU in the current chain has been received or sent.

For symmetric recovery, HDX-FF, and when multiple RQE chains are possible, the error recovery transition is delayed until the occurrence of an ERP synchronization event:

- For immediate response mode sessions, this event is RQD or RQE,CD.

- For delayed response mode sessions, this event is CHASE.

In all of these cases, the error recovery transition in the HDX machines occurs only after all chains in the session have been completely received. Thus, the error recovery procedure begins simply, with many DFC FSMs in their reset states.

STOP-BRACKET-INITIATION PROTOCOL

The stop-bracket-initiation protocol uses STOP BRACKET INITIATION (SBI) and BRACKET INITIATION STOPPED (BIS) to control the flow of normal-flow requests that initiate a new bracket. The principal FSMs used in this protocol are FSM_SBI_SEND (page 5-91) and FSM_SBI_RCV (page 5-91).

SBI is sent by either half-session to request that the receiving half-session stop initiating brackets by continued sending of BB and the BID request. The receiving half-session may continue to send BB and BID until it sends BIS in reply, i.e., BIS need not be sent at the next entry to the between bracket state following the receipt of SBI.

BIS is sent by the half-session that received SBI to acknowledge its agreement not to send BB or BID. A positive response to BIS places the SBI receiver (FSM_SBI_RCV) in NOBB state. While in NOBB state, any attempt to send BB or BID is rejected by DFC. ff

A BIS can also be sent unsolicited (i.e., when SBI has not been received) to inform the receiving half-session that the sending half-session will not send any subsequent BB or BID requests.

When the FM profile allows the use of (SBI, BIS) sequences to be initiated by either half-session, the LU.SVC_MGR for the primary half-session sends UNBIND if FSM_SBI_SEND and FSM_SBI_RCV are in the NOBB state and BSM is in the BETB state, or when FSM_SBI_RCV is in the NOBB state and FSM_BSM is in the PEND_BB state. This causes a session to be ended when the ability to initiate new work has been blocked for both directions.

The SBI protocol allows the LU services manager to end a session without interfering with any sync point requests that might have been issued by the sync point manager of the partner LU. This is true because sync point requests can occur only inside a bracket or at the end of a bracket. The receipt of CTERM(Forced) (see Chapter 8) or an equivalent signal from an end user of the LU results in sending an UNBIND without use of the SBI protocol. In these cases, as when the session fails (see Chapter 1), a sync point request from the partner LU may be overtaken by the UNBIND; this may cause locks on protected resources in the partner LU to be held until the session can be reactivated and the sync point managers resynchronized (using STSN, see Chapter 4).

QUIESCE PROTOCOL

The quiesce protocol provides a means for a half-session to stop its partner half-session from sending normal-flow requests. Only the normal flow is affected; the expedited flow is not affected. This protocol may be used for various reasons, e.g., one half-session may wish to end the session (via UNBIND) after it finishes receiving the rest of the current chain, or one half-session may wish to temporarily stop recieving because it has run low on some resource (like a buffer pool or auxiliary storage).

The quiesce protocol is symmetric; either half-session may "quiesce" its partner. For descriptive convenience, we shall call one half-session A and its partner half-session B. QEC, QC, and RELQ are the DFC requests used in the quiesce protocol.

QEC may be sent by A to request B to quiesce (stop sending normal-flow requests) at the end of the FMD chain that B is currently sending (if any). After receiving QEC, B may not begin any normal-flow request chain other than QC.

QC is sent by B after receiving QEC, to indicate that it has quiesced. QC is a normal-flow synchronizing request; it is the last normal-flow request sent by a quiesced half-session until RELQ is received.

While quiesced, a half-session accepts all FMD and normal-flow DFC requests and responds appropriately. Any FMD normal-flow requests to be sent in reply must be sent later. If this is not possible, a negative response (to the request that required a reply) must be sent with the sense code 0828, Reply Not Allowed. The decision to send a 0828 response is user defined. There is no enforcement of this condition by the DFC layer.

RELQ may be sent by A to remove the quiesced condition of B; i.e., to indicate that B may send normal-flow DFC and FMD requests.

If RELQ is received by a half-session that is not quiesced, but is otherwise able to process the request, a positive response is sent.

Typically, QEC is sent with the intention that RELQ will be sent at some time after the quiesce sequence, and that secondary-to-primary requests will resume on the normal flow.

The FSMs used to enforce this protocol are FSM_QEC_SEND (page 5-87) and FSM_QEC_RCV (page 5-87).

SHUTDOWN PROTOCOL

The shutdown protocol provides a means for a primary half-session to stop its partner secondary half-session from sending normal-flow requests. Only the normal flow is affected; the expedited flow is not affected. This protocol may be used when the primary wishes to end the session in an orderly manner. The secondary is "shut down" before ending the session with UNBIND.

The shutdown protocol is not symmetric; only the primary may shut down its partner (secondary). SHUTD, SHUTC, and RELQ are the DFC requests used in the shutdown protocol.

SHUTD is sent by the primary to request that the secondary stop sending normal-flow requests as soon as convenient. The secondary determines what convenient is; for example, it could be at the end of current bracket. After reaching the

convenient point, the secondary sends SHUTC. After receiving a positive response to SHUTC, the secondary has been shut down (quiesced) and may not send any normal-flow requests unless it subsequently receives a RELQ.

Since SHUTC is expedited, it may pass normal-flow requests that were previously sent by the secondary. The secondary may avoid this race condition by asking and waiting for a definite response to the last request sent (if the primary is using immediate response mode), or by sending CHASE and waiting for the CHASE response before sending SHUTC.

While in shutdown (quiesced) state, a half-session accepts all FMD and normal-flow DFC requests and responds appropriately. Any FMD normal-flow requests to be sent in reply are sent later. If this is not possible, a negative response (to the request that required a reply) is sent with the sense code 0828, Reply Not Allowed. The decision to send a 0828 response is user defined. There is no enforcement of this condition by the DFC layer.

RELQ may be sent by the primary to remove the shutdown (quiesced) condition of the secondary, i.e., to indicate that the secondary may send normal-flow requests.

If RELQ is received by a secondary that is not quiesced, but is otherwise able to process the request, a positive response is sent.

The FSMs used to enforce this protocol are FSM_SHUTD_SEND (page 5-92) and FSM_SHUTD_RCV (page 5-92).

RELATIONSHIP OF QUIESCE AND SHUTDOWN PROTOCOLS

The implications of the quiesce and shutdown protocols to FMDS are as follows. The quiecse protocol requires a more stringent quiescing than does the shutdown protocol. QEC (of the quiesce protocol) requests the receiving FMDS to stop sending requests on the normal flow after the end of the current chain, if any. SHUTD (of the shutdown protocol) requests the receiving FMDS to stop sending requests on the normal flow when it is ready to end the session.

Note that RELQ is used in both the quiesce and shutdown protocols, hence, one RELQ will remove a half-session from the quiesced condition, i.e., issuing a positive response to RELQ resets both FSM_QEC_RCV (page 5-87) and FSM_SHUTD_RCV (page 5-92).

QUEUED RESPONSE PROTOCOL

DFC enforces the setting of the QRI bit on requests. See Chapter 2 for a discussion of this RH indicator.

The setting of the QRI bit is the same for all RUs in a chain. DFC enforces this using FSM_QRI_CHAIN_SEND (page 5-89) and FSM_QRI_CHAIN_RCV (page 5-88).

QR can be indicated on any request chain; ¬QR cannot be indicated: (1) on CHASE when a normal-flow request chain indicating QR is outstanding and delayed response mode is specified for that flow, (2) on any normal-flow request chain when a normal-flow request chain indicating QR is outstanding and immediate response mode is specified for that flow, or (3) on any request in a chain that indicates (¬BB, EB) when half-duplex contention or full-duplex is specified as the normal-flow send/receive mode, unless higher-level protocols can be invoked to avoid an FMD request, sent before RSP(RQ(EB)), being received after the RSP(RQ(EB)) is received. These protocol rules are enforced by FSM_QRI_CHECK_SEND (page 5-88) and the procedure DFC.SEND_CHECKS (PAGE 5-42).

DFC REQUEST/RESPONSE REFERENCE

DFC REQUEST/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. Figures 5-3 and 5-4 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/response flows on.

DFC REQUEST ----->	BID BIS RTR	CANCEL CHASE QC	LUSTAT	QEC RELQ RSHUTD SBI SHUTC SHUTD SIGNAL
HEADER INDICATORS				
TH BYTE 0 BIT 7 EFI	NORMAL	NORMAL	NORMAL	EXP
RH BYTE 0 BIT 0 RRI	RQ	RQ	RQ	RQ
BITS 1-2 RU_CTGY	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 reserved	0	0	0	0

Notes:

1. *XX means either XX or ¬XX.
2. See Chapter 2 and Appendix D for complete TH and RH descriptions.
3. If EBI=EB, CDI must be ¬CD.
4. For LUSTAT, (DR1I,DR2I) = (0,1) | (1,0) | (1,1).

Figure 5-3. DFC Request Formats

HEADER INDICATORS		DFC RESPONSE----->	BID BIS CANCEL CHASE QC RTR	LUSTAT	QEC RELQ RSHUTD SBI SHUTC SHUTD SIGNAL
TH BYTE 0	BIT 7	EFI	NORMAL	NORMAL	EXP
RH BYTE 0	BIT 0	RRI	RSP	RSP	RSP
	BITS 1-2	RU_CTGY	DFC	DFC	DFC
	BIT 3	reserved	0	0	0
	BIT 4	FI	1	1	1
	BIT 5	SDI	*SD	*SD	*SD
	BIT 6	BCI	BC	BC	BC
	BIT 7	ECI	EC	EC	EC
RH BYTE 1	BIT 0	DR1I	DR1	DR1	DR1
	BIT 1	reserved	0	0	0
	BIT 2	DR2I	¬DR2	*DR2	¬DR2
	BIT 3	RTI	POS|NEG	POS|NEG	POS|NEG
	BIT 4	reserved	0	0	0
	BIT 5	reserved	0	0	0
	BIT 6	QRI	*QR	*QR	¬QR
	BIT 7	PI	*PAC	*PAC	¬PAC
RH BYTE 2	BIT 0	reserved	0	0	0
	BIT 1	reserved	0	0	0
	BIT 2	reserved	0	0	0
	BIT 3	reserved	0	0	0
	BIT 4	reserved	0	0	0
	BIT 5	reserved	0	0	0
	BIT 6	reserved	0	0	0
	BIT 7	reserved	0	0	0

Notes

1. *XX means either XX or ¬XX.

2. See Chapter 2 and Appendix D for complete TH and RH descriptions.

Figure 5-4. DFC Response Formats

DFC REQUEST/RESPONSE DESCRIPTIONS (ALPHABETICAL ORDER)

BID (BID)

Flow: Bidder to first speaker
(Normal)

Principal FSMs: FSM_BSM_FSP (Page 5-70)
FSM_BSM_BIDDER (Page 5-68)
FSM_RTR_FSP (Page 5-90)
FSM_RTR_BIDDER (Page 5-90)

BID is used by the bidder to request permission to initiate a bracket, and is used only when using brackets. See "Brackets Protocol" on page 5-14.

BIS (BRACKET INITIATION STOPPED)

Flow: Primary to secondary and secondary to primary
(Normal)

Principal FSMs: FSM_SBI_SEND (Page 5-91)
FSM_SBI_RCV (Page 5-91)

BIS is sent by the half-session that received SBI to acknowledge its agreement not to send BB or BID. It is used only when using brackets. See "Stop-bracket-initiation protocol" on page 5-19.

CANCEL (CANCEL)

Flow: Primary to secondary and secondary to primary
(Normal)

Principal FSMs: FSM_CHAIN_SEND (Page 5-72)
FSM_CHAIN_RCV (Page 5-72)
FSM_BSM_BIDDER (Page 5-68)
FSM_BSM_FSP (Page 5-70)

CANCEL may be sent by a half-session to terminate a partially sent chain of FMD requests. CANCEL may be sent only when a chain is in process (i.e., FSM_CHAIN_SEND:INC). The sending half-session may send CANCEL to end a partially sent chain if a negative response is received for a request in the chain, or for some other reason. See "Chaining Protocol" on page 5-8.

The setting of EBI on CANCEL may override the setting of EBI on the first request of the chain. See "Brackets Protocol" on page 5-14.

CHASE (CHASE)

Flow: Primary to secondary and secondary to primary (Normal)

Principal FSMs: Receive requests correlation table CT_RCV_RQ_NORM

CHASE is sent by a half-session to request the receiving half-session to return all outstanding normal-flow responses to requests previously received from the issuer of CHASE. The receiver of CHASE sends the response to CHASE after processing (and sending any necessary responses to) all requests received before the CHASE.

A half-session can use CHASE before issuing SHUTDOWN COMPLETE (SHUTC), so that no valid negative responses will be received after the half-session has quiesced and become unable to correct the requests in error. When the half-session uses immediate response mode, an FMD request specifying definite response serves the same purpose as CHASE; i.e., if the receiving half-session uses immediate response mode and the sending half-session can send requests specifying definite response, it is not necessary to use CHASE.

LUSTAT (LOGICAL UNIT STATUS)

Flow: Primary to secondary and secondary to primary (Normal)

Principal FSMs: None in DFC

LUSTAT is used by one half-session to send four bytes of status information to its paired half-session. The RU format (see Appendix E) allows the sending of either end user information or LU status information, e.g., about a specified LU component. If the high-order two bytes of status information are 0 then the low-order two bytes carry end user information and may be set to any value. In general, LUSTAT is used to report about failures and error recovery conditions for a local device of an LU. No specific LUSTAT FSMs are required in DFC to handle the sending and receiving of LUSTAT.

QC (QUIESCE COMPLETE)

Flow: Primary to secondary and secondary to primary (Normal)

Principal FSMs: FSM_QEC_SEND (Page 5-87)
FSM_QEC_RCV (Page 5-87)

QC is sent by a half-session after receiving QEC, to indicate that it has quiesced. See "Quiesce Protocol" on page 5-20.

QEC (QUIESCE AT END OF CHAIN)

Flow: Primary to secondary and secondary to primary (Expedited)

Principal FSMs: FSM_QEC_SEND (Page 5-87)
FSM_QEC_RCV (Page 5-87)

QEC is sent by a half-session to quiesce its partner half-session after it (the partner) finishes sending the current chain (if any). See "Quiesce Protocol" on page 5-20.

RELQ (RELEASE QUIESCE)

Flow: Primary to secondary and secondary to primary (Expedited)

Principal FSMs: FSM_QEC_SEND (Page 5-87)
FSM_QEC_RCV (Page 5-87)
FSM_SHUTD_SEND (Page 5-92)
FSM_SHUTD_RCV (Page 5-92)

RELQ is used to release a half-session from a quiesced state. See "Quiesce Protocol" (page 5-20) and "Shutdown Protocol" (page 5-21).

RSHUTD (REQUEST SHUTDOWN)

Flow: Secondary to primary (Expedited)

Principal FSM: None

RSHUTD is sent from the secondary to the primary to indicate that the secondary is ready to have the session deactivated. No specific RSHUTD FSMs are required in DFC to handle the sending and receiving of RSHUTD. Note: Contrary to its name, RSHUTD does not request a shutdown--SHUTD is not a proper reply; rather, it requests an UNBIND.

RTR (READY TO RECEIVE)

Flow: First speaker to bidder (Normal)

Principal FSMs: FSM_BSM_FSP (Page 5-70)
FSM_BSM_BIDDER (Page 5-68)
FSM_RTR_FSP (Page 5-90)
FSM_RTR_BIDDER (Page 5-90)

RTR indicates to the bidder that it is now allowed to initiate a bracket. RTR is issued by the first speaker, and is used only when using brackets. See "Brackets Protocol" on page 5-14.

SBI (STOP BRACKET INITIATION)

Flow: Primary to secondary and secondary to primary
(Expedited)

Principal FSMs: FSM_SBI_SEND (Page 5-91)
FSM_SBI_RCV (Page 5-91)

SBI is sent by either half-session to request that the receiving half-session stop initiating brackets by continued sending of BB and the BID request. See "Stop-Bracket-Initiation Protocol" on page 5-19.

SHUTC (SHUTDOWN COMPLETE)

Flow: Secondary to primary
(Expedited)

Principal FSMs: FSM_SHUTD_SEND (Page 5-92)
FSM_SHUTD_RCV (Page 5-92)

SHUTC is sent by a secondary half-session to indicate it is in the shutdown (quiesced) state. See "Shutdown Protocol" on page 5-21.

SHUTD (SHUTDOWN)

Flow: Primary to secondary
(Expedited)

Principal FSMs: FSM_SHUTD_SEND (Page 5-92)
FSM_SHUTD_RCV (Page 5-92)

SHUTD is sent by the primary to request that the secondary shutdown (quiesce) as soon as convenient. See "Shutdown Protocol" on page 5-21.

SIG (SIGNAL)

Flow: Primary to secondary and secondary to primary (Expedited)

Principal FSMs: None in DFC

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. These values are used in higher level protocols and are defined in Appendix E. No specific SIG FSMs are required in DFC to handle the sending and receiving of SIG.

```
SESSACT.DFC_INITIALIZE: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THE PURPOSE OF THIS PROCEDURE IS TO SET UP, IN THE SCB, VARIOUS │
  │                 SESSION PARAMETERS, DFC COMMAND USAGE, AND DFC FSM USAGE FOR THE │
  │                 SESSION. THIS PROCEDURE IS EXECUTED AT SESSION ACTIVATION TIME. │
  │                 THIS PROCEDURE IS NOT CALLED BY DFC; IT IS CALLED BY THE COMMON │
  │                 SESSION CONTROL MANAGER (CSC_MGR, CHAPTER 13) ON SENDING OR │
  │                 RECEIVING A POSITIVE RESPONSE TO A SESSION ACTIVATION REQUEST. │
  │                                                                      │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
  │                 DFC_INIT_DFC_USAGE              PAGE 5-32            │
  │                 DFC_INIT_FSM_USAGE              PAGE 5-34            │
  │                 DFC_INIT_MISC_SESSION_PARMS     PAGE 5-37            │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  CALL DFC_INIT_MISC_SESSION_PARMS;                  /* PAGE 5-37      */
  CALL DFC_INIT_DFC_USAGE;                           /* PAGE 5-32      */
  CALL DFC_INIT_FSM_USAGE;                           /* PAGE 5-34      */

  RETURN;
END SESSACT.DFC_INITIALIZE;
```

```
DFC_INIT_DFC_USAGE: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE SETS  UP SCB INDICATORS FOR EACH  DFC COMMAND.  THESE  │
│                 INDICATORS SPECIFY WHETHER OR NOT THE DFC COMMAND MAY BE SENT AND/OR  │
│                 RECEIVED.   THE SETTING  OF  THESE INDICATORS  IS  BASED  ON THE  FM   │
│                 PROFILE.                                                 │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 SESSACT.DFC_INITIALIZE              PAGE 5-31            │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 DFC_INIT_DFC_USAGE_BID_RTR          PAGE 5-33            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

  SCB.DFC_REQUESTS = 0;                                 /* INITIALIZE INDICATORS TO
                                                           NOT_ALLOWED          */
  IF SCB.HALF_SESSION = PRI THEN
    DO;
    . IF SCB.PRI_CHAIN_USE = MULTIPLE THEN
    .   SCB.DFC_CANCEL_SEND = ALLOWED;
    . IF SCB.SEC_CHAIN_USE = MULTIPLE THEN
    .   SCB.DFC_CANCEL_RCV = ALLOWED;
    END;
  ELSE                                                  /* SECONDARY            */
    DO;
    . IF SCB.SEC_CHAIN_USE = MULTIPLE THEN
    .   SCB.DFC_CANCEL_SEND = ALLOWED;
    . IF SCB.PRI_CHAIN_USE = MULTIPLE THEN
    .   SCB.DFC_CANCEL_RCV = ALLOWED;
    END;

  SELECT ANYORDER(SCB.FM_PROFILE);
  . WHEN(0)
  .   DO;
  .   . IF SCB.HALF_SESSION = PRI THEN
  .   .   SCB.DFC_LUSTAT_RCV = ALLOWED;
  .   . ELSE
  .   .   SCB.DFC_LUSTAT_SEND = ALLOWED;
  .   END;
  . WHEN(3)
  .   DO;
  .   . SCB.DFC_CHASE_RCV = ALLOWED;
  .   . SCB.DFC_CHASE_SEND = ALLOWED;
  .   . SCB.DFC_SIG_RCV = ALLOWED;
  .   . SCB.DFC_SIG_SEND = ALLOWED;
  .   . IF SCB.HALF_SESSION = PRI THEN
  .   .   DO;
  .   .   . SCB.DFC_RSHUTD_RCV = ALLOWED;
  .   .   . SCB.DFC_LUSTAT_RCV = ALLOWED;
  .   .   . SCB.DFC_SHUTC_RCV = ALLOWED;
  .   .   . SCB.DFC_SHUTD_SEND = ALLOWED;
  .   .   END;
  .   . ELSE
  .   .   DO;
  .   .   . SCB.DFC_RSHUTD_SEND = ALLOWED;
  .   .   . SCB.DFC_LUSTAT_SEND = ALLOWED;
  .   .   . SCB.DFC_SHUTC_SEND = ALLOWED;
  .   .   . SCB.DFC_SHUTD_RCV = ALLOWED;
  .   .   END;
  .   . CALL DFC_INIT_DFC_USAGE_BID_RTR;              /* PAGE 5-33            */
  .   END;
  . WHEN(4)
  .   DO;
  .   . SCB.DFC_CHASE_RCV = ALLOWED;
  .   . SCB.DFC_CHASE_SEND = ALLOWED;
  .   . SCB.DFC_LUSTAT_RCV = ALLOWED;
  .   . SCB.DFC_LUSTAT_SEND = ALLOWED;
  .   . SCB.DFC_QC_RCV = ALLOWED;
  .   . SCB.DFC_QC_SEND = ALLOWED;
  .   . SCB.DFC_QEC_RCV = ALLOWED;
  .   . SCB.DFC_QEC_SEND = ALLOWED;
  .   . SCB.DFC_RELQ_RCV = ALLOWED;
  .   . SCB.DFC_RELQ_SEND = ALLOWED;
  .   . SCB.DFC_SIG_RCV = ALLOWED;
  .   . SCB.DFC_SIG_SEND = ALLOWED;
  .   . IF SCB.HALF_SESSION = PRI THEN
  .   .   DO;
  .   .   . SCB.DFC_RSHUTD_RCV = ALLOWED;
  .   .   . SCB.DFC_SHUTC_RCV = ALLOWED;
  .   .   . SCB.DFC_SHUTD_SEND = ALLOWED;
  .   .   END;
  .   . ELSE
  .   .   DO;
  .   .   . SCB.DFC_RSHUTD_SEND = ALLOWED;
  .   .   . SCB.DFC_SHUTC_SEND = ALLOWED;
  .   .   . SCB.DFC_SHUTD_RCV = ALLOWED;
  .   .   END;
  .   . CALL DFC_INIT_DFC_USAGE_BID_RTR;              /* PAGE 5-33            */
  .   END;
```

```
   . WHEN(6)
   .   DO;
   .   . IF SCB.HALF_SESSION = PRI THEN
   .   .   SCB.DFC_LUSTAT_RCV = ALLOWED;
   .   . ELSE
   .   .   SCB.DFC_LUSTAT_SEND = ALLOWED;
   .   END;
   . WHEN(7)
   .   DO;
   .   . IF SCB.THIS_HALF_SESSION_RQ_MODE = DELAYED THEN
   .   .   SCB.DFC_CHASE_SEND = ALLOWED;
   .   . IF SCB.PARTNER_HALF_SESSION_RQ_MODE = DELAYED THEN
   .   .   SCB.DFC_CHASE_RCV = ALLOWED;
   .   . SCB.DFC_LUSTAT_RCV = ALLOWED;
   .   . SCB.DFC_LUSTAT_SEND = ALLOWED;
   .   . SCB.DFC_SIG_RCV = ALLOWED;
   .   . SCB.DFC_SIG_SEND = ALLOWED;
   .   . IF SCB.HALF_SESSION = PRI THEN
   .   .   SCB.DFC_RSHUTD_RCV = ALLOWED;
   .   . ELSE
   .   .   SCB.DFC_RSHUTD_SEND = ALLOWED;
   .   END;
   . WHEN(18)
   .   DO;
   .   . SCB.DFC_CHASE_RCV = ALLOWED;
   .   . SCB.DFC_CHASE_SEND = ALLOWED;
   .   . SCB.DFC_LUSTAT_RCV = ALLOWED;
   .   . SCB.DFC_LUSTAT_SEND = ALLOWED;
   .   . SCB.DFC_SIG_RCV = ALLOWED;
   .   . SCB.DFC_SIG_SEND = ALLOWED;
   .   . IF SCB.USING_BRACKETS = YES THEN
   .   .   DO;
   .   .   . SCB.DFC_BIS_RCV = ALLOWED;
   .   .   . SCB.DFC_BIS_SEND = ALLOWED;
   .   .   . SCB.DFC_SBI_RCV = ALLOWED;
   .   .   . SCB.DFC_SBI_SEND = ALLOWED;
   .   .   END;
   .   . CALL DFC_INIT_DFC_USAGE_BID_RTR;          /* PAGE 5-33          */
   .   END;
   . OTHERWISE;
   END;

   RETURN;
END DFC_INIT_DFC_USAGE;



DFC_INIT_DFC_USAGE_BID_RTR: PROCEDURE;

                                                                          /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE SETS THE SCB INDICATORS FOR BID AND RTR USAGE.
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 DFC_INIT_DFC_USAGE                PAGE 5-32
   └──────────────────────────────────────────────────────────────────────┘
                                                                          */

   IF SCB.USING_BRACKETS = YES THEN
     DO;
     . IF SCB.FIRST_SPEAKER = YES THEN
     .   DO;
     .   . SCB.DFC_BID_RCV = ALLOWED;
     .   . SCB.DFC_RTR_SEND = ALLOWED;
     .   END;
     . ELSE                                        /* BIDDER             */
     .   DO;
     .   . SCB.DFC_BID_SEND = ALLOWED;
     .   . SCB.DFC_RTR_RCV = ALLOWED;
     .   END;
     END;

   RETURN;
END DFC_INIT_DFC_USAGE_BID_RTR;
```

```
DFC_INIT_FSM_USAGE: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE  SETS UP THE FSM  USAGE FOR THIS  HALF-SESSION'S DFC.  │
  │                 IT USES THE  "#" VARIABLE TO SELECT FSM'S FOR  THE HALF-SESSION.  IF  │
  │                 AN FSM IS TO BE USED, ITS #NAME  IS SET TO THE CHARACTER STRING NAME  │
  │                 OF THE  FSM TO BE  USED FOR THE  HALF-SESSION.  IF  NO FSM IS  TO BE  │
  │                 USED, THE #NAME IS SET TO NO_OP.                             │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 SESSACT.DFC_INITIALIZE              PAGE 5-31                │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 DFC_INIT_FSM_USAGE_BSM_SBI_RTR      PAGE 5-35                │
  │                 DFC_INIT_FSM_USAGE_HDX_RES          PAGE 5-36                │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

                                /* INITIALIZE ALL FSM'S TO 'NO_OP'                */
  #FSM_BSM = 'NO_OP';
  #FSM_CHAIN_RCV = 'NO_OP';
  #FSM_CHAIN_SEND = 'NO_OP';
  #FSM_CONTROL_BSM_RSP_RCV = 'NO_OP';
  #FSM_CONTROL_BSM_RSP_SEND = 'NO_OP';
  #FSM_CONTROL_HDX_RSP_RCV = 'NO_OP';
  #FSM_CONTROL_HDX_RSP_SEND = 'NO_OP';
  #FSM_EBCD_RCV = 'NO_OP';
  #FSM_EBCD_SEND = 'NO_OP';
  #FSM_HDX = 'NO_OP';
  #FSM_IMM_RQ_MODE_RCV = 'NO_OP';
  #FSM_IMM_RQ_MODE_SEND = 'NO_OP';
  #FSM_QEC_RCV = 'NO_OP';
  #FSM_QEC_SEND = 'NO_OP';
  #FSM_QRI_CHECK_SEND = 'NO_OP';
  #FSM_QRI_CHAIN_RCV = 'NO_OP';
  #FSM_QRI_CHAIN_SEND = 'NO_OP';
  #FSM_RES = 'NO_OP';
  #FSM_RTR = 'NO_OP';
  #FSM_SBI_RCV = 'NO_OP';
  #FSM_SBI_SEND = 'NO_OP';
  #FSM_SHUTD = 'NO_OP';

                    /* SET UP FSM'S TO BE USED BY THIS HALF SESSION              */
  #FSM_QRI_CHECK_SEND = 'FSM_QRI_CHECK_SEND';
  IF SCB.HALF_SESSION = PRI THEN
    DO;
    . IF SCB.PRI_CHAIN_USE = MULTIPLE THEN
    .   DO;
    .   . #FSM_CHAIN_SEND = 'FSM_CHAIN_SEND';
    .   . #FSM_QRI_CHAIN_SEND = 'FSM_QRI_CHAIN_SEND';
    .   END;
    . IF SCB.SEC_CHAIN_USE = MULTIPLE THEN
    .   DO;
    .   . #FSM_CHAIN_RCV = 'FSM_CHAIN_RCV';
    .   . #FSM_QRI_CHAIN_RCV = 'FSM_QRI_CHAIN_RCV';
    .   END;
    END;
  ELSE                                              /* SECONDARY               */
    DO;
    . IF SCB.SEC_CHAIN_USE = MULTIPLE THEN
    .   DO;
    .   . #FSM_CHAIN_SEND = 'FSM_CHAIN_SEND';
    .   . #FSM_QRI_CHAIN_SEND = 'FSM_QRI_CHAIN_SEND';
    .   END;
    . IF SCB.PRI_CHAIN_USE = MULTIPLE THEN
    .   DO;
    .   . #FSM_CHAIN_RCV = 'FSM_CHAIN_RCV';
    .   . #FSM_QRI_CHAIN_RCV = 'FSM_QRI_CHAIN_RCV';
    .   END;
    END;

  IF SCB.DFC_QEC_RCV = ALLOWED THEN
    #FSM_QEC_RCV = 'FSM_QEC_RCV';
  IF SCB.DFC_QEC_SEND = ALLOWED THEN
    #FSM_QEC_SEND = 'FSM_QEC_SEND';

  IF SCB.DFC_SHUTD_RCV = ALLOWED THEN
    #FSM_SHUTD = 'FSM_SHUTD_RCV';
  ELSE
    IF SCB.DFC_SHUTD_SEND = ALLOWED THEN
      #FSM_SHUTD = 'FSM_SHUTD_SEND';

  IF SCB.THIS_HALF_SESSION_RQ_MODE = IMMEDIATE THEN
    #FSM_IMM_RQ_MODE_SEND = 'FSM_IMM_RQ_MODE_SEND';
  IF SCB.PARTNER_HALF_SESSION_RQ_MODE = IMMEDIATE THEN
    #FSM_IMM_RQ_MODE_RCV = 'FSM_IMM_RQ_MODE_RCV';

  CALL DFC_INIT_FSM_USAGE_BSM_SBI_RTR;              /* PAGE 5-35               */
  CALL DFC_INIT_FSM_USAGE_HDX_RES;                  /* PAGE 5-36               */

  RETURN;
END DFC_INIT_FSM_USAGE;
```

```
DFC_INIT_FSM_USAGE_BSM_SBI_RTR: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE SETS UP FSM USAGE FOR THE BSM, SBI, AND RTR FSM'S.
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 DFC_INIT_FSM_USAGE                 PAGE 5-34
  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):
  │                 UPM_FDX_BRACKETS                   PAGE 5-37
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  IF SCB.USING_BRACKETS = YES THEN
    DO;
    . IF SCB.SEND_RCV_MODE ¬= FULL_DUPLEX THEN
    .   DO;
    .   . IF SCB.DFC_SBI_RCV = ALLOWED THEN
    .   .   #FSM_SBI_RCV = 'FSM_SBI_RCV';
    .   .
    .   . IF SCB.DFC_SBI_SEND = ALLOWED THEN
    .   .   #FSM_SBI_SEND = 'FSM_SBI_SEND';
    .   .
    .   . #FSM_EBCD_RCV = 'FSM_EBCD_RCV';
    .   . #FSM_EBCD_SEND = 'FSM_EBCD_SEND';
    .   .
    .   . #FSM_CONTROL_BSM_RSP_RCV = 'FSM_CONTROL_BSM_RSP_RCV';
    .   . #FSM_CONTROL_BSM_RSP_SEND = 'FSM_CONTROL_BSM_RSP_SEND';
    .   .
    .   . IF SCB.FIRST_SPEAKER = YES THEN
    .   .   DO;
    .   .   . #FSM_BSM = 'FSM_BSM_FSP';
    .   .   . #FSM_RTR = 'FSM_RTR_FSP';
    .   .   END;
    .   . ELSE
    .   .   DO;
    .   .   . #FSM_BSM = 'FSM_BSM_BIDDER';
    .   .   . #FSM_RTR = 'FSM_RTR_BIDDER';
    .   .   END;
    .   END;
    . ELSE                                          /* FULL DUPLEX          */
    .   CALL UPM_FDX_BRACKETS;                      /* PAGE 5-37            */
    END;

  RETURN;
END DFC_INIT_FSM_USAGE_BSM_SBI_RTR;
```

```
DFC_INIT_FSM_USAGE_HDX_RES: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS  PROCEDURE SETS  UP  THE FSM  USAGE  FOR  NORMAL-FLOW SEND  AND  │
  │                 RECEIVE MODE (#FSM_HDX) AND RESOURCE (#FSM_RES) FSM'S.       │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 DFC_INIT_FSM_USAGE                 PAGE 5-34                 │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  IF  ¬(SCB.HALF_SESSION = PRI & SCB.RECOVERY_RESP = SYMMETRIC) THEN
    #FSM_RES = 'FSM_RES';

  SELECT ANYORDER(SCB.SEND_RCV_MODE);
  . WHEN(HDX_CONTENTION)
  .   DO;
  .   . #FSM_CONTROL_HDX_RSP_RCV = 'FSM_CONTROL_HDX_RSP_RCV';
  .   . #FSM_CONTROL_HDX_RSP_SEND = 'FSM_CONTROL_HDX_RSP_SEND';
  .   .
  .   . IF (SCB.HALF_SESSION = PRI & SCB.CONT_WIN = PRI) |
  .   .   (SCB.HALF_SESSION = SEC & SCB.CONT_WIN = SEC) THEN
  .   .   #FSM_HDX = 'FSM_HDX_CONT_WINNER';
  .   . ELSE
  .   .   DO;
  .   .   . #FSM_HDX = 'FSM_HDX_CONT_LOSER';
  .   .   . #FSM_RES = 'NO_OP';                          /* LOSER MAY NOT HAVE RESOURCE FSM */
  .   .   END;
  .   END;
  .
  . WHEN(HDX_FLIP_FLOP)
  .   DO;
  .   . IF SCB.RECOVERY_RESP = SYMMETRIC THEN
  .   .   DO;
  .   .   . IF SCB.THIS_HALF_SESSION_RQ_MODE = IMMEDIATE THEN
  .   .   .   #FSM_CONTROL_HDX_RSP_RCV = 'FSM_CONTROL_HDX_RSP_RCV_ERP_IM';
  .   .   . ELSE                                         /* DELAYED REQUEST MODE            */
  .   .   .   #FSM_CONTROL_HDX_RSP_RCV = 'FSM_CONTROL_HDX_RSP_RCV_ERP_DL';
  .   .   .
  .   .   . IF SCB.PARTNER_HALF_SESSION_RQ_MODE = IMMEDIATE THEN
  .   .   .   #FSM_CONTROL_HDX_RSP_SEND = 'FSM_CONTROL_HDX_RSP_SEND_ERP_IM';
  .   .   . ELSE                                         /* DELAYED REQUEST MODE            */
  .   .   .   #FSM_CONTROL_HDX_RSP_SEND = 'FSM_CONTROL_HDX_RSP_SEND_ERP_DL';
  .   .   END;
  .   . ELSE                                             /* NOT SYMMETRIC ERROR RECOVERY    */
  .   .   DO;
  .   .   . #FSM_CONTROL_HDX_RSP_RCV = 'FSM_CONTROL_HDX_RSP_RCV';
  .   .   . #FSM_CONTROL_HDX_RSP_SEND = 'FSM_CONTROL_HDX_RSP_SEND';
  .   .   END;
  .   .
  .   . #FSM_HDX = 'FSM_HDX_FF';
  .   .
  .   . IF SCB.USING_BRACKETS = YES & SCB.FIRST_SPEAKER = NO THEN
  .   .   #FSM_RES = 'NO_OP';                            /* BIDDER MAY NOT HAVE RES FSM     */
  .   END;
  .
  . WHEN(FULL_DUPLEX)
  .   ;
  END;

  RETURN;
END DFC_INIT_FSM_USAGE_HDX_RES;
```

```
DFC_INIT_MISC_SESSION_PARMS: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THE PURPOSE OF  THIS PROCEDURE IS TO SET UP  SESSION PARAMETERS THAT │
  │                 NEED TO BE KNOWN BY DFC.                                     │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 SESSACT.DFC_INITIALIZE            PAGE 5-31                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

                                                    /* SET UP BRACKET OPTIONS   */

  IF SCB.FM_PROFILE = (2 | 3 | 4 | 7 | 18) &
    (SCB.BRACKETS_RESET_STATE = BETB |
    SCB.PRI_EB_IND = MAY_SEND |
    SCB.SEC_EB_IND = MAY_SEND) THEN
    DO;                                             /* BRACKETS ARE BEING USED  */
    . SCB.USING_BRACKETS = YES;
    . IF (SCB.HALF_SESSION = PRI & SCB.CONT_WIN = PRI) |
    .   (SCB.HALF_SESSION = SEC & SCB.CONT_WIN = SEC) THEN
    .   SCB.FIRST_SPEAKER = YES;
    . ELSE
    .   SCB.FIRST_SPEAKER = NO;
    END;
  ELSE
    SCB.USING_BRACKETS = NO;

                                                    /* SET UP REQUEST MODE      */

  IF (SCB.HALF_SESSION = PRI & SCB.PRI_RQ_MODE = IMMEDIATE) |
    (SCB.HALF_SESSION = SEC & SCB.SEC_RQ_MODE = IMMEDIATE) THEN
    SCB.THIS_HALF_SESSION_RQ_MODE = IMMEDIATE;
  ELSE
    SCB.THIS_HALF_SESSION_RQ_MODE = DELAYED;

  IF (SCB.HALF_SESSION = PRI & SCB.SEC_RQ_MODE = IMMEDIATE) |
    (SCB.HALF_SESSION = SEC & SCB.PRI_RQ_MODE = IMMEDIATE) THEN
    SCB.PARTNER_HALF_SESSION_RQ_MODE = IMMEDIATE;
  ELSE
    SCB.PARTNER_HALF_SESSION_RQ_MODE = DELAYED;

                                                    /* SET UP RESPONSE MODE     */

  IF SCB.FM_PROFILE = (5 | 6 | 17) THEN
    DO;
    . SCB.THIS_HALF_SESSION_RSP_MODE = DELAYED;
    . SCB.PARTNER_HALF_SESSION_RSP_MODE = DELAYED;
    END;
  ELSE
    DO;
    . SCB.THIS_HALF_SESSION_RSP_MODE = IMMEDIATE;
    . SCB.PARTNER_HALF_SESSION_RSP_MODE = IMMEDIATE;
    END;

  NEWLIST CT_RCV_RQ_EXP ENTRY_NAME(CT_RCV_RQ_EXP_ENTRY);  /* CREATE...          */
  NEWLIST CT_RCV_RQ_NORM ENTRY_NAME(CT_NORM_ENTRY);       /* ...CORRELATION...  */
  NEWLIST CT_SEND_RQ_EXP ENTRY_NAME(CT_SEND_RQ_EXP_ENTRY); /* ...TABLES         */
  NEWLIST CT_SEND_RQ_NORM ENTRY_NAME(CT_NORM_ENTRY);

  RETURN;
END DFC_INIT_MISC_SESSION_PARMS;



UPM_FDX_BRACKETS: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THE FDX  WITHIN BRACKETS PROTOCOL IS  NOT USED WITH  ANY ARCHITECTED │
  │                 SESSIONS (SEE  SNA LU-LU  SESSION TYPES).   SEE APPROPRIATE  PRODUCT │
  │                 SPECIFICATION FOR SPECIFIC USAGE.                            │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 DFC_INIT_FSM_USAGE_BSM_SBI_RTR     PAGE 5-35                 │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

                                                    /* NOT ARCHITECTED          */

  RETURN;
END UPM_FDX_BRACKETS;
```

```
SESSACT.DFC_RESET: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO RESET  ALL DFC FSM'S AND  CORRELATION TABLES.  THIS  PROCEDURE IS │
  │                 CALLED: 1) BY  THE COMMON SESSION CONTROL  MANAGER (CSC_MGR, CHAPTER │
  │                 13)  ON  SENDING OR  RECEIVING  A  POSITIVE  RESPONSE TO  A  SESSION │
  │                 ACTIVATION REQUEST  AND 2) AS A  RESULT OF RESETTING A  SUBTREE THAT │
  │                 INCLUDES DFC.                                                │
  │                                                                              │
  │      INPUT:     RESET SIGNAL                                                 │
  │                                                                              │
  │      NOTE:      CSC_MGR.DFC_INITIALIZE HAS BEEN EXECUTED PRIOR TO THIS PROCEDURE. │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 DFC_RESET_HDX                    PAGE 5-39                   │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  CALL #FSM_CHAIN_RCV('RESET');                    /* PAGE 5-72                  */
  CALL #FSM_CHAIN_SEND('RESET');                   /* PAGE 5-72                  */
  CALL #FSM_CONTROL_BSM_RSP_RCV('RESET');          /* PAGE 5-73                  */
  CALL #FSM_CONTROL_BSM_RSP_SEND('RESET');         /* PAGE 5-74                  */
  CALL #FSM_CONTROL_HDX_RSP_RCV('RESET');          /* PAGE 5-75 OR 5-76 OR 5-77  */
  CALL #FSM_CONTROL_HDX_RSP_SEND('RESET');         /* PAGE 5-78 OR 5-79 OR 5-80  */
  CALL #FSM_EBCD_RCV('RESET');                     /* PAGE 5-81                  */
  CALL #FSM_EBCD_SEND('RESET');                    /* PAGE 5-81                  */
  CALL #FSM_IMM_RQ_MODE_RCV('RESET');              /* PAGE 5-86                  */
  CALL #FSM_IMM_RQ_MODE_SEND('RESET');             /* PAGE 5-86                  */
  CALL #FSM_QEC_RCV('RESET');                      /* PAGE 5-87                  */
  CALL #FSM_QEC_SEND('RESET');                     /* PAGE 5-87                  */
  CALL #FSM_QRI_CHECK_SEND('RESET');               /* PAGE 5-88                  */
  CALL #FSM_QRI_CHAIN_RCV('RESET');                /* PAGE 5-88                  */
  CALL #FSM_QRI_CHAIN_SEND('RESET');               /* PAGE 5-89                  */
  CALL #FSM_RES('RESET');                          /* PAGE 5-89                  */
  CALL #FSM_RTR('RESET');                          /* PAGE 5-90 OR 5-90          */
  CALL #FSM_SBI_RCV('RESET');                      /* PAGE 5-91                  */
  CALL #FSM_SBI_SEND('RESET');                     /* PAGE 5-91                  */
  CALL #FSM_SHUTD('RESET');                        /* PAGE 5-92 OR 5-92          */

  IF SCB.USING_BRACKETS = YES THEN
    DO;
    . IF SCB.BRACKETS_RESET_STATE = BETB THEN
    .   CALL #FSM_BSM('RESET_BETB');               /* PAGE 5-68 OR 5-70          */
    . ELSE
    .   CALL #FSM_BSM('RESET_INB');                /* PAGE 5-68 OR 5-70          */
    END;

  CALL DFC_RESET_HDX;                              /* RESET HDX FSMS (PAGE 5-39) */

  PURGE CT_RCV_RQ_EXP;                             /* RESET ...                  */
  PURGE CT_RCV_RQ_NORM;                            /* ... CORRELATION ...        */
  PURGE CT_SEND_RQ_EXP;                            /* ... TABLES                 */
  PURGE CT_SEND_RQ_NORM;

  SCB.SQN_SEND_CNT = 0;                            /* RESET SEND SEQUENCE COUNTER */

  RETURN;
END SESSACT.DFC_RESET;
```

```
DFC_RESET_HDX: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE RESETS THE HDX FSM'S.                         │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                SESSACT.DFC_RESET                  PAGE 5-38                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SELECT ANYORDER(SCB.SEND_RCV_MODE);
  . WHEN(HDX_CONTENTION)
  .   CALL #FSM_HDX('RESET_CONT');                    /* PAGE 5-82 TO 5-83      */
  . WHEN(HDX_FLIP_FLOP)
  .   DO;
  .   . IF SCB.USING_BRACKETS = YES THEN
  .   .   DO;
  .   .   . IF #FSM_BSM = BETB THEN                   /* PAGE 5-68 OR 5-70
  .   .   .                                              BSM RESET TO BETB STATE */
  .   .   .   CALL #FSM_HDX('RESET_CONT');            /* PAGE 5-84              */
  .   .   . ELSE                                      /* BSM IS RESET INB STATE */
  .   .   .   DO;
  .   .   .   . IF (SCB.HALF_SESSION = PRI & SCB.HDX_FF_RESET_STATE = SEND_FOR_PRI) |
  .   .   .   .   (SCB.HALF_SESSION = SEC & SCB.HDX_FF_RESET_STATE = SEND_FOR_SEC) THEN
  .   .   .   .   CALL #FSM_HDX('RESET_SEND');       /* PAGE 5-84              */
  .   .   .   . ELSE
  .   .   .   .   CALL #FSM_HDX('RESET_RCV');        /* PAGE 5-84              */
  .   .   .   END;
  .   .   END;
  .   . ELSE                                          /* NOT USING BRACKETS     */
  .   .   DO;
  .   .   . IF (SCB.HALF_SESSION = PRI & SCB.HDX_FF_RESET_STATE = SEND_FOR_PRI) |
  .   .   .   (SCB.HALF_SESSION = SEC & SCB.HDX_FF_RESET_STATE = SEND_FOR_SEC) THEN
  .   .   .   CALL #FSM_HDX('RESET_SEND');           /* PAGE 5-84              */
  .   .   . ELSE
  .   .   .   CALL #FSM_HDX('RESET_RCV');            /* PAGE 5-84              */
  .   .   END;
  .   END;
  . OTHERWISE;
  END;

  RETURN;
END DFC_RESET_HDX;
```

```
DEQUEUE.Q_TC_TO_DFC: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE  IS CALLED BY  THE DISPATCHER AS  A RESULT OF  A SEND  │
   │                 DONE  BY THE  SCHEDULER. ITS  FUNCTION IS  TO DEQUEUE  A REQUEST  OR  │
   │                 RESPONSE  FROM  Q_TC_TO_DFC  (IF ALLOWABLE)  AND  CALL  THE  DFC.RCV  │
   │                 PROCEDURE TO PROCESS IT.                             │
   │                                                                      │
   │      INPUT:     OPEN QUEUE SIGNAL FROM SCHEDULER                     │
   │                                                                      │
   │      NOTE:      IF THE BIDDER HAS  SENT A BID OR BB REQUEST  WHOSE RESPONSE WILL NOT  │
   │                 BE QUEUED (QRI=¬QR), NOTHING MAY BE  DEQUEUED UNTIL THE RESPONSE (TO  │
   │                 BID OR BB) IS RECEIVED.                              │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 FSM_HDX_CONT_LOSER                PAGE 5-82          │
   │                                                                      │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
   │                 FSM_BSM_BIDDER                    PAGE 5-68          │
   │                 FSM_HDX_CONT_LOSER                PAGE 5-82          │
   │                 FSM_HDX_FF                        PAGE 5-84          │
   │                 FSM_QRI_CHECK_SEND                PAGE 5-88          │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL DEQUEUE_ALLOWED BIT(1);                       /* LOCAL VARIABLE     */

  DEQUEUE_ALLOWED = NO;                             /* INITIALIZE TO NOT DEQUEUE */

  IF FIRST_ENTRY(SCB.Q_TC_TO_DFC)->RRI = RSP THEN   /* ALWAYS OK TO...    */
    DEQUEUE_ALLOWED = YES;                          /* ...DEQUEUE RESPONSE */
  ELSE
    IF SCB.SEND_RCV_MODE = FULL_DUPLEX THEN         /* ALWAYS OK TO DEQUEUE.  .. */
  DEQUEUE_ALLOWED = YES;                            /* ...WHEN USING FULL DUPLEX */
    ELSE                                            /* NOT FULL DUPLEX    */
      DO;
      . IF SCB.USING_BRACKETS = YES THEN            /* RUNNING WITH BRACKETS */
      .   DO;
      .   . IF SCB.FIRST_SPEAKER = YES THEN         /* BRACKETS FIRST SPEAKER MAY... */
      .   .   DEQUEUE_ALLOWED = YES;                /* ...ALWAYS DEQUEUE, SO DEADLOCK */
      .   .                                         /* ...CONDITIONS MAY BE PREVENTED */
      .   . ELSE                                    /* BIDDER             */
      .   .   DO;
      .   .   . IF #FSM_HDX = (*S,R) THEN           /* PAGE 5-82 TO 5-84
      .   .   .                                        BIDDER MAY DEQUEUE ONLY...
      .   .   .                                        ...WHEN IN RECEIVE STATE */
      .   .   .   DO;
      .   .   .   . IF FSM_QRI_CHECK_SEND = RESET &  /* RSP WILL NOT BE QUEUED AND... */
      .   .   .   .   (FSM_BSM_BIDDER = PEND_INB |   /* ...WAITING FOR BB RSP OR... */
      .   .   .   .   LAST_ENTRY(CT_SEND_RQ_NORM)->CT_DFC_RQ_CODE = BID) THEN
      .   .   .   .                                 /* ...WAITING FOR BID RSP THEN... */
      .   .   .   .   ;                             /* ...BIDDER MAY NOT DEQUEUE */
      .   .   .   .                                 /* SEE NOTE IN PROLOGUE */
      .   .   .   .  ELSE                           /* OTHERWISE BIDDER MAY DEQUEUE */
      .   .   .   . DEQUEUE_ALLOWED = YES;
      .   .   .   END;
      .   .   END;
      .   END;
      . ELSE                                        /* NOT USING BRACKETS */
      .   DO;
      .   . IF SCB.SEND_RCV_MODE = HDX_CONTENTION THEN
      .   .   DO;
      .   .   . IF (SCB.HALF_SESSION = PRI & SCB.CONT_WIN = PRI) |      /* ALWAYS OK FOR... */
      .   .   .   (SCB.HALF_SESSION = SEC & SCB.CONT_WIN = SEC) THEN     /* CONTENTION... */
      .   .   .   DEQUEUE_ALLOWED = YES;           /* ...WINNER TO DEQUEUE, SO ... */
      .   .   .                                     /* ...DEADLOCKS CAN BE PREVENTED */
      .   .   . ELSE                                /* CONTENTION LOSER   */
      .   .   .   IF FSM_HDX_CONT_LOSER = (*S,R) THEN     /*  PAGE 5-82
      .   .   .                                        MAY DEQUEUE ONLY WHEN... */
      .   .   .     DEQUEUE_ALLOWED = YES;          /* ...IN RECEIVE STATE */
      .   .   END;
      .   . ELSE                                    /* HDX FLIP FLOP      */
      .   .   IF FSM_HDX_FF = (*S,R) THEN           /* PAGE 5-84
      .   .                                           MAY DEQUEUE ONLY WHEN... */
      .   .     DEQUEUE_ALLOWED = YES;              /* ...IN RECEIVE STATE */
      .   END;
      END;

  IF DEQUEUE_ALLOWED = YES THEN
    DO;
    . REMOVE MU FROM SCB.Q_TC_TO_DFC;               /* DEQUEUE RQ OR RSPO*/
    . SEND MU TO DFC.RCV;                           /* PAGE 5-50          */
    END;

  RETURN;                                           /* RETURN TO DISPATCHER */
END DEQUEUE.Q_TC_TO_DFC;
```

```
DFC.SEND: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  ENFORCES DATA FLOW CONTROL PROTOCOL FOR SENDING REQUESTS AND
   │                RESPONSES                                             │
   │                                                                      │
   │     INPUT:                                                           │
   │                1)  REQUESTS FROM FMDS CONTAIN THE FOLLOWING INFORMATION: RRI=RQ,
   │                    EFI, RU_CTGY=FMD|DFC, FI, SDI, BCI, ECI, DR1I, DR2I, ERI, QRI,
   │                    BBI, EBI, CDI, CSI, EDI, PDI, RU                  │
   │                                                                      │
   │                2)  RESPONSES FROM FMDS--THE RULES FOR SENDING RESPONSES DEPEND
   │                    UPON THE TYPE OF RESPONSE ASKED FOR BY THE CHAIN:  │
   │                                                                      │
   │                    •  NO-RESPONSE CHAINS REQUIRE NO RESPONSES.       │
   │                                                                      │
   │                    •  EXCEPTION OR DEFINITE RESPONSE CHAINS REQUIRE EITHER (A) A
   │                       NEGATIVE RESPONSE TO ONE RU IN THE CHAIN OR (B) A POSITIVE
   │                       RESPONSE TO THE LAST RU IN THE CHAIN (IN THE CASE OF
   │                       EXCEPTION RESPONSE CHAINS THIS MEANS NO NEGATIVE RESPONSES
   │                       WILL BE FORTHCOMING). A POSITIVE RESPONSE TO AN
   │                       EXCEPTION-RESPONSE REQUEST CHAIN IS DISCARDED BY DFC.SEND,
   │                       RATHER THAN EMITTED AS OUTPUT. IT IS USED TO CLEAN UP THE
   │                       CORRELATION TABLE.                             │
   │                                                                      │
   │                    RESPONSES FROM FMDS CONTAIN THE FOLLOWING INFORMATION: RRI=RSP,
   │                    EFI, SNF, RU_CTGY, FI, SDI, BCI, ECI, DR1I, DR2I, RTI, QRI, RU
   │                                                                      │
   │     OUTPUT:                                                          │
   │                1)  REQUESTS PASSED TO CPMGR.SEND CONTAIN THE SAME INFORMATION AS
   │                    DESCRIBED FOR REQUESTS UNDER INPUT, WITH THE ADDITION OF THE
   │                    SNF.                                              │
   │                                                                      │
   │                2)  RESPONSES PASSED TO CPMGR.SEND CONTAIN THE FOLLOWING
   │                    INFORMATION: RRI=RSP, EFI, SNF, FI, SDI, BCI, ECI, DR1I, DR2I,
   │                    RTI, QRI, RU                                      │
   │                                                                      │
   │                3)  A REJECT WITH THE SENSE CODE INDICATING THE TYPE OF ERROR IS
   │                    RETURNED TO THE SENDING PROCEDURE IF AN ERROR IS DETECTED.
   │                                                                      │
   │                4)  THE SEQUENCE NUMBER FIELD (SNF) ASSIGNED IS SENT TO SENDING
   │                    PROCEDURE                                         │
   │                                                                      │
   │                5)  A BETB SIGNAL IS SENT TO THE SENDING PROCEDURE TO INDICATE
   │                    BETWEEN BRACKETS CONDITION                        │
   │                                                                      │
   │     NOTE:      DFC_SEND ASSUMES HDX CONTENTION LOSERS AND BRACKET BIDDERS HAVE A
   │                QUEUE (Q_TC_TO_DFC).                                  │
   │                                                                      │
   │     REFERS TO THE FOLLOWING PROCEDURE(S):                            │
   │                BETWEEN_BRACKETS_CONDITION          PAGE 5-58         │
   │                DFC.SEND_CHECKS                     PAGE 5-42         │
   │                SEND_CT_CLEANUP                     PAGE 5-49         │
   │                SEND_CT_INITIALIZE                  PAGE 5-46         │
   │                SEND_DISCARD_CHECKS                 PAGE 5-47         │
   │                SEND_FSMS                           PAGE 5-48         │
   │                SEND_SNF_ASSIGN                     PAGE 5-45         │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  IF TC.CPMGR.SEND_CHECKS = NG |                          /* CHAPTER 4      */
    DFC.SEND_CHECKS = NG THEN                             /* PAGE 5-42      */
                                                          /* THESE CHECKS NEED NOT BE DONE IF
                                                             ALREADY DONE IN A HIGHER LAYER */
    SEND SEND_CHECK(SEND_CHECK_SENSE) TO SENDING_PROCEDURE; /* SEND_CHECK_SENSE CONTAINS SENSE
                                                             BYTES INDICATING TYPE OF ERROR */
  ELSE
    DO;
    . IF RRI = RQ THEN                                    /* ASSIGN SEQUENCE NUMBER... */
    .   DO;                                               /* ...FIELD FOR REQUESTS     */
    .   . CALL SEND_SNF_ASSIGN;                           /* PAGE 5-45                 */
    .   . SEND MU TO SENDING_PROCEDURE;                   /* SEND BACK ASSIGNED SNF VALUE */
    .   END;
    .
    . CALL SEND_CT_INITIALIZE;                            /* INITIALIZE CORRELATION TABLE
    .                                                        PAGE 5-46          */
    .
    . IF SEND_DISCARD_CHECKS = DO_NOT_DISCARD THEN        /* PAGE 5-47          */
    .   DO;
    .   . CALL SEND_FSMS;                                 /* PAGE 5-48          */
    .   . SEND MU TO CPMGR.SEND;                          /* CHAPTER 4          */
    .   .
    .   . IF BETWEEN_BRACKETS_CONDITION = YES THEN        /* PAGE 5-58          */
    .   .   SEND 'BETB' TO SENDING_PROCEDURE;
    .   END;
    . ELSE
    .   DISCARD MU;
    .
    . CALL SEND_CT_CLEANUP;                               /* CLEANUP CORRELATION TABLE
    .                                                        PAGE 5-49          */
    END;

  RETURN;                                                 /* RETURN TO DISPATCHER */
END DFC.SEND;
```

```
DFC.SEND_CHECKS: PROCEDURE RETURNS(BIT(1));

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO PERFORM ALL DFC SEND ERROR CHECKS.                │
 │                                                                      │
 │      OUTPUT:    RETURN CODE  (RC) = NO  GOOD (NG) IF  AN ERROR IS  FOUND; OTHERWISE,
 │                 RC=OK.                                               │
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 DFC.SEND                        PAGE 5-41            │
 │                                                                      │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
 │                 CT_KEY_SEARCH                   PAGE 5-60            │
 │                 RESPONSES_OWED                  PAGE 5-43            │
 │                 SEND_RSP_SENSE_CKS              PAGE 5-44            │
 │                 USAGE_CHECKS                    PAGE 5-61            │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
 DCL RC BIT(1);

 RC = OK;
 SELECT;
 . WHEN(EFI = NORMAL & RRI = RQ)
 .   DO;
 .   . IF USAGE_CHECKS = NG |                          /* PAGE 5-61                    */
 .   .    SEND_OR_RECEIVE_CHECK(#FSM_HDX) |            /* PAGE 5-82 TO 5-84            */
 .   .    SEND_OR_RECEIVE_CHECK(#FSM_QEC_RCV) |        /* PAGE 5-87                    */
 .   .    SEND_OR_RECEIVE_CHECK(#FSM_SHUTD) |          /* PAGE 5-92 OR 5-92            */
 .   .    RESPONSES_OWED = YES |                       /* PAGE 5-43                    */
 .   .    SEND_OR_RECEIVE_CHECK(#FSM_IMM_RQ_MODE_SEND) |  /* PAGE 5-86                */
 .   .    SEND_OR_RECEIVE_CHECK(#FSM_CHAIN_SEND) |     /* PAGE 5-72                    */
 .   .    SEND_OR_RECEIVE_CHECK(#FSM_BSM) |            /* PAGE 5-68 OR 5-70            */
 .   .    SEND_OR_RECEIVE_CHECK(#FSM_CONTROL_HDX_RSP_RCV) |
 .   .                                                /* PAGE 5-75 OR 5-76 OR 5-77    */
 .   .    SEND_OR_RECEIVE_CHECK(#FSM_CONTROL_HDX_RSP_SEND) |
 .   .                                                /* PAGE 5-78 OR 5-79 OR 5-80    */
 .   .    SEND_OR_RECEIVE_CHECK(#FSM_SBI_RCV) |        /* PAGE 5-91                    */
 .   .    SEND_OR_RECEIVE_CHECK(#FSM_EBCD_SEND) |      /* PAGE 5-81                    */
 .   .    SEND_OR_RECEIVE_CHECK(#FSM_RTR) |            /* PAGE 5-90 OR 5-90            */
 .   .    SEND_OR_RECEIVE_CHECK(#FSM_QRI_CHECK_SEND) | /* PAGE 5-88                    */
 .   .    SEND_OR_RECEIVE_CHECK(#FSM_QRI_CHAIN_SEND) | /* PAGE 5-89                    */
 .   .   (SCB.SEND_RCV_MODE = HDX_CONTENTION &
 .   .   QRI =  ¬QR &
 .   .   ((BBI =  ¬BB & EBI = EB) |
 .   .   #FSM_BSM = PEND_TERM_S)) THEN                 /* PAGE 5-68 OR 5-70            */
 .   .   RC = NG;                                      /* ERROR FOUND                  */
 .   END;
 .
 . WHEN(EFI = NORMAL & RRI = RSP)
 .   DO;
 .   . CT_PTR = CT_RCV_RQ_NORM;                        /* SET PTR TO CORRELATION
 .   .                                                   TABLE TO BE USED             */
 .   . SCB.KEY = SNF;                                  /* SEARCH CORRELATION TABLE
 .   .                                                   FOR ENTRY WITH THIS KEY      */
 .   . IF CT_KEY_SEARCH = FOUND THEN                   /* DOES A REQUEST EXIST IN
 .   .                                                   CORRELATION TABLE FOR THIS
 .   .                                                   RESPONSE (PAGE 5-60)         */
 .   .   DO;                                           /* YES                          */
 .   .   . SELECT;
 .   .   . . WHEN(USAGE_CHECKS = NG)                   /* PAGE 5-61                    */
 .   .   . .   RC = NG;                                /* RSP RH NOT FORMATTED CORRECTLY */
 .   .   . . WHEN(¬(RU_CTGY = DFC & RQ_CODE = CANCEL) & CT_RSP_TO_NOT_CANCEL = SENT)
 .   .   . .   RC = NG;                                /* ALREADY SENT RSP TO THIS CHAIN */
 .   .   . . WHEN(SEND_RSP_SENSE_CKS = NG)             /* PAGE 5-44                    */
 .   .   . .   RC = NG;                                /* SENSE BYTES INCORRECT        */
 .   .   . . WHEN(SCB.THIS_HALF_SESSION_RSP_MODE = IMMEDIATE &
 .   .   . .     CT_NORM_ENTRY_PTR ¬= FIRST_ENTRY(CT_RCV_RQ_NORM))
 .   .   . .   RC = NG;                                /* SEND RESPONSES IN ORDER WHEN... */
 .   .   . .                                           /* ...USING IMMMEDIATE RSP MODE */
 .   .   . . WHEN(SCB.THIS_HALF_SESSION_RSP_MODE = DELAYED &
 .   .   . .     RU_CTGY = DFC & RQ_CODE = CHASE &
 .   .   . .     CT_NORM_ENTRY_PTR ¬= FIRST_ENTRY(CT_RCV_RQ_NORM))
 .   .   . .   RC = NG;                                /* RSP TO CHASE MUST NOT BE SENT
 .   .   . .                                             BEFORE RSPS TO ALL RQS RECEIVED
 .   .   . .                                             PRIOR TO CHASE ARE SENT      */
 .   .   . . OTHERWISE;
 .   .   . END;
 .   .   END;
 .   . ELSE                                            /* RSP NOT IN CORRELATION TABLE */
 .   .   RC = NG;
 .   END;
```

```
  . WHEN(EFI = EXP & RRI = RQ)
  .   DO;
  .   . IF USAGE_CHECKS = NG |                                  /* PAGE 5-61                    */
  .   .    SEND_OR_RECEIVE_CHECK(@FSM_QEC_SEND) |              /* PAGE 5-87                    */
  .   .    SEND_OR_RECEIVE_CHECK(@FSM_SHUTD) |                 /* PAGE 5-92 OR 5-92            */
  .   .    SEND_OR_RECEIVE_CHECK(@FSM_SBI_SEND) THEN           /* PAGE 5-91                    */
  .   .   RC = NG;
  .   END;
  .
  . WHEN(EFI = EXP & RRI = RSP)
  .   DO;
  .   . RC = NG;
  .   . SCAN CT_RCV_RQ_EXP PTR(CT_RCV_RQ_EXP_ENTRY_PTR) WHILE(RC = NG);
  .   . . IF CT_RCV_RQ_EXP_ID = SNF THEN
  .   . .   RC = OK;
  .   . SCANEND;
  .   .
  .   . IF RC = NG |                                           /* ENTRY NOT FOUND OR...        */
  .   .   USAGE_CHECKS = NG THEN                               /* ...FORMAT ERROR(PAGE 5-61)   */
  .   .   RC = NG;
  .   END;
  END;

  RETURN(RC);
END DFC.SEND_CHECKS;



RESPONSES_OWED: PROCEDURE RETURNS(BIT(1));

                                                                                              /*
  ┌──────────────────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO TEST IF,  IN HALF-DUPLEX SEND/RECEIVE MODE,  THERE ARE PREVIOUSLY     │
  │                 RECEIVED REQUESTS  THAT HAVE NOT BEEN  RESPONDED TO, AND, IF  SO, TO     │
  │                 SET SENSE CODE 200D.                                                     │
  │                                                                                          │
  │      OUTPUT:    RC  =  YES  IF HALF-DUPLEX  MODE  AND CT_RCV_RQ_NORM  IS NOT  EMPTY;     │
  │                 OTHERWISE, RC = NO.                                                      │
  │                                                                                          │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                           │
  │                 DFC.SEND_CHECKS                    PAGE 5-42                             │
  └──────────────────────────────────────────────────────────────────────────────────────────┘
                                                                                              */
  DCL RC BIT(1);

  RC = NO;
  IF SCB.SEND_RCV_MODE = (HDX_CONTENTION|HDX_FLIP_FLOP) THEN
    IF ¬EMPTY(CT_RCV_RQ_NORM) THEN                            /* ANY RQ IN RCV CORRELATION TABLE? */
      DO;                                                     /* RESPONSE(S) OWED */
      . RC = YES;
      . SEND_CHECK_SENSE = X'200D';
      END;

  RETURN(RC);
END RESPONSES_OWED;
```

```
SEND_RSP_SENSE_CKS: PROCEDURE RETURNS(BIT(1));

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO MAKE  SURE RESPONSES TO  EXR'S (EXCEPTION REQUESTS)  ARE NEGATIVE
   │                 RESPONSES WITH THE CORRECT SENSE.  IF  THE RECEIVED REQUEST WAS SENT
   │                 TO THE LAYER ABOVE DFC AS AN EXR, THEN:
   │
   │                 •    THE RESPONSE TO THE EXR IS A NEGATIVE RESPONSE AND
   │
   │                 •    THE SENSE BYTES  ON THE NEGATIVE RESPONSE (TO THE  EXR) ARE THE
   │                      SAME AS THE SENSE BYTES THAT WERE SPECIFIED ON THE EXR.
   │
   │                 ONE  EXCEPTION  IS THE  EXR  WITH  SENSE  BYTES 0813  (BRACKET  RACE
   │                 ERROR--RTR NOT FORTHCOMING).  THE -RSP TO THIS EXR MAY CONTAIN SENSE
   │                 BYTES 0813 OR 0814 (BRACKET RACE ERROR--RTR FORTHCOMING).
   │
   │      NOTE:      MORE ENFORCEMENT OF SENSE CODES IS OPTIONAL IN THIS PROCEDURE; E.G.,
   │                 IT MAY ENFORCE  THAT RACE ERRORS (080B, 0813, 0814,  081B, 0846) ARE
   │                 SENT ONLY WHEN THE REQUEST HAS BEEN CONVERTED TO AN EXR BY DFC.
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 DFC.SEND_CHECKS                    PAGE 5-42
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RC BIT(1);

  RC = OK;                                          /* INITIALIZE RETURN CODE TO OK    */

  IF RU_CTGY = DFC & RQ_CODE = CANCEL THEN          /* RSP TO CANCEL                   */
    DO;
    . IF CT_EXR_SENSE_FOR_CANCEL ¬= 0 THEN          /* CANCEL WAS AN EXR               */
    .   DO;
    .   . IF SDI =  ¬SD |                           /* POSITIVE RESPONSE OR...         */
    .   .   CT_EXR_SENSE_FOR_CANCEL(0:15) ¬= SNC(0:15) THEN
    .   .   RC = NG;                                /* ...SENSE BYTES ON RSP ARE...    */
    .   .                                           /* ...NOT THE SAME AS IN THE EXR   */
    .   .
    .   END;
    END;
  ELSE                                              /* RSP TO NOT CANCEL               */
    DO;
    . IF CT_EXR_SENSE_FOR_NOT_CANCEL ¬= 0 THEN      /* RQ WAS AN EXR                   */
    .   DO;
    .   . IF SDI =  ¬SD THEN                        /* POSITIVE RESPONSE               */
    .   .   RC = NG;
    .   . ELSE                                      /* NEGATIVE RSP                    */
    .   .   DO;
    .   .   . IF CT_EXR_SENSE_FOR_NOT_CANCEL = X'0813' THEN /* 0813 SENSE MAY...     */
    .   .   .   IF SNC(0:15) ¬= X'0813' & SNC(0:15) ¬= X'0814' THEN
    .   .   .                                       /* ...BE OVERRIDDEN WITH 0814      */
    .   .   .     RC = NG;
    .   .   .   ELSE                                /* EXR SENSE NOT 0813              */
    .   .   .     IF CT_EXR_SENSE_FOR_NOT_CANCEL(0:15) ¬= SNC(0:15) THEN
    .   .   .                                       /* SENSE IN RSP NOT THE...         */
    .   .   .       RC = NG;                        /* ...SAME AS IN EXR               */
    .   .   END;
    .   END;
    END;

  RETURN(RC);
END SEND_RSP_SENSE_CKS;
```

```
SEND_SNF_ASSIGN: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO ASSIGN THE SEQUENCE NUMBER OR ID TO THE REQUEST.          │
  │                                                                              │
  │      OUTPUT:    THE SNF FIELD  IN THE REQUEST CONTAINS THE  ASSIGNED SEQUENCE NUMBER
  │                 OR ID.                                                       │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 DFC.SEND                          PAGE 5-41                  │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 UPM_ID_ASSIGN_EXP                 PAGE 5-46                  │
  │                 UPM_ID_ASSIGN_NORM                PAGE 5-45                  │
  │                 UPM_SQN_ASSIGN_NORM               PAGE 5-45                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  SELECT ANYORDER(EFI);
  . WHEN(NORMAL)
  .   SELECT ANYORDER(SCB.SQN_USAGE);
  .   . WHEN(SEQUENCE_NUMBERS)
  .   .   DO;
  .   .   . SCB.SQN_SEND_CNT = SCB.SQN_SEND_CNT + 1;      /* INCREMENT 2-BYTE SEQUENCE
  .   .   .                                                   VARIABLE          */
  .   .   . SNF = SCB.SQN_SEND_CNT;                       /* ASSIGN SQN TO REQUEST */
  .   .   END;
  .   . WHEN(IDENTIFIERS)
  .   .   CALL UPM_ID_ASSIGN_NORM;                        /* PAGE 5-45             */
  .   . WHEN(NO_SNF)                                      /* THIS IS FOR FID3 CASE */
  .   .   CALL UPM_SQN_ASSIGN_NORM;                       /* PAGE 5-45             */
  .   END;
  .
  . WHEN(EXP)
  .   CALL UPM_ID_ASSIGN_EXP;                            /* PAGE 5-46             */
  END;

  RETURN;
END SEND_SNF_ASSIGN;



UPM_ID_ASSIGN_NORM: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS UPM ASSIGNS ID'S FOR NORMAL-FLOW REQUESTS.              │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 SEND_SNF_ASSIGN                   PAGE 5-45                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

                                                        /* NOT APCHITECTED       */

  RETURN;
END UPM_ID_ASSIGN_NORM;



UPM_SQN_ASSIGN_NORM: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS UPM  HANDLES CORRELATION TABLE SEQUENCE  NUMBERS FOR PU  TYPE 1
  │                 NODES (FID3 TH).                                             │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 SEND_SNF_ASSIGN                   PAGE 5-45                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

                                                        /* NOT ARCHITECTED       */

  RETURN;
END UPM_SQN_ASSIGN_NORM;
```

```
UPM_ID_ASSIGN_EXP: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM ASSIGNS ID'S FOR EXPEDITED-FLOW REQUESTS.           │
   │                                                                              │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
   │                 SEND_SNF_ASSIGN                  PAGE 5-45                   │
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

                                                 /* NOT ARCHITECTED             */

  RETURN;
END UPM_ID_ASSIGN_EXP;



SEND_CT_INITIALIZE: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO INITIALIZE THE CORRELATION TABLE.                         │
   │                                                                              │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
   │                 DFC.SEND                         PAGE 5-41                   │
   │                                                                              │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
   │                 CT_ENTRY_ADD_OR_UPDATE           PAGE 5-59                   │
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  SELECT ANYORDER;
  . WHEN(EFI = NORMAL & RRI = RQ)
  .   DO;
  .   . CT_PTR = CT_SEND_RQ_NORM;                  /* SET UP CORRELATION TABLE
  .   .                                               TO BE USED               */
  .   . CALL CT_ENTRY_ADD_OR_UPDATE;               /* ADD OR UPDATE ENTRY IN
  .   .                              CORRELATION TABLE(PAGE 5-59)               */
  .   END;
  .
  . WHEN(EFI = NORMAL & RRI = RSP)
  .   ;
  .
  . WHEN(EFI = EXP & RRI = RQ)
  .   DO;
  .   . CREATE CT_SEND_RQ_EXP_ENTRY;
  .   .
  .   . CT_SEND_RQ_EXP_ID = SNF;
  .   . CT_SEND_RQ_EXP_DFC_RQ_CODE = RQ_CODE;
  .   .
  .   . INSERT CT_SEND_RQ_EXP_ENTRY IN CT_SEND_RQ_EXP;
  .   END;
  .
  . WHEN(EFI = EXP & RRI = RSP)
  .   ;
  END;

  RETURN;
END SEND_CT_INITIALIZE;
```

```
SEND_DISCARD_CHECKS: PROCEDURE RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO DETERMINE WHEN A RQ|RSP IS TO BE DISCARDED.               │
  │                                                                              │
  │      OUTPUT:    A  RETURN CODE  IS  SET  TO DO_DISCARD  WHEN  THE  RQ|RSP IS  TO  BE  │
  │                 DISCARDED. OTHERWISE THE RETURN CODE IS SET TO DO_NOT_DISCARD.│
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 DFC.SEND                                PAGE 5-41            │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL RC BIT(1);

  RC = DO_NOT_DISCARD;
  SELECT ANYORDER;
  . WHEN(EFI = NORMAL & RRI = RQ)
  .   ;                                                 /* NO DISCARD CONDITIONS          */
  .
  .    WHEN(EFI = NORMAL & RRI = RSP)
  .     DO;
  .     . IF RTI = POS &                                /* POSITIVE RESPONSE AND...       */
  .     .   CT_ERI = ER & (CT_DR1I = DR1 | CT_DR2I = DR2) THEN /* ...RQE REQUEST           */
  .     .   RC = DO_DISCARD;
  .     .                                               /* NOTE: THIS CHECK NEED NOT BE
  .     .                                                  MADE IF BOUND TO RECEIVE RQD
  .     .                                                  CHAINS ONLY                    */
  .     END;
  .
  . WHEN(EFI = EXP & RRI = RQ)
  .   ;                                                 /* NO DISCARD CONDITIONS          */
  .
  .    WHEN(EFI = EXP & RRI = RSP)
  .      ;                                              /* NO DISCARD CONDITIONS          */
  END;

  RETURN(RC);
END SEND_DISCARD_CHECKS;
```

```
SEND_FSMS: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO UPDATE ALL FSMS HAVING SEND RQ OR SEND RSP INPUTS.│
   │                                                                      │
   │      NOTE:      THE ORDER OF CALLS IS SIGNIFICANT  FOR THE FOLLOWING FSMS, WHICH ARE
   │                 CALLED IN THE ORDER LISTED:                          │
   │                                                                      │
   │                 •    #FSM_HDX                                        │
   │                 •    #FSM_BSM                                        │
   │                 •    #FSM_CONTROL_BSM_RSP_RCV                        │
   │                 •    #FSM_CONTROL_HDX_RSP_RCV                        │
   │                 •    #FSM_CONTROL_HDX_RSP_SEND                       │
   │                                                                      │
   │                 THE REASON FOR THE CALLS HAVING THIS ORDER IS, THAT THE FSM'S CALLED
   │                 LATER MAY CAUSE ADDITIONAL STATE CHANGES TO OCCUR IN THE FSMS CALLED
   │                 EARLIER. FOR EXAMPLE, IT IS POSSIBLE  FOR TWO STATE CHANGES TO OCCUR
   │                 IN  #FSM_BSM.  THE  FIRST  ONE OCCURS  WHEN  #FSM_BSM  IS CALLED  TO
   │                 PROCESS  THE  REQUEST  BEING  SENT.  THE  SECOND  ONE  OCCURS  WHEN
   │                 #FSM_CONTROL_BSM_RSP_RCV, ALSO  PROCESSING THE  REQUEST BEING  SENT,
   │                 DETECTS AN EC AND CALLS #FSM_BSM  WITH A NEGATIVE RESPONSE AS INPUT.
   │                 THIS NEGATIVE RESPONSE  COULD CAUSE #FSM_BSM TO MAKE  A SECOND STATE
   │                 CHANGE.                                              │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 DFC.SEND                          PAGE 5-41          │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  SELECT ANYORDER;
  . WHEN(EFI = NORMAL & RRI = RQ)
  .   DO;
  .   . CALL #FSM_HDX;                          /* PAGE 5-82 TO 5-84      */
  .   . CALL #FSM_QEC_RCV;                      /* PAGE 5-87              */
  .   . CALL #FSM_SHUTD;                        /* PAGE 5-92 OR 5-92      */
  .   . CALL #FSM_IMM_RQ_MODE_SEND;             /* PAGE 5-86              */
  .   . CALL #FSM_CHAIN_SEND;                   /* PAGE 5-72              */
  .   . CALL #FSM_BSM;                          /* PAGE 5-68 OR 5-70      */
  .   . CALL #FSM_CONTROL_BSM_RSP_RCV;          /* PAGE 5-73              */
  .   . CALL #FSM_CONTROL_HDX_RSP_RCV;          /* PAGE 5-75 OR 5-76 OR 5-77 */
  .   . CALL #FSM_CONTROL_HDX_RSP_SEND;         /* PAGE 5-78 OR 5-79 OR 5-80 */
  .   . CALL #FSM_SBI_RCV;                      /* PAGE 5-91              */
  .   . CALL #FSM_EBCD_SEND;                    /* PAGE 5-81              */
  .   . CALL #FSM_RTR;                          /* PAGE 5-90 OR 5-90      */
  .   . CALL #FSM_QRI_CHECK_SEND;               /* PAGE 5-88              */
  .   . CALL #FSM_QRI_CHAIN_SEND;               /* PAGE 5-89              */
  .   END;
  .
  . WHEN(EFI = NORMAL & RRI = RSP)
  .   DO;
  .   . CALL #FSM_QEC_SEND;                     /* PAGE 5-87              */
  .   . CALL #FSM_IMM_RQ_MODE_RCV;              /* PAGE 5-86              */
  .   . CALL #FSM_CHAIN_RCV;                    /* PAGE 5-72              */
  .   . CALL #FSM_CONTROL_BSM_RSP_SEND;         /* PAGE 5-74              */
  .   . CALL #FSM_CONTROL_HDX_RSP_SEND;         /* PAGE 5-78 OR 5-79 OR 5-80 */
  .   . CALL #FSM_SBI_SEND;                     /* PAGE 5-91              */
  .   . CALL #FSM_RTR;                          /* PAGE 5-90 OR 5-90      */
  .   END;
  .
  . WHEN(EFI = EXP & RRI = RQ)
  .   DO;
  .   . CALL #FSM_QEC_SEND;                     /* PAGE 5-87              */
  .   . CALL #FSM_SHUTD;                        /* PAGE 5-92 OR 5-92      */
  .   . CALL #FSM_SBI_SEND;                     /* PAGE 5-91              */
  .   END;
  .
  . WHEN(EFI = EXP & RRI = RSP)
  .   DO;
  .   . CALL #FSM_QEC_RCV;                      /* PAGE 5-87              */
  .   . CALL #FSM_SHUTD;                        /* PAGE 5-92 OR 5-92      */
  .   . CALL #FSM_SBI_RCV;                      /* PAGE 5-91              */
  .   END;
  END;

  RETURN;
END SEND_FSMS;
```

```
SEND_CT_CLEANUP: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO CLEAN UP CORRELATION TABLE.                           │
  │                                                                          │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
  │                 DFC.SEND                          PAGE 5-41              │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                                */

  SELECT ANYORDER;
  . WHEN(EFI = NORMAL & RRI = RQ)
  .   DO;
  .   . IF  ¬RQN THEN
  .   .   DO;
  .   .   . IF CT_ENTRY_TYPE = WHOLE_CHAIN_NO_CANCEL &
  .   .   .   CT_RSP_TO_NOT_CANCEL = RECEIVED THEN
  .   .   .   REMOVE CT_NORM_ENTRY FROM CT_SEND_RQ_NORM DISCARD;
  .   .   END;
  .   .
  .   . IF EMPTY(CT_SEND_RQ_NORM) THEN
  .   .   CALL #FSM_QRI_CHECK_SEND('NO_OUTSTANDING_RQS');  /* PAGE 5-88          */
  .   END;
  .
  . WHEN(EFI = NORMAL & RRI = RSP)
  .   SELECT ANYORDER(CT_ENTRY_TYPE);
  .   . WHEN(PARTIAL_CHAIN)
  .   .   CT_RSP_TO_NOT_CANCEL = SENT;
  .   . WHEN(WHOLE_CHAIN_NO_CANCEL)
  .   .   REMOVE CT_NORM_ENTRY FROM CT_RCV_RQ_NORM DISCARD;
  .   . WHEN(WHOLE_CHAIN_WITH_CANCEL)
  .   .   DO;
  .   .   . IF RU_CTGY = DFC & RQ_CODE = CANCEL THEN
  .   .   .   REMOVE CT_NORM_ENTRY FROM CT_RCV_RQ_NORM DISCARD;
  .   .   . ELSE
  .   .   .   CT_RSP_TO_NOT_CANCEL = SENT;
  .   .   END;
  .   . WHEN(CANCEL_ONLY)
  .   .   REMOVE CT_NORM_ENTRY FROM CT_RCV_RQ_NORM DISCARD;
  .   END;
  .
  . WHEN(EFI = EXP & RRI = RQ)
  .   ;
  .
  . WHEN(EFI = EXP & RRI = RSP)
  .   REMOVE CT_RCV_RQ_EXP_ENTRY FROM CT_RCV_RQ_EXP DISCARD;
  END;

  RETURN;
END SEND_CT_CLEANUP;
```

```
DFC.RCV: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO ENFORCE PROPER DATA FLOW CONTROL PROTOCOLS FOR RECEIVED REQUESTS
 │                 AND RESPONSES.
 │
 │      INPUT:
 │                 1)   REQUESTS FROM CPMGR.RCV CONTAIN FOLLOWING INFORMATION: RRI=RQ,
 │                      EFI, SQN|ID, RU_CTGY, FI, SDI, BCI, ECI, DR1I, DR2I, ERI, QRI,
 │                      BBI, EBI, CDI, CSI, EDI, PDI, RU (IN ITS ENTIRETY).
 │
 │                 2)   RESPONSES FROM CPMGR.RCV CONTAIN FOLLOWING INFORMATION:
 │                      RRI=RSP, EFI, SQN|ID, RU_CTGY, FI, SDI, BCI, ECI, DR1I, DR2I,
 │                      RTI, QRI, RU (IN ITS ENTIRETY)
 │
 │      OUTPUT:
 │                 1)   REQUESTS AND RESPONSES TO FMDS.RCV CONTAIN INFORMATION AS
 │                      SPECIFIED FOR INPUT, ABOVE.
 │
 │                 2)   A BETB SIGNAL IS SENT TO FMDS.RCV TO INDICATE A BETWEEN
 │                      BRACKETS CONDITION.
 │
 │      NOTE:      DFC_RCV ASSUMES THE FOLLOWING:
 │
 │                 • HDX CONTENTION LOSERS AND BRACKET BIDDERS HAVE A QUEUE
 │                   (Q_TC_TO_DFC).
 │
 │                 • SEQUENCE NUMBERS FOR PU_T1 (FID3 TH) ARE MANAGED INTERNALLY SO AS
 │                   TO LOOK THE SAME AS OTHER PU (FID) TYPES.
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 FSM_HDX_CONT_LOSER                  PAGE 5-82
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 BETWEEN_BRACKETS_CONDITION          PAGE 5-58
 │                 RCV_CHECKS                          PAGE 5-53
 │                 RCV_CT_CLEANUP                      PAGE 5-56
 │                 RCV_CT_INITIALIZE                   PAGE 5-52
 │                 RCV_DISCARD_CHECKS                  PAGE 5-54
 │                 RCV_FORMAT                          PAGE 5-51
 │                 RCV_FSMS                            PAGE 5-55
 │                 UPM_RECEIVE_CHECKS_PROCESS          PAGE 5-57
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL DISCARD_SW BIT(1);

  CALL RCV_FORMAT;                                      /* FORMAT INPUT IF NECESSARY.
                                                           PAGE 5-51                  */
  CALL RCV_CT_INITIALIZE;                               /* INITIALIZE CORRELATION TABLE
                                                           (PAGE 5-52)                */
  IF RCV_CHECKS = OK THEN                               /* CHECK FOR RECEIVE ERROR
                                                           CONDITIONS. THESE CHECKS ARE
                                                           OPTIONAL. (PAGE 5-53)      */
    DO;
    . IF BCI = BC | ECI = EC THEN                       /* BEGIN CHAIN OR END CHAIN   */
    .   CALL RCV_FSMS;                                  /* FINITE-STATE MACHINES
    .                                                      (PAGE 5-55)                */
    . DISCARD_SW = RCV_DISCARD_CHECKS;
    .
    . CALL RCV_CT_CLEANUP;                              /* CLEAN UP CORRELATION TABLE
    .                                                      (PAGE 5-56)                */
    . IF DISCARD_SW = DO_NOT_DISCARD THEN               /* DISCARD CHECKS (PAGE 5-54) */
    .   DO;
    .   . SEND MU TO FMD.RCV;             /* SEND RQ|RSP TO FMDS LAYER
    .   .                                                  (CHAPTER 6)                */
    .   . IF BETWEEN_BRACKETS_CONDITION = YES THEN      /* PAGE 5-58                  */
    .   .   SEND 'BETB' TO FMD.RCV;                     /* CHAPTER 6                  */
    .   END;
    . ELSE
    .   DO;
    .   . IF BETWEEN_BRACKETS_CONDITION = YES THEN      /* PAGE 5-58                  */
    .   .   SEND 'BETB' TO FMD.RCV;                     /* CHAPTER 6                  */
    .   .
    .   . DISCARD MU;
    .   END;
    END;
  ELSE                                                  /* RECEIVE CHECK ERROR        */
    CALL UPM_RECEIVE_CHECKS_PROCESS;                    /* PAGE 5-57                  */

  RETURN;                                               /* RETURN TO DISPATCHER       */
END DFC.RCV;
```

```
RCV_FORMAT: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO ALLOW  REQUESTS AND RESPONSES  TO BE RECEIVED  FROM HALF-SESSIONS │
   │                 NOT SUPPORTING  NEWLY REQUIRED SEND  FORMAT CHECKS.   THIS PROCEDURE │
   │                 MAKES THE FORMAT CORRECT WITH RESPECT TO NEW SEND CHECKS.  │
   │                                                                          │
   │      OUTPUT:    RQ|RSP HAS CORRECT FORMAT WITH RESPECT TO NEW SEND CHECKS. │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 DFC.RCV                          PAGE 5-50               │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */

  SELECT ANYORDER;
  . WHEN(EPI = NORMAL & RRI = RQ)
  .   IF EBI = EB THEN                          /* IF EBI IS SET, IT ...        */
  .     CDI =  ¬CD;                             /* ...OVERRIDES CD SO TURN OFF CDI */
  .
  . WHEN(EPI = NORMAL & RRI = RSP)
  .   ;                                         /* NO FORMAT CONDITIONS         */
  .
  .    WHEN(EPI = EXP & RRI = RQ)
  .      ;                                      /* NO FORMAT CONDITIONS         */
  .
  .      WHEN(EPI = EXP & RRI = RSP)
  .        ;                                    /* NO FORMAT CONDITIONS         */
  END;

  RETURN;
END RCV_FORMAT;
```

```
RCV_CT_INITIALIZE: PROCEDURE;

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO INITIALIZE THE CORRELATION TABLE.                         │
 │                                                                              │
 │      OUTPUT:    THE POINTER TO THE CORRELATION TABLE ENTRY IS INITIALIZED.    │
 │                                                                              │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
 │                 DFC.RCV                          PAGE 5-50                   │
 │                                                                              │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
 │                 CT_ENTRY_ADD_OR_UPDATE           PAGE 5-59                   │
 │                 CT_KEY_SEARCH                    PAGE 5-60                   │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  SELECT ANYORDER;
  . WHEN(EFI = NORMAL & RRI = RQ)
  .   DO;
  .   . CT_PTR = CT_RCV_RQ_NORM;                    /* CORRELATION TABLE TO BE USED    */
  .   . CALL CT_ENTRY_ADD_OR_UPDATE;                /* ADD OR UPDATE AN ENTRY IN THE
  .   .                                               CORRELATION TABLE. THIS ALSO
  .   .                                               INITIALIZES THE PTR TO THAT
  .   .                                               ENTRY. (PAGE 5-59)          */
  .   END;
  .
  . WHEN(EFI = NORMAL & RRI = RSP)
  .   DO;
  .   . CT_PTR = CT_SEND_RQ_NORM;                   /* SET UP PTR TO CORRELATION TABLE */
  .   . SCB.KEY = SNF;                              /* LOCATE ENTRY IN CORR. TABLE...  */
  .   . CT_ENTRY = CT_KEY_SEARCH;                   /* ...CORRESPONDING TO THIS RESPONSE.
  .   .                                               THIS SETS UP PTR TO CORRECT ENTRY.
  .   .                                               (PAGE 5-60)                 */
  .   END;
  .
  . WHEN(EFI = EXP & RRI = RQ)
  .   DO;
  .   . CREATE CT_RCV_RQ_EXP_ENTRY;                 /* CREATE NEW CORRELATION TABLE
  .   .                                               ENTRY. THE ENTRY PTR IS SET
  .   .                                               AT THIS TIME.               */
  .   . CT_RCV_RQ_EXP_ID = SNF;
  .   . CT_RCV_RQ_EXP_DFC_RQ_CODE = RQ_CODE;
  .   . CT_RCV_RQ_EXP_EXR_SENSE = 0;
  .   .
  .   . INSERT CT_RCV_RQ_EXP_ENTRY IN CT_RCV_RQ_EXP; /* INSERT ENTRY INTO...        */
  .   END;                                          /* ...CORRELATION TABLE        */
  .
  . WHEN(EFI = EXP & RRI = RSP)
  .   DO;
  .   . CT_ENTRY = NOT_FOUND;
  .   . SCAN CT_SEND_RQ_EXP PTR(CT_SEND_RQ_EXP_ENTRY_PTR) WHILE(CT_ENTRY = NOT_FOUND);
  .   . . IF CT_SEND_RQ_EXP_ID = SNF THEN           /* LOCATE ENTRY IN CORR...     */
  .   . .    CT_ENTRY = FOUND;                     /* ...TABLE                    */
  .   . .
  .   . SCANEND;
  .   END;
  . END;

  RETURN;
END RCV_CT_INITIALIZE;
```

```
RCV_CHECKS: PROCEDURE RETURNS(BIT(1));

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO DETECT  RECEIVE ERROR  CONDITIONS.  A RECEIVE  ERROR IS  ONE THAT │
 │                 CANNOT  OCCUR  IF   THE  OTHER  HALF-SESSION  HAS   IMPLEMENTED  THE │
 │                 ARCHITECTURE CORRECTLY. THESE CHECKS ARE  OPTIONAL.  (SOME, NONE, OR │
 │                 ALL MAY BE DONE).                                            │
 │                                                                              │
 │      OUTPUT:    A RETURN CODE OF OK (NO RECEIVE ERROR FOUND) OR NG (NO GOOD--RECEIVE │
 │                 ERROR FOUND).                                                │
 │                                                                              │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
 │                 DFC.RCV                              PAGE 5-50               │
 │                                                                              │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
 │                 USAGE_CHECKS                         PAGE 5-61               │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL RC BIT(1);

  RC = OK;                                              /* INITIALIZE RETURN VALUE TO OK   */
  SELECT ANYORDER;
  . WHEN(EFI = NORMAL & RRI = RQ)
  .   DO;
  .   . IF USAGE_CHECKS = NG |                          /* PAGE 5-61                       */
  .   .    SEND_OR_RECEIVE_CHECK(#FSM_HDX) |            /* PAGE 5-82 TO 5-84               */
  .   .    SEND_OR_RECEIVE_CHECK(#FSM_QEC_SEND) |       /* PAGE 5-87                       */
  .   .    SEND_OR_RECEIVE_CHECK(#FSM_SHUTD) |          /* PAGE 5-92 OR 5-92               */
  .   .    SEND_OR_RECEIVE_CHECK(#FSM_IMM_RQ_MODE_RCV) | /* PAGE 5-86                      */
  .   .    SEND_OR_RECEIVE_CHECK(#FSM_CHAIN_RCV) |      /* PAGE 5-72                       */
  .   .    SEND_OR_RECEIVE_CHECK(#FSM_BSM) |            /* PAGE 5-68 OR 5-70               */
  .   .    SEND_OR_RECEIVE_CHECK(#FSM_CONTROL_HDX_RSP_SEND) |
  .   .                                                /* PAGE 5-78,5-79, OR 5-80         */
  .   .    SEND_OR_RECEIVE_CHECK(#FSM_CONTROL_HDX_RSP_RCV) |
  .   .                                                /* PAGE 5-75,5-76, OR 5-77         */
  .   .    SEND_OR_RECEIVE_CHECK(#FSM_SBI_SEND) |       /* PAGE 5-91                       */
  .   .    SEND_OR_RECEIVE_CHECK(#FSM_EBCD_RCV) |       /* PAGE 5-81                       */
  .   .    SEND_OR_RECEIVE_CHECK(#FSM_RTR) |            /* PAGE 5-90 OR 5-90               */
  .   .    SEND_OR_RECEIVE_CHECK(#FSM_QRI_CHAIN_RCV) THEN /* PAGE 5-88                     */
  .   .   RC = NG;
  .   END;
  .
  . WHEN(EFI = NORMAL & RRI = RSP)
  .   DO;
  .   . IF CT_ENTRY = FOUND THEN                       /* IS THIS A RESPONSE TO A REQUEST
  .   .                                                    IN THE CORRELATION TABLE    */
  .   .   SELECT ANYORDER;                              /* YES                             */
  .   .   . WHEN(USAGE_CHECKS = NG)                     /* PAGE 5-61                       */
  .   .   .   RC = NG;
  .   .   . WHEN(CT_DFC_RQ_CODE ¬= CANCEL & CT_RSP_TO_NOT_CANCEL = RCVD)
  .   .   .   RC = NG;
  .   .   . WHEN(RTI = POS & CT_DFC_RQ_CODE ¬= LUSTAT &
  .   .   .      CT_ERI = ER & (CT_DR1I = DR1 | CT_DR2I = DR2))
  .   .   .   RC = NG;                                  /* +RSP TO RQE NOT LUSTAT RQ ARE
  .   .   .                                               NOT ALLOWED. +RSP TO RQE LUSTAT RQ
  .   .   .                                               MAY BE RECEIVED FROM HALF-SESSIONS
  .   .   .                                               NOT SUPPORTING NEWLY REQUIRED
  .   .   .                                               SEND CHECKS.                */
  .   .   . OTHERWISE;
  .   .   END;
  .   . ELSE                                            /* RSP NOT TO REQUEST IN...        */
  .   .   RC = NG;                                      /* ...CORRELATION TABLE            */
  .   END;
  .
  . WHEN(EFI = EXP & RRI = RQ)
  .   DO;
  .   . IF USAGE_CHECKS = NG |                          /* PAGE 5-61                       */
  .   .    SEND_OR_RECEIVE_CHECK(#FSM_QEC_RCV) |        /* PAGE 5-87                       */
  .   .    SEND_OR_RECEIVE_CHECK(#FSM_SHUTD) |          /* PAGE 5-92 OR 5-92               */
  .   .    SEND_OR_RECEIVE_CHECK(#FSM_SBI_RCV) THEN     /* PAGE 5-91                       */
  .   .   RC = NG;
  .   END;
  .
  . WHEN(EFI = EXP & RRI = RSP)
  .   IF CT_ENTRY = NOT_FOUND |                         /* ENTRY NOT IN CORR TABLE OR...   */
  .     USAGE_CHECKS = NG THEN                          /* ...FORMAT ERROR(PAGE 5-61)      */
  .     RC = NG;
  END;

  RETURN(RC);
END RCV_CHECKS;
```

```
RCV_DISCARD_CHECKS: PROCEDURE RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO DETERMINE IF INPUT RQ|RSP IS TO BE DISCARDED.             │
  │                                                                              │
  │      OUTPUT:    RETURN CODE  (RC) SET  TO DO_DISCARD  IF RQ|RSP  IS TO  BE DISCARDED
  │                 OTHERWISE RC IS SET TO DO_NOT_DISCARD.                       │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 DFC.RCV                          PAGE 5-50                   │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL RC BIT(1);

  RC = DO_NOT_DISCARD;                                  /* INITIALIZE RETURN CODE TO
                                                           DO NOT DISCARD          */
  SELECT ANYORDER;
  . WHEN(EFI = NORMAL & RRI = RQ)
  .   DO;
  .   . IF #FSM_CHAIN_RCV = PURGE THEN                 /* IN PURGING CHAIN STATE
  .   .                                                    (PAGE 5-72)          */
  .   .   RC = DO_DISCARD;                              /* REQUEST TO BE DISCARDED */
  .   END;
  .
  . WHEN(EFI = NORMAL & RRI = RSP)
  .   DO;
  .   . IF RTI = POS & CT_DFC_RQ_CODE = LUSTAT &
  .   .    CT_ERI = ER & (CT_DR1I = DR1 | CT_DR2I = DR2) THEN
  .   .   RC = DO_DISCARD;                              /* DISCARD +RSP TO LUSTAT(RQE)
  .   .                                                    RECEIVED FROM HALF-SESSIONS
  .   .                                                    NOT SUPPORTING NEWLY REQUIRED
  .   .                                                    SEND CHECKS.          */
  .   END;
  .
  . WHEN(EFI = EXP & RRI = RQ)
  .   ;                                                 /* NO DISCARD CONDITIONS   */
  .
  .    WHEN(EFI = EXP & RRI = RSP)
  .      ;                                              /* NO DISCARD CONDITIONS   */
  END;

  RETURN(RC);
END RCV_DISCARD_CHECKS;
```

```
RCV_FSMS: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  TO UPDATE THE RECEIVE FINITE-STATE MACHINES. ALSO, CONTENTION ERRORS  │
   │                ARE DETECTED  AT THIS TIME.  WHEN  THEY ARE DETECTED THE  REQUEST IS  │
   │                CONVERTED TO AN EXCEPTION REQUEST (EXR).                              │
   │                                                                                      │
   │     NOTE:      THE ORDER OF CALLS IS SIGNIFICANT FOR THE FOLLOWING FSM'S, WHICH ARE  │
   │                CALLED IN THE ORDER LISTED:                                           │
   │                                                                                      │
   │                •  #FSM_HDX                                                           │
   │                •  #FSM_BSM                                                           │
   │                •  #FSM_CONTROL_BSM_RSP_SEND                                          │
   │                •  #FSM_CONTROL_HDX_RSP_SEND                                          │
   │                •  #FSM_CONTROL_HDX_RSP_RCV                                           │
   │                                                                                      │
   │                THE REASON FOR THE  CALLS HAVING THIS ORDER IS THAT  THE FSMS CALLED  │
   │                LATER MAY CAUSE ADDITIONAL STATE CHANGES TO OCCUR IN THE FSMS CALLED  │
   │                EARLIER. FOR EXAMPLE, IT IS POSSIBLE  FOR TWO STATE CHANGES TO OCCUR  │
   │                IN  #FSM_BSM.  THE  FIRST  ONE OCCURS  WHEN  #FSM_BSM  IS CALLED  TO  │
   │                PROCESS  THE REQUEST  BEING RECEIVED.   THE SECOND  ONE OCCURS  WHEN  │
   │                #FSM_CONTROL_BSM_RSP_RCV,   ALSO   PROCESSING  THE   REQUEST   BEING  │
   │                RECEIVED, DETECTS AN EC AND CALLS  #FSM_BSM WITH A NEGATIVE RESPONSE  │
   │                AS INPUT.                                                             │
   │                                                                                      │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                                        │
   │                DFC.RCV                                  PAGE 5-50                    │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  SELECT ANYORDER;
  . WHEN(EFI = NORMAL & RRI = RQ)
  .   DO;
  .   . CALL #FSM_RES;                          /* PAGE 5-89                    */
  .   . CALL #FSM_HDX;                          /* PAGE 5-82 TO 5-84            */
  .   . CALL #FSM_QEC_SEND;                     /* PAGE 5-87                    */
  .   . CALL #FSM_SHUTD;                        /* PAGE 5-92 OR 5-92            */
  .   . CALL #FSM_IMM_RQ_MODE_RCV;              /* PAGE 5-86                    */
  .   . CALL #FSM_CHAIN_RCV;                    /* PAGE 5-72                    */
  .   . CALL #FSM_BSM;                          /* PAGE 5-68 OR 5-70            */
  .   . CALL #FSM_CONTROL_BSM_RSP_SEND;         /* PAGE 5-74                    */
  .   . CALL #FSM_CONTROL_HDX_RSP_SEND;         /* PAGE 5-78 OR 5-79 OR 5-80    */
  .   . CALL #FSM_CONTROL_HDX_RSP_RCV;          /* PAGE 5-75 OR 5-76 OR 5-77    */
  .   . CALL #FSM_SBI_SEND;                     /* PAGE 5-91                    */
  .   . CALL #FSM_EBCD_RCV;                     /* PAGE 5-81                    */
  .   . CALL #FSM_RTR;                          /* PAGE 5-90 OR 5-90            */
  .   . CALL #FSM_QRI_CHAIN_RCV;                /* PAGE 5-88                    */
  .   END;
  .
  . WHEN(EFI = NORMAL & RRI = RSP)
  .   DO;
  .   . CALL #FSM_QEC_RCV;                      /* PAGE 5-87                    */
  .   . CALL #FSM_IMM_RQ_MODE_SEND;             /* PAGE 5-86                    */
  .   . CALL #FSM_CONTROL_BSM_RSP_RCV;          /* PAGE 5-73                    */
  .   . CALL #FSM_CONTROL_HDX_RSP_RCV;          /* PAGE 5-75 OR 5-76 OR 5-77    */
  .   . CALL #FSM_SBI_RCV;                      /* PAGE 5-91                    */
  .   . CALL #FSM_RTR;                          /* PAGE 5-90 OR 5-90            */
  .   END;
  .
  . WHEN(EFI = EXP & RRI = RQ)
  .   DO;
  .   . CALL #FSM_QEC_RCV;                      /* PAGE 5-87                    */
  .   . CALL #FSM_SHUTD;                        /* PAGE 5-92 OR 5-92            */
  .   . CALL #FSM_SBI_RCV;                      /* PAGE 5-91                    */
  .   END;
  .
  . WHEN(EFI = EXP & RRI = RSP)
  .   DO;
  .   . CALL #FSM_QEC_SEND;                     /* PAGE 5-87                    */
  .   . CALL #FSM_SHUTD;                        /* PAGE 5-92 OR 5-92            */
  .   . CALL #FSM_SBI_SEND;                     /* PAGE 5-91                    */
  .   END;
  END;

  RETURN;
END RCV_FSMS;
```

```
RCV_CT_CLEANUP: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO CLEAN UP CORRELATION TABLES.                              │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                DFC.RCV                          PAGE 5-50                    │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL TEMP_PTR PTR;
  SELECT ANYORDER;
  . WHEN(EFI = NORMAL & RRI = RQ)
  .   DO;
  .   . IF ¬RQN THEN
  .   .   DO;
  .   .   . IF CT_ENTRY_TYPE = WHOLE_CHAIN_NO_CANCEL &
  .   .   .   CT_RSP_TO_NOT_CANCEL = SENT THEN
  .   .   .   REMOVE CT_NORM_ENTRY FROM CT_RCV_RQ_NORM DISCARD;
  .   .   . ELSE
  .   .   .   DO;
  .   .   .   . IF SDI = SD THEN
  .   .   .   .   DO;
  .   .   .   .   . IF RU_CTGY = DFC & RQ_CODE = CANCEL THEN
  .   .   .   .   .   CT_EXR_SENSE_FOR_CANCEL = SNC(0:15);
  .   .   .   .   . ELSE
  .   .   .   .   .   IF CT_EXR_SENSE_FOR_NOT_CANCEL = 0 THEN
  .   .   .   .   .     CT_EXR_SENSE_FOR_NOT_CANCEL = SNC(0:15);
  .   .   .   .   END;
  .   .   .   END;
  .   .   END;
  .   END;
  .
  . WHEN(EFI = NORMAL & RRI = RSP)
  .   DO;
  .   . SELECT ANYORDER;
  .   . . WHEN(CT_ENTRY_TYPE = WHOLE_CHAIN_NO_CANCEL | CT_ENTRY_TYPE = CANCEL_ONLY)
  .   . .   DO;
  .   . .   . IF SCB.PARTNER_HALF_SESSION_RSP_MODE = IMMEDIATE |
  .   . .   .    (RU_CTGY = DFC & RQ_CODE = CHASE) THEN
  .   . .   .   SCAN CT_SEND_RQ_NORM PTR(SCB.SCAN_PTR)
  .   . .   .      UNTIL(SCB.SCAN_PTR = CT_NORM_ENTRY_PTR);
  .   . .   .   . TEMP_PTR = SCB.SCAN_PTR;      /*AVOID SETTING SCAN_PTR TO NULL          */
  .   . .   .   . REMOVE TEMP_PTR->CT_NORM_ENTRY FROM CT_SEND_RQ_NORM DISCARD;
  .   . .   .   SCANEND;
  .   . .   . ELSE
  .   . .   .   REMOVE CT_NORM_ENTRY FROM CT_SEND_RQ_NORM DISCARD;
  .   . .   END;
  .   . .
  .   . . WHEN(CT_ENTRY_TYPE = WHOLE_CHAIN_WITH_CANCEL)
  .   . .   DO;
  .   . .   . IF RU_CTGY = DFC & RQ_CODE = CANCEL THEN
  .   . .   .   DO;
  .   . .   .   . IF SCB.PARTNER_HALF_SESSION_RSP_MODE = IMMEDIATE THEN
  .   . .   .   .   SCAN CT_SEND_RQ_NORM PTR(SCB.SCAN_PTR)
  .   . .   .   .      UNTIL(SCB.SCAN_PTR = CT_NORM_ENTRY_PTR);
  .   . .   .   .   . TEMP_PTR = SCB.SCAN_PTR;   /*AVOID SETTING SCAN_PTR TO NULL          */
  .   . .   .   .   . REMOVE TEMP_PTR->CT_NORM_ENTRY FROM CT_SEND_RQ_NORM DISCARD;
  .   . .   .   .   SCANEND;
  .   . .   .   . ELSE
  .   . .   .   .   REMOVE CT_NORM_ENTRY FROM CT_SEND_RQ_NORM DISCARD;
  .   . .   .   END;
  .   . .   . ELSE
  .   . .   .   CT_RSP_TO_NOT_CANCEL = RECEIVED;
  .   . .   END;
  .   . .
  .   . . WHEN(CT_ENTRY_TYPE = PARTIAL_CHAIN)
  .   . .   CT_RSP_TO_NOT_CANCEL = RECEIVED;
  .   . END;
  .   .
  .   . IF EMPTY(CT_SEND_RQ_NORM) THEN
  .   .   CALL #FSM_QPI_CHECK_SEND('NO_OUTSTANDING_RQS');  /* PAGE 5-88              */
  .   END;
  .
  . WHEN(EFI = EXP & RRI = RQ)
  .   DO;
  .   . IF SDI = SD THEN
  .   .   CT_RCV_RQ_EXP_EXR_SENSE = SNC(0:15);
  .   END;
  .
  . WHEN(EFI = EXP & RRI = RSP)
  .   REMOVE CT_SEND_RQ_EXP_ENTRY FROM CT_SEND_RQ_EXP DISCARD;
  END;

  RETURN;
END RCV_CT_CLEANUP;
```

```
UPM_RECEIVE_CHECKS_PROCESS: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  TO PROCESS RECEIVE  ERROR CONDITIONS.  THESE ERRORS   │
   │                OCCUR ONLY WHEN THE OTHER  HALF-SESSION VIOLATES  THE │
   │                ARCHITECTURE.  THIS PROCEDURE TAKES THE FOLLOWING     │
   │                ACTIONS:                                              │
   │                                                                      │
   │                  •  END THE SESSION  BY SENDING UNBIND; THE  OTHER   │
   │                     HALF-SESSION HAS COMMITTED  A SERIOUS  VIOLATION │
   │                     OF  THE ARCHITECTURE.  UNBIND CARRIES THE SENSE  │
   │                     CODE  INDICATING THE  NATURE OF  THE RECEIVE     │
   │                     CHECK  ERROR.  THIS  SENSE  IS  AVAILABLE  IN    │
   │                     THE RECEIVE_CHECK_SENSE FIELD.                   │
   │                                                                      │
   │                  •  NOTIFY APPROPRIATE OPERATOR ASSOCIATED WITH  THE │
   │                     NAU (FOR SSCP, THIS IS  THE NETWORK OPERATOR; FOR│
   │                     PU, THE NODE  OPERATOR; AND FOR LU, THE TERMINAL │
   │                     OR SUBSYSTEM OPERATOR).  SOME PRODUCTS MAY NOT   │
   │                     HAVE AN APPROPRIATE OPERATOR TO REPORT TO.       │
   │                                                                      │
   │                  •  LOG THE ERROR.                                   │
   │                                                                      │
   │     INPUT:     THE RECEIVE_CHECK_SENSE FIELD CONTAINS THE SENSE CODE │
   │                INDICATING THE TYPE OF ERROR DETECTED.                │
   │                                                                      │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
   │                DFC.RCV                                               │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

                                                  /* NOT ARCHITECTED    */

  RETURN;
END UPM_RECEIVE_CHECKS_PROCESS;
```

```
BETWEEN_BRACKETS_CONDITION: PROCEDURE RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO  DETERMINE THE  BETWEEN BRACKETS  CONDITION (THE  BRACKET FSM  IS      │
  │                 BETWEEN  BRACKETS  AND  THE  APPROPRIATE  CHAINING  FSM  IS  BETWEEN      │
  │                 CHAINS).                                                     │
  │                                                                              │
  │      OUTPUT:    RETURN CODE (RC)  INDICATING WHETHER THE BETWEEN  BRACKETS CONDITION      │
  │                 IS YES OR NO.                                                │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 DFC.RCV                          PAGE 5-50                   │
  │                 DFC.SEND                         PAGE 5-41                   │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL RC BIT(1);

  RC = NO;                                          /* INITIALIZE TO NO          */

  IF #FSM_BSM = BETB THEN                           /* BRACKET FSM BETWEEN BRACKETS */
    DO;
    . IF MUCB.DIRECTION = SEND THEN                 /* SENDING                   */
    .   SELECT ANYORDER;
    .   . WHEN(EFI = NORMAL & RRI = RQ)
    .   .   IF #FSM_CHAIN_SEND = BETC THEN /* BETWEEN CHAIN STATE (PAGE 5-72)  */
    .   .     RC = YES;
    .   . WHEN(EFI = NORMAL & RRI = RSP)
    .   .   IF #FSM_CHAIN_RCV = BETC THEN /* BETWEEN CHAIN STATE (PAGE 5-72)   */
    .   .     RC = YES;
    .   . WHEN(EFI = EXP & RRI = RQ);               /* IGNORE BRACKETS COND. ON EXP */
    .   . WHEN(EFI = EXP & RRI = RSP);              /* IGNORE BRACKETS COND. ON EXP */
    .   END;
    . ELSE                                          /* RECEIVING                 */
    .   SELECT ANYORDER;
    .   . WHEN(EFI = NORMAL & RRI = RQ)
    .   .   IF #FSM_CHAIN_RCV = BETC THEN /* BETWEEN CHAIN STATE (PAGE 5-72)   */
    .   .     RC = YES;
    .   . WHEN(EFI = NORMAL & RRI = RSP)
    .   .   IF #FSM_CHAIN_SEND = BETC THEN /* BETWEEN CHAIN STATE (PAGE 5-72)  */
    .   .     RC = YES;
    .   . WHEN(EFI = EXP & RRI = RQ);               /* IGNORE BRACKETS COND. ON EXP */
    .   . WHEN(EFI = EXP & RRI = RSP);              /* IGNORE BRACKETS COND. ON EXP */
    .   END;
    END;

  RETURN(RC);
END BETWEEN_BRACKETS_CONDITION;
```

```
CT_ENTRY_ADD_OR_UPDATE: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │     FUNCTION:  TO  ADD A  NEW ENTRY  TO  OR TO  UPDATE  AN ENTRY  IN A  NORMAL-FLOW │
  │                CORRELATION TABLE.                                            │
  │                                                                              │
  │     INPUT:     CT_PTR CONTAINS A POINTER TO THE CORRELATION TABLE TO BE ADDED TO OR │
  │                UPDATED.                                                      │
  │                                                                              │
  │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                                │
  │                RCV_CT_INITIALIZE                PAGE 5-52                    │
  │                SEND_CT_INITIALIZE               PAGE 5-46                    │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  IF  ¬RQN THEN
    DO;
    . IF  ¬(EMPTY(CT_PTR)) &
    .   LAST_ENTRY(CT_PTR)->CT_ENTRY_TYPE = PARTIAL_CHAIN THEN
    .   DO;                                                  /* UPDATE LAST ENTRY IN
    .   .                                                       CORRELATION TABLE    */
    .   . CT_NORM_ENTRY_PTR = LAST_ENTRY(CT_PTR);            /* SET PTR TO LAST ENTRY */
    .   . CT_END_SNF = SNF;
    .   . IF ECI = EC THEN
    .   .   DO;
    .   .   . CT_DR1I = DR1I;
    .   .   . CT_DR2I = DR2I;
    .   .   . CT_ERI = ERI;
    .   .   . CT_CDI = CDI;
    .   .   .
    .   .   . IF RU_CTGY = DFC & RQ_CODE = CANCEL THEN
    .   .   .   DO;
    .   .   .   . CT_ENTRY_TYPE = WHOLE_CHAIN_WITH_CANCEL;
    .   .   .   . CT_DFC_RQ_CODE = CANCEL;
    .   .   .   . IF CT_EBI =  ¬EB THEN
    .   .   .   .   CT_EBI = EBI;
    .   .   .   END;
    .   .   . ELSE
    .   .   .   CT_ENTRY_TYPE = WHOLE_CHAIN_NO_CANCEL;
    .   .   END;
    .   END;
    . ELSE                                                   /* TABLE EMPTY OR LAST ENTRY
    .                                                           WHOLE CHAIN          */
    .   DO;                                                  /* CREATE AND ADD NEW ENTRY TO
    .   .                                                       CORRELATION TABLE    */
    .   . CREATE CT_NORM_ENTRY;
    .   .
    .   . CT_BEG_SNF = SNF;
    .   . CT_END_SNF = SNF;
    .   . CT_RSP_TO_NOT_CANCEL = NOT_SENT_OR_RECEIVED;
    .   . CT_EXR_SENSE_FOR_NOT_CANCEL = 0;
    .   . CT_EXR_SENSE_FOR_CANCEL = 0;
    .   . CT_RU_CTGY = RU_CTGY;
    .   . CT_DR1I = DR1I;
    .   . CT_DR2I = DR2I;
    .   . CT_ERI = ERI;
    .   . CT_QRI = QRI;
    .   . CT_BBI = BBI;
    .   . CT_EBI = EBI;
    .   . CT_CDI =  ¬CD;
    .   .
    .   . IF ECI = EC THEN
    .   .   DO;
    .   .   . CT_CDI = CDI;
    .   .   . IF RU_CTGY = DFC & RQ_CODE = CANCEL THEN
    .   .   .   CT_ENTRY_TYPE = CANCEL_ONLY;
    .   .   . ELSE
    .   .   .   CT_ENTRY_TYPE = WHOLE_CHAIN_NO_CANCEL;
    .   .   END;
    .   . ELSE                                               /* NOT END CHAIN        */
    .   .   CT_ENTRY_TYPE = PARTIAL_CHAIN;
    .   .
    .   . IF RU_CTGY = DFC THEN
    .   .   CT_DFC_RQ_CODE = RQ_CODE;
    .   . ELSE
    .   .   CT_DFC_RQ_CODE = 0;
    .   .
    .   . INSERT CT_NORM_ENTRY IN CT_PTR;
    .   END;
    END;

  RETURN;
END CT_ENTRY_ADD_OR_UPDATE;
```

```
CT_KEY_SEARCH: PROCEDURE RETURNS(BIT(1));
                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO SCAN A CORRELATION TABLE LOOKING FOR A SPECIFIC ENTRY │
  │                                                                      │
  │      INPUT:     CT_PTR  CONTAINS POINTER  TO THE  CORRELATION TABLE  TO BE  SCANNED.  │
  │                 "KEY"  CONTAINS  A SEQUENCE  NUMBER  RELATING  TO  THE ENTRY  TO  BE  │
  │                 SEARCHED FOR.                                        │
  │                                                                      │
  │      OUTPUT:    RETURN CODE(RC) INDICATING  WHETHER OR NOT THE ENTRY  WAS FOUND.  IF  │
  │                 THE ENTRY WAS FOUND, CT_NORM_ENTRY_PTR CONTAINS ITS POINTER.  │
  │                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
  │                 DFC.SEND_CHECKS                  PAGE 5-42           │
  │                 RCV_CT_INITIALIZE                PAGE 5-52           │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL RC BIT(1);

  RC = NOT_FOUND;
  SCAN CT_PTR PTR(CT_NORM_ENTRY_PTR) WHILE(RC = NOT_FOUND);
  . IF CT_END_SNF - CT_BEG_SNF >= 0 THEN
  .    DO;
  .    . IF SCB.KEY >= CT_BEG_SNF &
  .    .    SCB.KEY <= CT_END_SNF THEN
  .    .    RC = FOUND;
  .    END;
  . ELSE
  .    DO;
  .    . IF SCB.KEY <= CT_END_SNF |
  .    .    SCB.KEY >= CT_BEG_SNF THEN
  .    .    RC = FOUND;
  .    END;
  SCAN END;

  RETURN(RC);
END CT_KEY_SEARCH;
```

```
USAGE_CHECKS: PROCEDURE RETURNS(BIT(1));

                                                                                /*
┌──────────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE PERFORMS USAGE CHECKS  ON ALL REQUESTS AND RESPONSES.  │
│                 USAGE  CHECKS  ARE  CHECKS  INVOLVING THE  RH  AND  VARIOUS  SESSION  │
│                 ACTIVATION  PARAMETERS.   USAGE  CHECKS   ARE  BY  DEFINITION  STATE  │
│                 INDEPENDENT, AND THUS INVOLVE NO FSM STATES.                 │
│                                                                              │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
│                 DFC.SEND_CHECKS                  PAGE 5-42                   │
│                 RCV_CHECKS                       PAGE 5-53                   │
│                                                                              │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
│                 USAGE_CHECKS_EXP_RQ              PAGE 5-62                   │
│                 USAGE_CHECKS_EXP_RSP             PAGE 5-63                   │
│                 USAGE_CHECKS_NORMAL_RQ_DFC       PAGE 5-64                   │
│                 USAGE_CHECKS_NORMAL_RQ_FMD       PAGE 5-66                   │
│                 USAGE_CHECKS_NORMAL_RSP          PAGE 5-67                   │
└──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL RC BIT(1);
  DCL USAGE_SENSE BIT(16);

  RC = OK;
  USAGE_SENSE=X'0000';
  SELECT ANYORDER;
  . WHEN(EFI = NORMAL & RRI = RQ)
  .   DO;
  .   . IF RU_CTGY = DFC THEN
  .   .   USAGE_SENSE = USAGE_CHECKS_NORMAL_RQ_DFC;      /* PAGE 5-64          */
  .   . ELSE                                             /* FM DATA            */
  .   .   USAGE_SENSE = USAGE_CHECKS_NORMAL_RQ_FMD;      /* PAGE 5-66          */
  .   END;
  .
  . WHEN(EFI = NORMAL & RRI = RSP)
  .   USAGE_SENSE = USAGE_CHECKS_NORMAL_RSP;             /* PAGE 5-67          */
  .
  . WHEN(EFI = EXP & RRI = RQ)
  .   USAGE_SENSE = USAGE_CHECKS_EXP_RQ;                 /* PAGE 5-62          */
  .
  . WHEN(EFI = EXP & RRI = RSP)
  .   USAGE_SENSE = USAGE_CHECKS_EXP_RSP;                /* PAGE 5-63          */
  END;

  IF USAGE_SENSE ¬= X'0000' THEN                         /* USAGE ERROR FOUND (WHEN A USAGE
                                                            ERROR IS FOUND THE USAGE_SENSE
                                                            FIELD CONTAINS THE APPPOPRIATE
                                                            SENSE CODE)        */
    DO;                                                  /* SET UP SEND OR RECEIVE SENSE */
    . RC=NG;                                             /* NO GOOD RETURN CODE */
    . IF MUCB.DIRECTION = SEND THEN
    .   SEND_CHECK_SENSE = USAGE_SENSE;
    . ELSE
    .   RECEIVE_CHECK_SENSE = USAGE_SENSE;
    END;

  RETURN(RC);
END USAGE_CHECKS;
```

```
USAGE_CHECKS_EXP_RQ: PROCEDURE RETURNS(BIT(16));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  PERFORMS USAGE CHECKS FOR EXPEDITED-FLOW REQUESTS.           │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 USAGE_CHECKS                          PAGE 5-61              │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL USAGE_SENSE BIT(16);

  USAGE_SENSE = X'0000';
  SELECT ANYORDER;
  . WHEN(RU_CTGY ¬= DFC)
  .   USAGE_SENSE = X'4011';
  . WHEN(FI =  ¬FMH)
  .   USAGE_SENSE = X'400F';
  . WHEN(BCI =  ¬BC | ECI =  ¬EC)
  .   USAGE_SENSE = X'400B';
  . WHEN(DR1I =  ¬DR1 | DR2I = DR2 | ERI = ER)
  .   USAGE_SENSE = X'4014';
  . WHEN(QRI = QR)
  .   USAGE_SENSE = X'4015';
  . WHEN(BBI = BB | EBI = EB)
  .   USAGE_SENSE = X'400C';
  . WHEN(CDI = CD)
  .   USAGE_SENSE = X'4009';
  . WHEN(CSI = CODE1)
  .   USAGE_SENSE = X'4010';
  . WHEN(EDI = ED)
  .   USAGE_SENSE = X'4016';
  . WHEN(PDI = PD)
  .   USAGE_SENSE = X'4017';
  . OTHERWISE
  .   DO;
  .   . IF MUCB.DIRECTION = SEND THEN
  .   .   SELECT ANYORDER;
  .   .   . WHEN(RQ_CODE = QEC & SCB.DFC_QEC_SEND = ALLOWED);
  .   .   . WHEN(RQ_CODE = RELQ & SCB.DFC_RELQ_SEND = ALLOWED);
  .   .   . WHEN(RQ_CODE = RSHUTD & SCB.DFC_RSHUTD_SEND = ALLOWED);
  .   .   . WHEN(RQ_CODE = SBI & SCB.DFC_SBI_SEND = ALLOWED);
  .   .   . WHEN(RQ_CODE = SHUTC & SCB.DFC_SHUTC_SEND = ALLOWED);
  .   .   . WHEN(RQ_CODE = SHUTD & SCB.DFC_SHUTD_SEND = ALLOWED);
  .   .   . WHEN(RQ_CODE = SIG & SCB.DFC_SIG_SEND = ALLOWED);
  .   .   . OTHERWISE
  .   .   .   USAGE_SENSE = X'1003';
  .   .   END;
  .   . ELSE                                          /* RECEIVING              */
  .   .   SELECT ANYORDER;
  .   .   . WHEN(RQ_CODE = QEC & SCB.DFC_QEC_RCV = ALLOWED);
  .   .   . WHEN(RQ_CODE = RELQ & SCB.DFC_RELQ_RCV = ALLOWED);
  .   .   . WHEN(RQ_CODE = RSHUTD & SCB.DFC_RSHUTD_RCV = ALLOWED);
  .   .   . WHEN(RQ_CODE = SBI & SCB.DFC_SBI_RCV = ALLOWED);
  .   .   . WHEN(RQ_CODE = SHUTC & SCB.DFC_SHUTC_RCV = ALLOWED);
  .   .   . WHEN(RQ_CODE = SHUTD & SCB.DFC_SHUTD_RCV = ALLOWED);
  .   .   . WHEN(RQ_CODE = SIG & SCB.DFC_SIG_RCV = ALLOWED);
  .   .   . OTHERWISE
  .   .   .   USAGE_SENSE = X'1003';
  .   .   END;
  .   END;
  END;

  RETURN(USAGE_SENSE);
END USAGE_CHECKS_EXP_RQ;
```

```
USAGE_CHECKS_EXP_RSP: PROCEDURE RETURNS(BIT(16));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  PERFORMS USAGE CHECKS ON EXPEDITED-FLOW RESPONSES.           │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 USAGE_CHECKS                          PAGE 5-61              │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL USAGE_SENSE BIT(16);

  USAGE_SENSE = X'0000';
  SELECT ANYORDER;
  . WHEN(RU_CTGY ¬= DFC)
  .   USAGE_SENSE = X'4011';
  . WHEN(FI =  ¬FMH)
  .   USAGE_SENSE = X'400F';
  . WHEN(SDI ¬= RTI)
  .   USAGE_SENSE = X'4013';
  . WHEN(BCI =  ¬BC | ECI =  ¬EC)
  .   USAGE_SENSE = X'400B';
  . WHEN(DR1I =  ¬DR1 | DR2I = DR2)
  .   USAGE_SENSE = X'4014';
  . WHEN(QRI = QR)
  .   USAGE_SENSE = X'4015';
  . WHEN((MUCB.DIRECTION = SEND & RQ_CODE¬=CT_RCV_RQ_EXP_DFC_RQ_CODE) |
  .      (MUCB.DIRECTION = RECEIVE & RQ_CODE ¬= CT_SEND_RQ_EXP_DFC_RQ_CODE))
  .   USAGE_SENSE = X'4012';
  . OTHERWISE;
  END;

  RETURN(USAGE_SENSE);
END USAGE_CHECKS_EXP_RSP;
```

```
USAGE_CHECKS_NORMAL_RQ_DFC: PROCEDURE RETURNS(BIT(16));

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE PERFORMS USAGE CHECKS FOR NORMAL-FLOW DFC REQUESTS.  │
 │                                                                              │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
 │                 USAGE_CHECKS                    PAGE 5-61                    │
 │                                                                              │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
 │                 USAGE_CHECKS_NORMAL_RQ_DFC_1    PAGE 5-65                    │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL USAGE_SENSE BIT(16);

  USAGE_SENSE = USAGE_CHECKS_NORMAL_RQ_DFC_1;           /* PAGE 5-65              */
  IF USAGE_SENSE = X'0000' THEN                         /* PAGE 5-65 A  RETURN VALUE OF  */
    DO;                                                 /* X'0000' MEANS NO ERRORS FOUND */
    . IF MUCB.DIRECTION = SEND THEN
    .   SELECT ANYORDER;
    .   . WHEN(RQ_CODE = BID & SCB.DFC_BID_SEND = ALLOWED);
    .   . WHEN(RQ_CODE = BIS & SCB.DFC_BIS_SEND = ALLOWED);
    .   . WHEN(RQ_CODE = CANCEL & SCB.DFC_CANCEL_SEND = ALLOWED);
    .   . WHEN(RQ_CODE = CHASE & SCB.DFC_CHASE_SEND = ALLOWED);
    .   . WHEN(RQ_CODE = LUSTAT & SCB.DFC_LUSTAT_SEND = ALLOWED);
    .   . WHEN(RQ_CODE = QC & SCB.DFC_QC_SEND = ALLOWED);
    .   . WHEN(RQ_CODE = RTR & SCB.DFC_RTR_SEND = ALLOWED);
    .   . OTHERWISE
    .   .   USAGE_SENSE = X'1003';                      /* FUNCTION NOT SUPPORTED */
    .   END;
    . ELSE                                              /* MUCB.DIRECTION=RECEIVE */
    .   SELECT ANYORDER;
    .   . WHEN(RQ_CODE = BID & SCB.DFC_BID_RCV = ALLOWED);
    .   . WHEN(RQ_CODE = BIS & SCB.DFC_BIS_RCV = ALLOWED);
    .   . WHEN(RQ_CODE = CANCEL & SCB.DFC_CANCEL_RCV = ALLOWED);
    .   . WHEN(RQ_CODE = CHASE & SCB.DFC_CHASE_RCV = ALLOWED);
    .   . WHEN(RQ_CODE = LUSTAT & SCB.DFC_LUSTAT_RCV = ALLOWED);
    .   . WHEN(RQ_CODE = QC & SCB.DFC_QC_RCV = ALLOWED);
    .   . WHEN(RQ_CODE = RTR & SCB.DFC_RTR_RCV = ALLOWED);
    .   . OTHERWISE
    .   .   USAGE_SENSE = X'1003';                      /* FUNCTION NOT SUPPORTED */
    .   END;

    . IF USAGE_SENSE = X'0000' THEN                     /* NO ERRORS FOUND SO FAR */
    .   SELECT ANYORDER;
    .   . WHEN(SCB.SEND_RCV_MODE = FULL_DUPLEX & CDI = CD)
    .   .   USAGE_SENSE = X'400D';
    .   . WHEN(SCB.USING_BRACKETS = NO & (EBI = EB | BBI = BB))
    .   .   USAGE_SENSE = X'400C';
    .   . OTHERWISE
    .   .   DO;
    .   .   . IF (MUCB.DIRECTION = SEND & SCB.HALF_SESSION=PRI) |
    .   .   .    (MUCB.DIRECTION = RECEIVE & SCB.HALF_SESSION = SEC) THEN
    .   .   .   DO;
    .   .   .   . IF SCB.PRI_EB_IND = MAY_NOT_SEND & EBI = EB THEN
    .   .   .   .   USAGE_SENSE = X'4004';
    .   .   .   END;
    .   .   . ELSE                                      /* (SEND & SEC)|(RECEIVE & PRI)  */
    .   .   .   IF SCB.SEC_EB_IND = MAY_NOT_SEND & EBI = EB THEN
    .   .   .     USAGE_SENSE = X'4004';
    .   .   END;
    .   END;
    END;


  RETURN(USAGE_SENSE);
END USAGE_CHECKS_NORMAL_RQ_DFC;
```

```
USAGE_CHECKS_NORMAL_RQ_DFC_1: PROCEDURE RETURNS(BIT(16));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  PERFORMS FORMAT CHECKS ON NORMAL-FLOW DFC REQUESTS.          │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 USAGE_CHECKS_NORMAL_RQ_DFC          PAGE 5-64                │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL USAGE_SENSE BIT(16);

  USAGE_SENSE = X'0000';
  SELECT ANYORDER;
  . WHEN(FI =  ¬FMH)
  .   USAGE_SENSE = X'400F';
  . WHEN(BCI =  ¬BC | ECI =  ¬EC)
  .   USAGE_SENSE = X'400B';
  . WHEN(CSI = CODE1)
  .   USAGE_SENSE = X'4010';
  . WHEN(EDI = ED)
  .   USAGE_SENSE = X'4016';
  . WHEN(PDI = PD)
  .   USAGE_SENSE = X'4017';
  . OTHERWISE
  .   SELECT ANYORDER;
  .   . WHEN(RQ_CODE = BID | RQ_CODE = BIS | RQ_CODE = RTR)
  .   .   SELECT ANYORDER;
  .   .   . WHEN(DR1I = ¬DR1 | DR2I = DR2 | ERI = ER)
  .   .   .   USAGE_SENSE = X'4014';
  .   .   . WHEN(BBI = BB | EBI = EB)
  .   .   .   USAGE_SENSE = X'400C';
  .   .   . WHEN(CDI = CD)
  .   .   .   USAGE_SENSE = X'4009';
  .   .   . OTHERWISE;
  .   .   END;
  .   . WHEN(RQ_CODE = CANCEL | RQ_CODE = CHASE | RQ_CODE = QC)
  .   .   SELECT ANYORDER;
  .   .   . WHEN(DR1I = ¬DR1 | DR2I = DR2 | ERI = ER)
  .   .   .   USAGE_SENSE = X'4014';
  .   .   . WHEN(BBI = BB)
  .   .   .   USAGE_SENSE = X'4003';
  .   .   . WHEN(EBI = EB & CDI = CD)
  .   .   .   USAGE_SENSE = X'4009';
  .   .   . OTHERWISE;
  .   .   END;
  .   . WHEN(RQ_CODE = LUSTAT)
  .   .   SELECT ANYORDER;
  .   .   . WHEN(DR1I = ¬DR1 & DR2I = ¬DR2)
  .   .   .   USAGE_SENSE = X'4014';
  .   .   . WHEN(EBI = EB & CDI = CD)
  .   .   .   USAGE_SENSE = X'4009';
  .   .   . OTHERWISE;
  .   .   END;
  .   END;
  END;

  RETURN(USAGE_SENSE);
END USAGE_CHECKS_NORMAL_RQ_DFC_1;
```

```
USAGE_CHECKS_NORMAL_RQ_FMD: PROCEDURE RETURNS(BIT(16));

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS  PROCEDURE  PERFORMS  USAGE  CHECKS  FOR  NORMAL-FLOW  FM  DATA  │
 │                 REQUESTS.                                            │
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 USAGE_CHECKS                    PAGE 5-61            │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL USAGE_SENSE BIT(16);

  USAGE_SENSE = X'0000';
  SELECT ANYORDER;
  . WHEN(RQD & ECI =  ¬EC & MUCB.DIRECTION = SEND)
  .   USAGE_SENSE = X'4007';
  . WHEN(CDI = CD & ECI =  ¬EC)
  .   USAGE_SENSE = X'4009';
  . WHEN(BBI = BB & BCI =  ¬BC)
  .   USAGE_SENSE = X'4003';
  . WHEN(EBI = EB & BCI =  ¬BC)
  .   USAGE_SENSE = X'4004';
  . WHEN(FI = FMH & BCI =  ¬BC)
  .   USAGE_SENSE = X'400F';
  . WHEN(SCB.USING_BRACKETS = NO & (BBI = BB | EBI = EB))
  .   USAGE_SENSE = X'400C';
  . WHEN(EBI = EB & CDI = CD)
  .   USAGE_SENSE = X'4009';
  . WHEN(SCB.ALT_CODE = NOT_USED & CSI = CODE1)
  .   USAGE_SENSE = X'4010';
  . WHEN(SCB.SEND_RCV_MODE = FULL_DUPLEX & CDI = CD)
  .   USAGE_SENSE = X'400D';
  . OTHERWISE
  .   SELECT ANYORDER;
  .   . WHEN((MUCB.DIRECTION = SEND & SCB.HALF_SESSION=PRI) |
  .   .     (MUCB.DIRECTION = RECEIVE & SCB.HALF_SESSION = SEC))
  .   .   SELECT ANYORDER;
  .   .   . WHEN(SCB.PRI_CHAIN_USE = SINGLE &  ¬(BCI = BC & ECI = EC))
  .   .   .   USAGE_SENSE = X'400B';
  .   .   . WHEN(SCB.PRI_EB_IND = MAY_NOT_SEND & EBI = EB)
  .   .   .   USAGE_SENSE = X'4004';
  .   .   . WHEN(RQD & SCB.PRI_DEF_RSP_CHAIN = NOT_ALLOWED)
  .   .   .   USAGE_SENSE = X'4007';
  .   .   . WHEN(RQE & ECI = EC & SCB.PRI_EXCP_RSP_CHAIN = NOT_ALLOWED)
  .   .   .   USAGE_SENSE = X'4006';
  .   .   . WHEN(RQE & ECI =  ¬EC & SCB.PRI_EXCP_RSP_CHAIN = NOT_ALLOWED &
  .   .   .     SCB.PRI_DEF_RSP_CHAIN = NOT_ALLOWED)
  .   .   .   USAGE_SENSE = X'4006';
  .   .   . WHEN(RQN & SCB.PRI_NO_RSP_CHAIN = NOT_ALLOWED)
  .   .   .   USAGE_SENSE = X'400A';
  .   .   . OTHERWISE;
  .   .   END;
  .   . OTHERWISE                                    /* (MUCB.DIRECTION=SEND & SEC) OR
  .   .                                                 (MUCB.DIRECTION=RECEIVE & PRI)  */
  .   .   SELECT ANYORDER;
  .   .   . WHEN(SCB.SEC_CHAIN_USE = SINGLE &  ¬(BCI = BC & ECI = EC))
  .   .   .   USAGE_SENSE = X'400B';
  .   .   . WHEN(SCB.SEC_EB_IND = MAY_NOT_SEND & EBI = EB)
  .   .   .   USAGE_SENSE = X'4004';
  .   .   . WHEN(RQD & SCB.SEC_DEF_RSP_CHAIN = NOT_ALLOWED)
  .   .   .   USAGE_SENSE = X'4007';
  .   .   . WHEN(RQE & ECI = EC & SCB.SEC_EXCP_RSP_CHAIN = NOT_ALLOWED)
  .   .   .   USAGE_SENSE = X'4006';
  .   .   . WHEN(RQE & ECI =  ¬EC & SCB.SEC_EXCP_RSP_CHAIN = NOT_ALLOWED &
  .   .   .     SCB.SEC_DEF_RSP_CHAIN = NOT_ALLOWED)
  .   .   .   USAGE_SENSE = X'4006';
  .   .   . WHEN(RQN & SCB.SEC_NO_RSP_CHAIN = NOT_ALLOWED)
  .   .   .   USAGE_SENSE = X'400A';
  .   .   . OTHERWISE;
  .   .   END;
  .   END;
  END;

  RETURN(USAGE_SENSE);
END USAGE_CHECKS_NORMAL_RQ_FMD;
```

```
USAGE_CHECKS_NORMAL_RSP: PROCEDURE RETURNS(BIT(16));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  PERFORM USAGE CHECKS ON NORMAL FLOW RESPONSES.               │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 USAGE_CHECKS                          PAGE 5-61              │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL USAGE_SENSE BIT(16);

  SELECT ANYORDER;
  . WHEN(BCI =  ¬BC | ECI =  ¬EC)
  .   USAGE_SENSE = X'400B';
  . WHEN(SDI ¬= RTI)
  .   USAGE_SENSE = X'4013';
  . OTHERWISE
  .   DO;
  .   . IF CT_ENTRY_TYPE = WHOLE_CHAIN_WITH_CANCEL &
  .   .   SNF = CT_END_SNF THEN                          /* THIS IS RSP TO CANCEL      */
  .   .   SELECT ANYORDER;
  .   .   . WHEN(RU_CTGY ¬= DFC)
  .   .   .   USAGE_SENSE = X'4011';
  .   .   . WHEN(FI =  ¬FMH)
  .   .   .   USAGE_SENSE = X'400F';
  .   .   . WHEN(DR1I ¬= DR1 | DR2I = DR2)
  .   .   .   USAGE_SENSE = X'4014';
  .   .   . WHEN(RQ_CODE ¬= CANCEL)
  .   .   .   USAGE_SENSE = X'4012';
  .   .   . OTHERWISE;
  .   .   END;
  .   . ELSE
  .   .   SELECT ANYORDER;
  .   .   . WHEN(RU_CTGY ¬= CT_RU_CTGY)
  .   .   .   USAGE_SENSE = X'4011';
  .   .   . WHEN(DR1I ¬= CT_DR1I | DR2I ¬= CT_DR2I)
  .   .   .   USAGE_SENSE = X'4014';
  .   .   . WHEN(RU_CTGY = DFC & FI =  ¬FMH)
  .   .   .   USAGE_SENSE = X'400F';
  .   .   . WHEN(RU_CTGY = DFC & RQ_CODE ¬= CT_DFC_RQ_CODE)
  .   .   .   USAGE_SENSE = X'4012';
  .   .   . WHEN(RU_CTGY = FMD & SCB.TYPE_OF_SESSION = LU_LU &
  .   .   .     SCB.FM_HDR_USAGE = FM_HEADERS & RTI = POS & FI = FMH)
  .   .   .   USAGE_SENSE = X'400F';
  .   .   . OTHERWISE;
  .   .   END;
  .   END;
  END;

  RETURN(USAGE_SENSE);
END USAGE_CHECKS_NORMAL_RSP;



UPM_RES: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS UPM  HANDLES SENDING OF UNAVL  AND AVL SIGNALS TO  THE RESOURCE │
  │                 FSM (FSM_RES, PAGE 5-87).                                    │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

                                                        /* NOT ARCHITECTED            */

  RETURN;
END UPM_RES;
```

```
FSM_BSM_BIDDER: FSM_DEFINITION CONTEXT(SCB),
MULTIPLE_ACTION_CODES(2),,,;
                                                                    /*
```

FUNCTION: THIS FSM ENFORCES THE BRACKETS PROTOCOL FOR THE BIDDER. SEE "BRACKETS PROTOCOL" ON PAGE 5-14 FOR PROSE DESCRIPTION.

THE TWO PRIMARY STATES IN THIS FSM ARE:

- BETB (BETWEEN BRACKETS): THIS STATE INDICATES NO BRACKET IS CURRENTLY BEING PROCESSED. ANY REQUEST CHAINS SENT OR RECEIVED IN THIS STATE (WITH THE EXCEPTION OF SOME DFC REQUESTS) HAVE THE BBI (BEGIN BRACKET INDICATOR) SET.

- INB (IN BRACKET): THIS STATE INDICATES A BRACKET IS CURRENTLY BEING PROCESSED. A REQUEST WITH BBI SET HAS PREVIOUSLY BEEN SENT OR RECEIVED TO BEGIN THE BRACKET. REQUEST CHAINS SENT IN THIS STATE DO NOT HAVE BBI SET. HOWEVER, THEY MAY HAVE EBI (END BRACKET INDICATOR) SET IF IT IS DESIRED THAT THE BRACKET BE ENDED.

THE REST OF THE STATES IN THIS FSM ARE TRANSITION STATES.

- PEND_BB (PENDING SENDING BEGIN BRACKET): THIS STATE IS ENTERED WHEN A POSITIVE RESPONSE TO BID IS RECEIVED OR A POSITIVE RESPONSE TO RTR IS SENT. IT MEANS THE BIDDER HAS BEEN GRANTED THE RIGHT TO START A BRACKET (BY SENDING BB).

- PEND_INB (PENDING ENTERING IN-BRACKET STATE): THIS STATE IS ENTERED WHEN THE BIDDER SENDS A BB REQUEST WHILE IN BETB. THE BIDDER IS REQUESTING PERMISSION TO BEGIN A BRACKET. PERMISSION IS GRANTED (BY THE FIRST SPEAKER) WHEN A POSITIVE RESPONSE TO THE BB REQUEST IS RECEIVED. THIS CAUSES A TRANSITION TO THE IN-BRACKET STATE (INB). PERMISSION IS DENIED WHEN A NEGATIVE RESPONSE TO THE BB REQUEST IS RECEIVED. THIS CAUSES A TRANSITION BACK TO BETWEEN-BRACKETS STATE (BETB).

- PEND_TERM_S AND PEND_TERM_R (PENDING TERMINATION OF THE BRACKET): THESE STATES ARE ENTERED FROM THE IN-BRACKET STATE (INB) WHEN A REQUEST CARRYING EB (END BRACKET) IS SENT OR RECEIVED. THE BRACKET IS TERMINATED (TRANSITION MADE TO BETWEEN-BRACKETS STATE (BETB)) IF THE END CHAIN REQUEST OF THE EB CHAIN DID NOT ASK FOR DEFINITE RESPONSE OR, A POSITIVE RESPONSE IS RECEIVED FOR THE EB CHAIN. THE BRACKET IS NOT TERMINATED (TRANSITION MADE BACK TO IN-BRACKET STATE (INB)) IF EB CHAIN IS CANCELED (USING DFC CANCEL REQUEST) OR A NEGATIVE RESPONSE TO THE EB CHAIN IS RECEIVED.

WHEN THE BRACKETS AND HALF-DUPLEX FLIP FLOP PROTOCOLS (SEE "SEND/RECEIVE MODE PROTOCOLS" ON PAGE 5-12) ARE USED, A TIGHT COUPLING EXISTS BETWEEN THE FSM'S, FSM_BSM_BIDDER AND FSM_HDX_FF (HALF-DUPLEX FLIP FLOP FSM, PAGE 5-84). A STRONG COORDINATION OF THE STATES OF THESE TWO FSM'S IS NECESSARY. THE GENERAL RULES ARE:

- WHENEVER FSM_BSM_BIDDER IS IN BETWEEN-BRACKETS STATE (BETB), FSM_HDX_FF IS IN ONE OF ITS CONTENTION STATES (CONT, CONT_SEND, OR CONT_RCV).

- WHEN FSM_BSM_BIDDER GOES TO IN-BRACKET STATE (INB), FSM_HDX_FF GOES TO SEND (SEND) OR RECEIVE (RCV) STATE. THE CDI BIT (CHANGE DIRECTION INDICATOR) DETERMINES WHETHER IT IS SEND OR RECEIVE STATE.

IN ORDER TO FOLLOW THESE RULES FSM_BSM_BIDDER TELLS FSM_HDX_FF WHEN TO GO TO CONTENTION, SEND, AND RECEIVE STATES. THIS IS DONE BY CALLING FSM_HDX_FF AND GIVING IT THE SIGNAL INPUTS BETB, INB_RCV, AND INB_SEND.

- BETB SIGNAL MEANS FSM_BSM_BIDDER IS GOING TO BETWEEN-BRACKETS STATE (BETB) AND FSM_HDX_FF IS TO GO TO CONTENTION STATE (CONT). NOTICE THAT THIS SIGNAL IS GIVEN TO FSM_HDX_FF ON END CHAIN REQUESTS. THIS IS BECAUSE THE CONTENTION STATE (CONT) MUST BE ENTERED ONLY WHEN BETWEEN SENDING OR RECEIVING CHAINS.

- INB_SEND SIGNAL MEANS FSM_BSM_BIDDER IS GOING TO IN-BRACKET STATE (INB) AND FSM_HDX_FF IS TO GO TO SEND STATE (SEND).

- INB_RCV SIGNAL MEANS FSM_BSM_BIDDER IS GOING TO IN-BRACKET STATE (INB) AND FSM_HDX_FF IS TO GO TO RECEIVE STATE (RCV).

NOTE: RECEIVED AND SENT RESPONSES COME TO THIS FSM FROM FSM_CONTROL_BSM_RSP_SEND (PAGE 5-74) AND FSM_CONTROL_BSM_RSP_RCV (PAGE 5-73). THESE FSM'S ALLOW RESPONSES TO COME TO FSM_BSM_BIDDER ONLY WHEN THE RESPONSE IS FOR THE CURRENT CHAIN AND THE LAST REQUEST OF THE CURRENT CHAIN HAS BEEN SENT OR RECEIVED. THESE CONTROL FSM'S RELIEVE FSM_BSM_BIDDER FROM HAVING STATES TO REMEMBER WHEN A NEGATIVE RESPONSE IS RECEIVED WHILE IN THE MIDDLE OF SENDING A CHAIN.

REFERENCED BY THE FOLLOWING PROCEDURE(S):

DEQUEUE.Q_TC_TO_DFC	PAGE 5-40
FSM_HDX_FF	PAGE 5-84

```
                                                                    */
```

INPUT \ STATE NAMES----->	BETB 1	INB 2	PEND BB 3	PEND INB 4	PEND TERM_R 5	PEND TERM_S 6
S,RQ,FMD|LUSTAT, BB, EB	-	>(S)	1	>(S)	>(S)	>(S)
S,RQ,FMD|LUSTAT, BB,¬EB, RQD, CD	4	>(S)	2(IR)	>(S)	>(S)	>(S)
S,RQ,FMD|LUSTAT, BB,¬EB, RQD,¬CD	4	>(S)	2(IS)	>(S)	>(S)	>(S)
S,RQ,FMD|LUSTAT, BB,¬EB, ¬RQD, CD	>(S)	>(S)	2(IR)	>(S)	>(S)	>(S)
S,RQ,FMD|LUSTAT, BB,¬EB, EC,¬RQD,¬CD	>(S)	>(S)	2(IS)	>(S)	>(S)	>(S)
S,RQ,FMD , BB,¬EB, ¬EC	4	>(S)	-	>(S)	>(S)	>(S)
S,RQ,FMD|EBDFC ,¬BB, EB, RQD	>(S)	6,1(B)	>(S)	>(S)	>(S)	>(S)
S,RQ,FMD|EBDFC ,¬BB, EB, EC,¬RQD	>(S)	1(B)	>(S)	>(S)	>(S)	>(S)
S,RQ,CANCEL , EB	>(S)	1(B)	>(S)	>(S)	>(S)	1(B)
S,RQ,FMD ,¬BB, EB, ¬EC	>(S)	6,1	>(S)	>(S)	>(S)	>(S)
S,RQ,FMD ,¬BB,¬EB, BC	>(S)	-	>(S)	>(S)	>(S)	>(S)
S,RQ,FMD , ¬BC, RQD, CD	-(B)	-	2(IR)	-	>(S)	-
S,RQ,FMD , ¬BC, RQD,¬CD	-(B)	-	2(IS)	-	>(S)	-
S,RQ,FMD , ¬BC, ¬RQD, CD	-(B)	-	2(IR)	>(S)	>(S)	1(B)
S,RQ,FMD , ¬BC, EC,¬RQD,¬CD	-(B)	-	2(IS)	>(S)	>(S)	1(B)
S,RQ,CANCEL , ¬EB, CD	-(B)	-	2(IR)	1	>(S)	2
S,RQ,CANCEL , ¬EB, ¬CD	-(B)	-	2(IS)	1	>(S)	2
S,RQ,BID	-	>(S)	>(S)	>(S)	>(S)	>(S)
S,RQ,OTHERDFC ,¬BB,¬EB	-	-	-	>(S)	>(S)	>(S)
R,+RSP,FMD|LUSTAT,CT(BB,¬EB, CD)	-	-	-	2(IR)	-	-
R,+RSP,FMD|LUSTAT,CT(BB,¬EB,¬CD)	-	-	-	2(IS)	-	-
R,-RSP,FMD|LUSTAT,CT(BB,¬EB)	-	-	-	1	-	-
R,+RSP,¬BID ,CT(¬BB, EB)	-	-	-	-	-	1(B)
R,-RSP,¬BID ,CT(¬BB, EB)	-	-	-	-	-	2
R,+RSP, BID	3	-	-	-	-	-
R,RQ,FMD|LUSTAT, BB, EB	-	-	-	-	>(R)	-(C)
R,RQ,FMD|LUSTAT, BB,¬EB, CD	2(IS)	>(R)	>(R)	2(IS)	>(R)	>(R)
R,RQ,FMD|LUSTAT, BB,¬EB, EC, ¬CD	2(IR)	>(R)	>(R)	2(IR)	>(R)	>(R)
R,RQ,FMD|LUSTAT, BB,¬EB, ¬EC	2(IR)	>(R)	>(R)	2(IR)	>(R)	>(R)
R,RQ,FMD|EBDFC ,¬BB, EB, RQD	-(C)	5,1(B)	>(R)	-(C)	>(R)	-(C)
R,RQ,FMD|EBDFC ,¬BB, EB, EC,¬RQD	-(C)	1(B)	>(R)	-(C)	>(R)	-(C)
R,RQ,CANCEL , EB	-(C)	1(B)	>(R)	-(C)	1(B)	-(C)
R,RQ,FMD ,¬BB, EB, ¬EC	-(C)	5,1	>(R)	-(C)	>(R)	-(C)
R,RQ,FMD ,¬BB,¬EB, BC	-(C)	-	>(R)	-(C)	>(R)	-(C)
R,RQ,FMD , ¬BC, RQD	-(B)	-	-	-(C)	-	-
R,RQ,FMD , ¬BC, EC,¬RQD	-(B)	-	-	-(C)	1(B)	-
R,RQ,CANCEL , ¬EB	-(B)	-	-	-(C)	2	-
R,RQ,RTR	-	-(C)	-(C)	-(C)	>(R)	-(C)
R,RQ,OTHERDFC ,¬BB,¬EB	-	-	-	-	>(R)	-
S,+RSP,¬RTR,CT(¬BB, EB)	-	-	-	-	1(B)	-
S,-RSP,¬RTR,CT(¬BB, EB)	-	-	-	-	2	-
S,+RSP, RTR	3	-	-	-	-	-
'RESET_BETB' /* FROM DFC_RESET */	-	1	1	1	1	1
'RESET_INB' /* FROM DFC_RESET */	2	-	2	2	2	2

MULTIPLE_ACTION_CODE	DETERMINING CONDITION
1	SCB.BRKT_TERM_RULE=CONDITIONAL
2	SCB.BRKT_TERM_RULE=UNCONDITIONAL

OUTPUT CODE	FUNCTION
B	CALL #FSM_HDX('BETB'); /* PAGE 5-82 TO PAGE 5-84 */
C	IF SDI=¬SD & #FSM_CHAIN_RCV¬=PURGE THEN /* PAGE 5-72 */ CALL CHANGE_MU_TO_EXR(X'080B'); /* BRACKET CONTENTION ERROR */
IR	CALL #FSM_HDX('INB_RCV'); /* PAGE 5-82 TO PAGE 5-84 */
IS	CALL #FSM_HDX('INB_SEND'); /* PAGE 5-82 TO PAGE 5-84 */
S	SEND_CHECK_SENSE=X'2003'; /* BRACKET STATE ERROR */
R	RECEIVE_CHECK_SENSE=X'2003'; /* BRACKET STATE ERROR */

END FSM_BSM_BIDDER;

```
FSM_BSM_FSP: FSM_DEFINITION CONTEXT(SCB),

MULTIPLE_ACTION_CODES(2),,,;

                                                                        /*
```

FUNCTION: THIS FSM ENFORCES THE BRACKETS PROTOCOL FOR THE FIRST SPEAKER. SEE "BRACKETS PROTOCOL" ON PAGE 5-14 FOR PROSE DESCRIPTION.

THE TWO PRIMARY STATES IN THIS FSM ARE:

- BETB (BETWEEN BRACKETS): THIS STATE INDICATES NO BRACKET IS CURRENTLY BEING PROCESSED. ANY REQUEST CHAINS SENT OR RECEIVED IN THIS STATE (WITH THE EXCEPTION OF SOME DFC REQUESTS) MUST HAVE THE BBI (BEGIN BRACKET INDICATOR) SET.

- INB (IN BRACKET): THIS STATE INDICATES A BRACKET IS CURRENTLY BEING PROCESSED. A REQUEST WITH BBI SET HAS PREVIOUSLY BEEN SENT OR RECEIVED TO BEGIN THE BRACKET. REQUEST CHAINS SENT IN THIS STATE DO NOT HAVE BBI SET. HOWEVER, THEY MAY HAVE EBI (END BRACKET INDICATOR) SET IF IT IS DESIRED THAT THE BRACKET BE ENDED.

THE REST OF THE STATES IN THIS FSM ARE TRANSITION STATES.

- PEND_BB (PENDING SENDING BEGIN BRACKET): THIS STATE IS ENTERED WHEN A POSITIVE RESPONSE TO BID IS SENT OR A POSITIVE RESPONSE TO RTR IS RECEIVED. IT MEANS THE BIDDER HAS BEEN GRANTED THE RIGHT TO START A BRACKET (BY SENDING BB).

- PEND_INB (PENDING ENTERING IN-BRACKET STATE): THIS STATE IS ENTERED WHEN THE FIRST SPEAKER RECEIVES A BB REQUEST WHILE IN BETB. THE BIDDER IS REQUESTING PERMISSION TO BEGIN A BRACKET. PERMISSION IS GRANTED (BY THE FIRST SPEAKER) BY SENDING A POSITIVE RESPONSE TO THE BB REQUEST. THIS CAUSES A TRANSITION TO THE IN-BRACKET STATE (INB). PERMISSION IS DENIED BY SENDING A NEGATIVE RESPONSE TO THE BB REQUEST. THIS CAUSES A TRANSITION BACK TO BETWEEN-BRACKETS STATE (BETB).

- PEND_TERM_S AND PEND_TERM_R (PENDING TERMINATION OF THE BRACKET): THESE STATES ARE ENTERED FROM THE IN-BRACKETS STATE (INB) WHEN A REQUEST CARRYING EB (END BRACKET) IS SENT OR RECEIVED. THE BRACKET IS TERMINATED (TRANSITION MADE TO BETWEEN-BRACKETS STATE (BETB)) IF THE END CHAIN REQUEST OF THE EB CHAIN DID NOT ASK FOR DEFINITE RESPONSE OR, A POSITIVE RESPONSE IS RECEIVED FOR THE EB CHAIN. THE BRACKET IS NOT TERMINATED (TRANSITION MADE BACK TO IN-BRACKET STATE (INB)) IF EB CHAIN IS CANCELED (USING DFC CANCEL REQUEST) OR A NEGATIVE RESPONSE TO THE EB CHAIN IS RECEIVED.

WHEN THE BRACKETS AND HALF-DUPLEX FLIP FLOP PROTOCOLS (SEE "SEND/RECEIVE MODE PROTOCOLS" ON PAGE 5-12) ARE USED, A TIGHT COUPLING EXISTS BETWEEN THE FSM'S, FSM_BSM_FSP AND FSM_HDX_FF (HALF-DUPLEX FLIP FLOP FSM, PAGE 5-84). A STRONG COORDINATION OF THE STATES OF THESE TWO FSM'S IS NECESSARY. THE GENERAL RULES ARE:

- WHENEVER FSM_BSM_FSP IS IN BETWEEN BRACKETS STATE (BETB), FSM_HDX_FF IS IN ONE OF ITS CONTENTION STATES (CONT, CONT_SEND, OR CONT_RCV).

- WHEN FSM_BSM_FSP GOES TO IN-BRACKET STATE (INB), FSM_HDX_FF GOES TO SEND (SEND) OR RECEIVE (RCV) STATE. THE CDI BIT (CHANGE DIRECTION INDICATOR) DETERMINES WHETHER IT IS SEND OR RECEIVE STATE.

IN ORDER TO FOLLOW THESE RULES FSM_BSM_FSP TELLS FSM_HDX_FF WHEN TO GO TO CONTENTION, SEND, AND RECEIVE STATES. THIS IS DONE BY CALLING FSM_HDX_FF AND GIVING IT THE SIGNAL INPUTS BETB, INB_RCV, AND INB_SEND.

- BETB SIGNAL MEANS FSM_BSM_FSP IS GOING TO BETWEEN-BRACKETS STATE (BETB) AND FSM_HDX_FF IS TO GO TO CONTENTION STATE (CONT). NOTICE THAT THIS SIGNAL IS GIVEN TO FSM_HDX_FF ON END CHAIN REQUESTS. THIS IS BECAUSE THE CONTENTION STATE (CONT) MUST BE ENTERED ONLY WHEN BETWEEN SENDING OR RECEIVING CHAINS.

- INB_SEND SIGNAL MEANS FSM_BSM_FSP IS GOING TO IN BRACKET-STATE (INB) AND FSM_HDX_FF IS TO GO TO SEND STATE (SEND).

- INB_RCV SIGNAL MEANS FSM_BSM_FSP IS GOING TO IN BRACKET-STATE (INB) AND FSM_HDX_FF IS TO GO TO RECEIVE STATE (RCV).

NOTE: RECEIVED AND SENT RESPONSES COME TO THIS FSM FROM FSM_CONTROL_BSM_RSP_SEND (PAGE 5-74) AND FSM_CONTROL_BSM_RSP_RCV (PAGE 5-73). THESE FSM'S ALLOW RESPONSES TO COME TO FSM_BSM_FSP ONLY WHEN THE RESPONSE IS FOR THE CURRENT CHAIN AND THE LAST REQUEST OF THE CURRENT CHAIN HAS BEEN SENT OR RECEIVED. THESE CONTROL FSM'S RELIEVE FSM_BSM_FSP FROM HAVING STATES TO REMEMBER WHEN A NEGATIVE RESPONSE IS RECEIVED WHILE IN THE MIDDLE OF SENDING A CHAIN.

REFERENCED BY THE FOLLOWING PROCEDURE(S):
FSM_HDX_FF

PAGE 5-84

```
                                                                        */
```

INPUT \ STATE NAMES ----->	BETB 1	INB 2	PEND BB 3	PEND INB 4	PEND TERM_R 5	PEND TERM_S 6
S,RQ,FMD|LUSTAT, BB, EB	-	>(S)	>(S)	>(S)	>(S)	>(S)
S,RQ,FMD|LUSTAT, BB,¬EB, CD	2(IR)	>(S)	>(S)	>(S)	>(S)	>(S)
S,RQ,FMD|LUSTAT, BB,¬EB, EC, ¬CD	2(IS)	>(S)	>(S)	>(S)	>(S)	>(S)
S,RQ,FMD|LUSTAT, BB,¬EB, ¬EC	2(IS)	>(S)	>(S)	>(S)	>(S)	>(S)
S,RQ,FMD|EBDFC ,¬BB, EB, RQD	>(S)	6,1(B)	>(S)	>(S)	>(S)	>(S)
S,RQ,FMD|EBDFC ,¬BB, EB, EC,¬RQD	>(S)	1(B)	>(S)	>(S)	>(S)	>(S)
S,RQ,CANCEL , EB	>(S)	1(B)	>(S)	>(S)	>(S)	1(B)
S,RQ,FMD ,¬BB, EB, ¬EC	>(S)	6,1	>(S)	>(S)	>(S)	>(S)
S,RQ,FMD ,¬BB,¬EB, BC	>(S)	-	>(S)	>(S)	>(S)	>(S)
S,RQ,FMD , ¬BC, RQD	-(B)	-	>(S)	-	>(S)	-
S,RQ,FMD , ¬BC, EC,¬RQD	-(B)	-	>(S)	-	>(S)	1(B)
S,RQ,CANCEL , ¬EB	-(B)	-	>(S)	-	>(S)	2
S,RQ,RTR	-	>(S)	>(S)	>(S)	>(S)	>(S)
S,RQ,OTHERDFC ,¬BB,¬EB	-	-	-	>(S)	>(S)	>(S)
R,+RSP,¬RTR,CT(¬BB, EB)	-	-	-	-	-	1(B)
R,-RSP,¬RTR,CT(¬BB, EB)	-	-	-	-	-	2
R,+RSP, RTR	3	-	-	-	-	-
R,RQ,FMD|LUSTAT, BB, EB	-	-(C2)	1	>(R)	>(R)	-(C1)
R,RQ,FMD|LUSTAT, BB,¬EB, RQD, CD	4	-(C2)	2(IS)	>(R)	>(R)	-(C1)
R,RQ,FMD|LUSTAT, BB,¬EB, RQD,¬CD	4	-(C2)	2(IR)	>(R)	>(R)	-(C1)
R,RQ,FMD|LUSTAT, BB,¬EB, ¬RQD, CD	>(R)	-(C2)	2(IS)	>(R)	>(R)	-(C1)
R,RQ,FMD|LUSTAT, BB,¬EB, EC,¬RQD,¬CD	>(R)	-(C2)	2(IR)	>(R)	>(R)	-(C1)
R,RQ,FMD , BB,¬EB, ¬EC	4	-(C2)	-	>(R)	>(R)	-(C1)
R,RQ,FMD|EBDFC ,¬BB, EB, RQD	-(C1)	5,1(B)	>(R)	>(R)	>(R)	-(C1)
R,RQ,FMD|EBDFC ,¬BB, EB, EC,¬RQD	-(C1)	1(B)	>(R)	>(R)	>(R)	-(C1)
R,RQ,CANCEL , EB	-(C1)	1(B)	>(R)	>(R)	1(B)	-(C1)
R,RQ,FMD ,¬BB, EB, ¬EC	-(C1)	5,1	>(R)	>(R)	>(R)	-(C1)
R,RQ,FMD ,¬BB,¬EB, BC	-(C1)	-	>(R)	>(R)	>(R)	-(C1)
R,RQ,FMD , ¬BC, RQD, CD	-(B)	-	2(IS)	-	-	-
R,RQ,FMD , ¬BC, RQD,¬CD	-(B)	-	2(IR)	-	-	-
R,RQ,FMD , ¬BC, ¬RQD, CD	-(B)	-	2(IS)	>(R)	1(B)	-
R,RQ,FMD , ¬BC, EC,¬RQD,¬CD	-(B)	-	2(IR)	>(R)	1(B)	-
R,RQ,CANCEL , ¬EB, CD	-(B)	-	2(IS)	1	2	-
R,RQ,CANCEL , ¬EB, ¬CD	-(B)	-	2(IR)	1	2	-
R,RQ,BID	-	-(C2)	-	>(R)	>(R)	-(C1)
R,RQ,OTHERDFC ,¬BB,¬EB	-	-	-	>(R)	>(R)	-
S,+RSP,FMD|LUSTAT,CT(BB,¬EB, CD)	-	-	-	2(IS)	-	-
S,+RSP,FMD|LUSTAT,CT(BB,¬EB,¬CD)	-	-	-	2(IR)	-	-
S,-RSP,FMD|LUSTAT,CT(BB,¬EB)	-	-	-	1	-	-
S,+RSP,¬BID ,CT(¬BB, EB)	-	-	-	-	1(B)	-
S,-RSP,¬BID ,CT(¬BB, EB)	-	-	-	-	2	-
S,+RSP, BID	3	-	-	-	-	-
'RESET_BETB' /* FROM DFC_RESET */	-	1	1	1	1	1
'RESET_INB' /* FROM DFC_RESET */	2	-	2	2	2	2

MULTIPLE_ACTION_CODE	DETERMINING CONDITION
1	SCB.BRKT_TERM_RULE=CONDITIONAL
2	SCB.BRKT_TERM_RULE=UNCONDITIONAL

OUTPUT CODE	FUNCTION
B	CALL #FSM_HDX('BETB'); /* PAGE 5-82 TO 5-84 */
C1	IF SDI=¬SD & #FSM_CHAIN_RCV¬=PURGE THEN /* PAGE 5-72 */ CALL CHANGE_MU_TO_EXR(X'080B'); /* BRACKET CONTENTION ERROR */
C2	IF SDI=¬SD & #FSM_CHAIN_RCV¬=PURGE THEN /* PAGE 5-72 */ CALL CHANGE_MU_TO_EXR(X'0813'); /* THE RESPONSE TO THIS EXR MAY BE */ /* EITHER 0813(REJECT-NO RTR) OR 0814 */ /* (REJECT-RTR). */
IR	CALL #FSM_HDX('INB_RCV'); /* PAGE 5-82 TO 5-84 */
IS	CALL #FSM_HDX('INB_SEND'); /* PAGE 5-82 TO 5-84 */
S	SEND_CHECK_SENSE=X'2003'; /* BRACKET STATE ERROR */
R	RECEIVE_CHECK_SENSE=X'2003'; /* BRACKET STATE ERROR */

END FSM_BSM_PSP;

```
FSM_CHAIN_RCV: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO ENFORCE THE CHAINING PROTOCOL FOR RECEIVED CHAINS.  SEE "CHAINING │
│                 PROTOCOL" ON PAGE 5-8 FOR PROSE DESCRIPTION.  THE STATES ARE:        │
│                                                                          │
│                 •  BETC (BETWEEN-CHAINS STATE): MEANS NOT CURRENTLY IN THE PROCESS   │
│                    OF RECEIVING A  CHAIN.  THE NEXT REQUEST RU  RECEIVED MUST HAVE   │
│                    THE BEGIN CHAIN INDICATOR (BCI) SET.                  │
│                                                                          │
│                 •  INC  (IN-CHAIN  STATE):  MEANS  CURRENTLY  IN  THE  PROCESS  OF   │
│                    RECEIVING A CHAIN.   THE CHAIN IS ENDED WHEN A  REQUEST RU WITH   │
│                    THE END CHAIN INDICATOR (ECI) SET IS RECEIVED.        │
│                                                                          │
│                 •  PURGE  (PURGING-CHAIN  STATE):  MEANS  HAVE  SENT  A  NEGATIVE    │
│                    RESPONSE WHILE IN THE PROCESS OF RECEIVING A CHAIN.  THIS STATE   │
│                    IS USED TO PURGE (DISCARD) THE REMAINING REQUESTS IN THE CHAIN.   │
│                    PURGING  STOPS WHEN  A REQUEST  RU  WITH EC  IS RECEIVED.   SEE   │
│                    RCV_DISCARD_CHECKS PROCEDURE ON PAGE 5-54.            │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 FSM_CONTROL_HDX_RSP_SEND_ERP_IM     PAGE 5-80            │
│                 FSM_HDX_CONT_WINNER                 PAGE 5-83            │
│                 FSM_HDX_PF                          PAGE 5-84            │
│                 FSM_RES                             PAGE 5-89            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

┌──────────────────────────────────┬──────────┬──────────┬──────────┐
│                  STATE NAMES----->│ BETC     │ INC      │ PURGE    │
│ INPUTS                           │ 01       │ 02       │ 03       │
├──────────────────────────────────┼──────────┼──────────┼──────────┤
│ R,RQ,¬CANCEL, BC, EC             │ -        │ >(R)     │ >(R)     │
│ R,RQ,¬CANCEL, BC,¬EC             │ 2        │ >(R)     │ >(R)     │
│ R,RQ,¬CANCEL,¬BC, EC             │ >(R)     │ 1        │ 1        │
│ R,RQ,¬CANCEL,¬BC,¬EC             │ >(R)     │ -        │ -        │
├──────────────────────────────────┼──────────┼──────────┼──────────┤
│ R,RQ, CANCEL                     │ >(R)     │ 1        │ 1        │
├──────────────────────────────────┼──────────┼──────────┼──────────┤
│ S,-RSP,TO_CURRENT_CHAIN          │ -        │ 3        │ -        │
├──────────────────────────────────┼──────────┼──────────┼──────────┤
│ 'RESET' /* FROM DFC_RESET */     │ -        │ 1        │ 1        │
└──────────────────────────────────┴──────────┴──────────┴──────────┘

┌──────────┬──────────────────────────────────────────────────────────┐
│ OUTPUT   │ FUNCTION                                                 │
│  CODE    │                                                          │
├──────────┼──────────────────────────────────────────────────────────┤
│    R     │ RECEIVE_CHECK_SENSE=X'2002';/* CHAINING ERROR  */        │
└──────────┴──────────────────────────────────────────────────────────┘
END FSM_CHAIN_RCV;


FSM_CHAIN_SEND: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO ENFORCE THE CHAINING PROTOCOL  FOR SENDING CHAINS.  SEE "CHAINING │
│                 PROTOCOL" ON PAGE 5-8 FOR PROSE DESCRIPTION.  THE STATES ARE:        │
│                                                                          │
│                 •  BETC  (BETWEEN-CHAINS  STATE):   MEANS  NOT  CURRENTLY  IN  THE   │
│                    PROCESS OF SENDING A CHAIN.  THE NEXT REQUEST RU SENT MUST HAVE   │
│                    THE BEGIN CHAIN INDICATOR (BCI) SET.                  │
│                                                                          │
│                 •  INC  (IN-CHAIN  STATE):   MEANS CURRENTLY  IN  THE  PROCESS  OF   │
│                    SENDING A CHAIN.  THE CHAIN IS ENDED WHEN A REQUEST RU WITH THE   │
│                    END CHAIN INDICATOR (ECI) SET IS SENT.                │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

┌──────────────────────────────────┬──────────┬──────────┐
│                  STATE NAMES----->│ BETC     │ INC      │
│ INPUTS                           │ 01       │ 02       │
├──────────────────────────────────┼──────────┼──────────┤
│ S,RQ,¬CANCEL, BC, EC             │ -        │ >(S)     │
│ S,RQ,¬CANCEL, BC,¬EC             │ 2        │ >(S)     │
│ S,RQ,¬CANCEL,¬BC, EC             │ >(S)     │ 1        │
│ S,RQ,¬CANCEL,¬BC,¬EC             │ >(S)     │ -        │
├──────────────────────────────────┼──────────┼──────────┤
│ S,RQ, CANCEL                     │ >(S)     │ 1        │
├──────────────────────────────────┼──────────┼──────────┤
│ 'RESET' /* FROM DFC_RESET */     │ -        │ 1        │
└──────────────────────────────────┴──────────┴──────────┘

┌──────────┬──────────────────────────────────────────────────────────┐
│ OUTPUT   │ FUNCTION                                                 │
│  CODE    │                                                          │
├──────────┼──────────────────────────────────────────────────────────┤
│    S     │ SEND_CHECK_SENSE=X'2002'; /* CHAINING ERROR  */          │
└──────────┴──────────────────────────────────────────────────────────┘
END FSM_CHAIN_SEND;
```

```
FSM_CONTROL_BSM_RSP_RCV: FSM_DEFINITION CONTEXT(SCB);
                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │     FUNCTION:  TO PASS RESPONSES TO THE BRACKET STATE MANAGER (BSM) FSM. THE │
 │                RESPONSES ARE PASSED TO BSM ONLY WHEN TWO CONDITIONS ARE SATISFIED: │
 │                1) THE END CHAIN REQUEST HAS BEEN SENT AND 2) THE RESPONSE IS TO THE │
 │                CURRENT CHAIN (NOT TO A PREVIOUS CHAIN). ALL POSITIVE AND NEGATIVE │
 │                RESPONSES TO THE CURRENT CHAIN ARE PASSED TO #FSM_BSM, BUT STATE │
 │                CHANGES ARE NOT MADE ON ALL OF THEM. THIS FSM MAY FOLLOW A DIRECT │
 │                CALL TO #FSM_BSM AND MAY REINVOKE #FSM_BSM.           │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES-----> / INPUTS	RESET 1	INC 2	RSP RCVD 3
S,RQ,¬EC	2	-	-
S,RQ, EC	-	1	1(A3)
R,+RSP,TO_CURRENT_CHAIN	-(A1)	-	-
R,-RSP,TO_CURRENT_CHAIN	-(A1)	3(A2)	-
'RESET' /* FROM DFC_RESET */	-	1	1

OUTPUT CODE	FUNCTION
A1	CALL #FSM_BSM; /* PAGE 5-68 OR 5-70 */
A2	SNC_BSM_RCVD=SNC; /* SAVE THE SENSE */
A3	MU_PTR_SAVE=MU_PTR; /* SAVE CURRENT MSG UNIT PTR */ CREATE MU; /* CREATE A TEMPORARY MSG UNIT */ MUCB.DIRECTION=RECEIVE; RRI=RSP; /* BUILD A ... */ RU_CTGY=FMD; /* ... RSP IN ... */ RTI=NEG; /* ... TEMPORARY ... */ SDI=SD; /* ... MSG UNIT */ SNC=SNC_BSM_RCVD; /* USE SAVED SENSE */ CALL #FSM_BSM; /* CALL BSM WITH TEMPORARY RSP */ /* (PAGE 5-68 OR 5-70) */ DISCARD MU; /* DISCARD TEMPORARY RSP */ MU_PTR=MU_PTR_SAVE; /* RESTORE CURRENT MSG UNIT */

```
END FSM_CONTROL_BSM_RSP_RCV;
```

```
FSM_CONTROL_BSM_RSP_SEND: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO  PASS RESPONSES  TO THE  BRACKET  STATE MANAGER  (BSM) FSM.   THE │
│                 RESPONSES ARE PASSED TO BSM ONLY  WHEN TWO CONDITIONS ARE SATISFIED: │
│                 1) THE END CHAIN REQUEST HAS BEEN RECEIVED AND 2) THE RESPONSE IS TO │
│                 THE  CURRENT CHAIN  (NOT TO  A  PREVIOUS CHAIN).   ALL POSITIVE  AND │
│                 NEGATIVE RESPONSES TO THE CURRENT CHAIN  ARE PASSED TO #FSM_BSM, BUT │
│                 STATE CHANGES ARE  NOT MADE ON ALL  OF THEM.  THIS FSM  MAY FOLLOW A │
│                 DIRECT CALL TO #FSM_BSM AND MAY REINVOKE #FSM_BSM.                  │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES-----> / INPUTS	RESET 1	INC 2	RSP SENT 3
R,RQ,¬EC	2	-	-
R,RQ, EC	-	1	1(A3)
S,+RSP,TO_CURRENT_CHAIN	-(A1)	-	-
S,-RSP,TO_CURRENT_CHAIN	-(A1)	3(A2)	-
'RESET' /* FROM DFC_RESET */	-	1	1

OUTPUT CODE	FUNCTION
A1	CALL #FSM_BSM; /* PAGE 5-68 OR 5-70 */
A2	SNC_BSM_SENT=SNC; /* SAVE THE SENSE */
A3	MU_PTR_SAVE=MU_PTR; /* SAVE CURRENT MSG UNIT PTR */ CREATE MU; /* CREATE A TEMPORARY MSG UNIT */ MUCB.DIRECTION=SEND; RRI=RSP; /* BUILD A ... */ RU_CTGY=FMD; /* ... RSP IN ... */ RTI=NEG; /* ... TEMPORARY ... */ SDI=SD; /* ... MSG UNIT */ SNC=SNC_BSM_SENT; /* USE SAVED SENSE */ CALL #FSM_BSM; /* CALL BSM WITH TEMPORARY RSP */ /* (PAGE 5-68 OR 5-70) */ DISCARD MU; /* DISCARD TEMPORARY RSP */ MU_PTR=MU_PTR_SAVE; /* RESTORE CURRENT MSG UNIT */

```
END FSM_CONTROL_BSM_RSP_SEND;
```

```
FSM_CONTROL_HDX_RSP_RCV: FSM_DEFINITION CONTEXT(SCB);
```

```
/*
     FUNCTION:  TO PASS  RESPONSES TO THE  HALF-DUPLEX MANAGER FSM  (#FSM_HDX).  THE
                RESPONSES ARE  PASSED TO #FSM_HDX  ONLY WHEN BETWEEN  CHAINS.  I.E.,
                #FSM_HDX NEVER  GETS A  RESPONSES WHILE  IN THE  MIDDLE OF  A CHAIN.
                ONLY NEGATIVE RESPONSES  ARE PASSED TO #FSM_HDX.  STATE CHANGES ARE
                NOT  NECESSARILY MADE  BY #FSM_HDX  ON ALL  THE NEGATIVE  RESPONSES.
                POSITIVE  RESPONSES ARE  NOT  PASSED TO  #FSM_HDX  BECAUSE NO  STATE
                CHANGES ARE EVER MADE ON THEM.  THIS FSM MAY FOLLOW A DIRECT CALL TO
                #FSM_HDX AND MAY REINVOKE #FSM_HDX.

     REFERENCED BY THE FOLLOWING PROCEDURE(S):
                FSM_HDX_FF                              PAGE 5-84
*/
```

STATE NAMES----->/ INPUTS	RESET / 1	INC / 2	RSP RCVD / 3
S,RQ,¬EC	2	-	-
S,RQ, EC	-	1	1(A3)
R,-RSP,¬(CT(BB)&CT(EB))	-(A1)	3(A2)	-
'RESET' /* FROM DFC_RESET */	-	1	1

OUTPUT CODE	FUNCTION
A1	CALL #FSM_HDX; /* PAGE 5-82 TO &CA5F13 */
A2	SNC_HDX_RCVD=SNC; /* SAVE THE SENSE */
A3	MU_PTR_SAVE=MU_PTR; /* SAVE CURRENT MSG UNIT PTR */ CREATE MU; /* CREATE A TEMPORARY MSG UNIT */ MUCB.DIRECTION=RECEIVE;/* SET DIRECTION AS RECEIVE */ RRI=RSP; /* BUILD A ... */ RU_CTGY=FMD; /* ... RSP IN ... */ RTI=NEG; /* ... TEMPORARY ... */ SDI=SD; /* ... MSG UNIT */ SNC=SNC_HDX_RCVD; /* USE SAVED SENSE */ CALL #FSM_HDX; /* FSM GETS TEMP RSP */ /* (PAGE 5-82 TO 5-84) */ DISCARD MU; /* DISCARD TEMPORARY RSP */ MU_PTR=MU_PTR_SAVE; /* RESTORE CURRENT MSG UNIT */

```
END FSM_CONTROL_HDX_RSP_RCV;
```

```
FSM_CONTROL_HDX_RSP_RCV_ERP_DL: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO IDENTIFY  THE ERP SYNCHRONIZATION  EVENT (THE RESPONSE  TO CHASE)  │
│                THAT  MARKS THE  POINT AT  WHICH THE  HDX FSM   IS TO  MAKE ITS  ERP  │
│                TRANSITION.  WHEN THE  EVENT OCCURS,  THIS FSM  CALLS FSM_HDX_FF  TO  │
│                CAUSE  THE  ERP  TRANSITION.   WHEN A  NEGATIVE  RESPONSE  HAS  BEEN  │
│                RECEIVED,  BUT CHASE  HAS NOT  BEEN SENT,  THE SENSE  CODE FROM  THE  │
│                RESPONSE IS  SAVED SO THAT  WHEN THE  SYNC EVENT OCCURS  A TEMPORARY  │
│                RESPONSE CAN BE USED TO CALL FSM_HDX_FF WITH THE CORRECT SENSE CODE.  │
│                AS  WITH  THE  OTHER FSM_CONTROL_HDX_RSP_XXXX  MACHINES,  THIS  FSM  │
│                SERIALIZES  VARIOUS  RACE  CONDITIONS THAT  CAN  OCCUR  AND  SHIELDS  │
│                FSM_HDX_FF FROM THE  COMPLEXITIES THAT THEY CREATE.   THE BASIC IDEA  │
│                IS TO PRESENT ALL RESPONSES TO FSM_HDX_FF  AT THE END OF A PERIOD OF  │
│                ACTIVITY  SO  THAT FSM_HDX_FF  WILL  NOT  HAVE TO  CONTAIN  NUMEROUS  │
│                PENDING STATES  OR COMPLICATED CHECKING  OF PENDING STATES  IN OTHER  │
│                FSMS.                                                                 │
│                                                                                      │
│                THERE ARE THREE MAJOR STATES IN THIS FSM:                             │
│                                                                                      │
│                •  1&2: RESET                                                         │
│                                                                                      │
│                •  3&6: THE SYNC EVENT HAS BEEN SENT BUT ITS RESPONSE HAS NOT BEEN    │
│                   RECEIVED.                                                          │
│                                                                                      │
│                •  4&5: A NEGATIVE RESPONSE HAS BEEN  RECEIVED, BUT THE SYNC EVENT    │
│                   HAS NOT YET BEEN SENT.                                             │
│                                                                                      │
│                THIS FSM  IS USED ONLY WHEN  SYMMETRIC ERROR RECOVERY IS  BEING USED  │
│                AND THIS HALF-SESSION IS USING DELAYED REQUEST MODE.                  │
│                                                                                      │
│     REFERS TO THE FOLLOWING PROCEDURE(S):                                            │
│                FSM_HDX_FF                          PAGE 5-84                         │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

┌──────────────────────┬──────────┬──────────┬──────────┬──────────┬──────────┬──────────┐
│ STATE NAMES----->    │ RESET    │ RESET    │ SENTSE   │ WAITSE   │ WAITSE   │ CMSSE    │
│                      │ BETC     │ INC      │ BETC     │ BETC     │ INC      │ BETC     │
│                      │          │          │ CHASE    │          │          │ CHASE    │
│                      │          │          │          │ -RSP     │ -RSP     │ -RSP     │
│ INPUTS               │ 1        │ 2        │ 3        │ RCVD 4   │ RCVD 5   │ RCVD 6   │
├──────────────────────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────────┤
│ S,RQ,¬EC             │ 2        │ -        │ >(S)     │ >(S)     │ >(S)     │ >(S)     │
│ S,RQ, EC,¬CHASE      │ -        │ 1        │ >(S)     │ >(S)     │ 4        │ >(S)     │
│ S,RQ,     CHASE      │ 3        │ /NOTE    │ >(S)     │ 6        │ /NOTE    │ >(S)     │
├──────────────────────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────────┤
│ R,+RSP,¬CHASE        │ -        │ -        │ -        │ -        │ -        │ -        │
│ R,-RSP,¬CHASE,¬(CT(BB)&CT(EB)) │ 4(A2) │ 5(A2) │ 6(A2)  │ -        │ -        │ -        │
│ R,+RSP, CHASE        │ -        │ -        │ 1        │ -        │ -        │ 1(A3)    │
│ R,-RSP, CHASE        │ -        │ -        │ 1(A1)    │ -        │ -        │ 1(A3)    │
├──────────────────────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────────┤
│ 'RESET' /* FROM DFC_RESET */ │ -    │ 1     │ 1        │ 1        │ 1        │ 1        │
└──────────────────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┘

┌──────────┬──────────────────────────────────────────────────────────────┐
│ OUTPUT   │ FUNCTION                                                     │
│  CODE    │                                                              │
├──────────┼──────────────────────────────────────────────────────────────┤
│   A1     │ CALL FSM_HDX_FF;       /* PAGE 5-84                      */ │
├──────────┼──────────────────────────────────────────────────────────────┤
│   A2     │ SNC_HDX_RCVD=SNC;      /* SAVE THE SENSE                 */ │
├──────────┼──────────────────────────────────────────────────────────────┤
│   A3     │ MU_PTR_SAVE=MU_PTR;    /* SAVE CURRENT MSG UNIT PTR      */ │
│          │ CREATE MU;             /* CREATE A TEMPORARY MSG UNIT    */ │
│          │ MUCB.DIRECTION=RECEIVE;/* SET DIRECTION AS RECEIVE       */ │
│          │ RRI=RSP;               /* BUILD A ...                    */ │
│          │ RU_CTGY=FMD;           /* ... RSP IN ...                 */ │
│          │ RTI=NEG;               /* ... TEMPORARY ...              */ │
│          │ SDI=SD;                /* ... MSG UNIT                   */ │
│          │ SNC=SNC_HDX_RCVD;      /* USE SAVED SENSE                */ │
│          │ CALL FSM_HDX_FF;       /* FSM GETS TEMP RSP(PAGE 5-84)   */ │
│          │ DISCARD MU;            /* DISCARD TEMPORARY RSP          */ │
│          │ MU_PTR=MU_PTR_SAVE;    /* RESTORE CURRENT MSG UNIT       */ │
├──────────┼──────────────────────────────────────────────────────────────┤
│   S      │ SEND_CHECK_SENSE=X'200C'; /* ERP SYNC STATE ERROR        */ │
└──────────┴──────────────────────────────────────────────────────────────┘
 /* NOTE: THIS CONDITION DETECTED AS SEND ERROR BY FSM_CHAIN_SEND (PAGE 5-72) */

END FSM_CONTROL_HDX_RSP_RCV_ERP_DL;
```

```
FSM_CONTROL_HDX_RSP_RCV_ERP_IM: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO IDENTIFY  THE ERP SYNCHRONIZATION  EVENT (SYNC EVENT  RESPONSE TO
│                RQD OR RQE WITH CD) THAT MARKS THE POINT AT WHICH THE HDX FSM  IS TO
│                MAKE ITS  ERP TRANSITION.  WHEN THAT  EVENT OCCURS,  THIS FSM  CALLS
│                FSM_HDX_FF TO CAUSE THE ERP TRANSITION.   WHEN A NEGATIVE SYNC EVENT
│                RESPONSE IS RECEIVED AND DFC IS BETWEEN CHAINS, FSM_HDX_FF IS CALLED
│                IMMEDIATELY.  WHEN A  RESPONSE IS RECEIVED AND A  SYNC EVENT REQUEST
│                (RQD OR RQE,CD)  HAS  NOT BEEN SENT THE SENSE CODE  FROM THE RESPONSE
│                IS  SAVED  SO THAT  WHEN  THE  SYNC  EVENT  RESPONSE IS  RECEIVED  A
│                TEMPORARY RESPONSE CAN  BE USED TO CALL FSM_HDX_FF  WITH THE CORRECT
│                SENSE CODE.   AS WITH THE OTHER  FSM_CONTROL_HDX_RSP_XXXX  MACHINES,
│                THIS  FSM SERIALIZES  VARIOUS  RACE CONDITIONS  THAT  CAN OCCUR  AND
│                SHIELDS  FSM_HDX_FF FROM  THE COMPLEXITIES  THAT  THEY CREATE.   THE
│                BASIC IDEA IS TO PRESENT ALL RESPONSES TO FSM_HDX_FF AT THE END OF A
│                PERIOD  OF ACTIVITY  SO THAT  FSM_HDX_FF  WILL NOT  HAVE TO  CONTAIN
│                NUMEROUS PENDING STATES OR COMPLICATED CHECKING OF PENDING STATES IN
│                OTHER FSMS.
│
│                THERE ARE THREE MAJOR STATES IN THIS FSM:
│
│                •  1&2:  RESET
│
│                •  3,4,8,&9:  THE  SYNC EVENT HAS BEEN  SENT BUT ITS  RESPONSE HAS
│                   NOT BEEN RECEIVED.  THIS RESPONSE MAY BE  POSITIVE OR NEGATIVE.
│                   IF THE  SYNC EVENT WAS RQE  WITH CD THEN THE  POSITIVE RESPONSE
│                   MAY BE IMPLIED BY THE RECEIPT OF  A REQUEST.  THIS MAY OCCUR IN
│                   STATE 3.
│
│                •  5,6,&7:  A  NEGATIVE RESPONSE HAS  BEEN RECEIVED, BUT  THE SYNC
│                   EVENT HAS NOT YET BEEN SENT.
│
│                THIS FSM  IS USED ONLY WHEN  SYMMETRIC ERROR RECOVERY IS  BEING USED
│                AND THIS HALF-SESSION IS USING IMMEDIATE REQUEST MODE.
│
│     REFERS TO THE FOLLOWING PROCEDURE(S):
│                FSM_HDX_FF                          PAGE 5-84
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----->	RESET BETC	RESET INC	SENTSE BETC NOTCAN	SENTSE BETC CANCEL	WAITSE BETC -RSP RCVD	WAITSE INC -RSP RCVD CURCHN	WAITSE INC -RSP RCVD PRECHN	SENTSE BETC NOTCAN -RSP RCVD	SENTSE BETC CANCEL -RSP RCVD
INPUTS	1	2	3	4	5	6	7	8	9
S,RQ,¬CANCEL,¬EC	2	-	>(S)	>(S)	>(S)	>(S)	>(S)	>(S)	>(S)
S,RQ,¬CANCEL, EC,¬(RQD|(RQE&CD))	-	1	>(S)	>(S)	>(S)	>(S)	>(S)	>(S)	>(S)
S,RQ,¬CANCEL, EC, (RQD|(RQE&CD))	3	3	>(S)	>(S)	8	1(A3)	8	>(S)	>(S)
S,RQ, CANCEL	/NOTE1	4	/NOTE1	/NOTE2	/NOTE1	9	9	/NOTE1	/NOTE1
R,+RSP,¬CANCEL, TO_CURRENT_CHAIN	-	-	1	-	-	-	-	1(A3)	-
R,-RSP,¬CANCEL, TO_CURRENT_CHAIN, ¬(CT(BB)&CT(EB))	5(A2)	6(A2)	1(A1)	9(A2)	-	-	6	1(A3)	-
R,-RSP,¬CANCEL,¬TO_CURRENT_CHAIN, ¬(CT(BB)&CT(EB))	5(A2)	7(A2)	8(A2)	9(A2)	-	-	-	-	-
R,+RSP, CANCEL	-	-	-	1	-	-	-	-	1(A3)
R,-RSP, CANCEL	-	-	-	1(A1)	-	-	-	-	1(A3)
R,RQ 'RESET' /* FROM DFC_RESET */	- -	/NOTE2 1	1 1	>(R) 1	/NOTE2 1	/NOTE2 1	/NOTE2 1	>(R) 1	>(R) 1

OUTPUT CODE	FUNCTION	
A1	CALL FSM_HDX_FF;	/* PAGE 5-84 */
A2	SNC_HDX_RCVD=SNC;	/* SAVE THE SENSE */
A3	MU_PTR_SAVE=MU_PTR;	/* SAVE CURRENT MSG UNIT PTR */
	CREATE MU;	/* CREATE A TEMPORARY MSG UNIT */
	MUCB.DIRECTION=RECEIVE;	/* SET DIRECTION AS RECEIVE */
	RRI=RSP;	/* BUILD A ... */
	RU_CTGY=FMD;	/* ... RSP IN ... */
	RTI=NEG;	/* ... TEMPORARY ... */
	SDI=SD;	/* ... MSG UNIT */
	SNC=SNC_HDX_RCVD;	/* USE SAVED SENSE */
	CALL FSM_HDX_FF;	/* FSM GETS TEMP RSP(PAGE 5-84) */
	DISCARD MU;	/* DISCARD TEMPORARY RSP */
	MU_PTR=MU_PTR_SAVE;	/* RESTORE CURRENT MSG UNIT */
S	SEND_CHECK_SENSE=X'200C';	/* ERP SYNC STATE ERROR */
R	RECEIVE_CHECK_SENSE=X'200C';	/* ERP SYNC STATE ERROR */

```
/* NOTE1: THIS CONDITION IS DETECTED AS SEND ERROR BY FSM_CHAIN_SEND (PAGE 5-72) */
/* NOTE2: THIS CONDITION IS DETECTED AS RECEIVE ERROR BY FSM_CHAIN_RCV (PAGE 5-72)  */

END FSM_CONTROL_HDX_RSP_RCV_ERP_IM;
```

```
FSM_CONTROL_HDX_RSP_SEND: FSM_DEFINITION CONTEXT(SCB);
                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO PASS  RESPONSES TO THE  HALF-DUPLEX MANAGER FSM  (#FSM_HDX).  THE │
│                 RESPONSES ARE  PASSED TO #FSM_HDX  ONLY WHEN BETWEEN  CHAINS.  I.E.,  │
│                 #FSM_HDX NEVER GETS A RESPONSE WHILE IN THE MIDDLE OF A CHAIN.  ONLY  │
│                 NEGATIVE RESPONSES  ARE PASSED TO  #FSM_HDX.  STATE CHANGES  ARE NOT  │
│                 NECESSARILY  MADE  BY  #FSM_HDX  ON   ALL  THE  NEGATIVE  RESPONSES.  │
│                 POSITIVE  RESPONSES ARE  NOT  PASSED TO  #FSM_HDX  BECAUSE NO  STATE  │
│                 CHANGES ARE  EVER MADE  ON THEM.  NOTE  THAT THIS  FSM MAY  FOLLOW A  │
│                 DIRECT CALL TO #FSM_HDX AND THAT THIS FSM MAY REINVOKE #FSM_HDX.      │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 FSM_HDX_FF                          PAGE 5-84            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

┌──────────────────────────────────────┬──────────┬──────────┬──────────┐
│                    STATE NAMES----->│ RESET    │ INC      │ RSP      │
│                                      │          │          │ SENT     │
│ INPUTS                               │ 1        │ 2        │ 3        │
├──────────────────────────────────────┼──────────┼──────────┼──────────┤
│ R,RQ,¬EC                             │ 2        │ -        │ -        │
│ R,RQ, EC                             │ -        │ 1        │ 1(A3)    │
├──────────────────────────────────────┼──────────┼──────────┼──────────┤
│ S,-RSP,¬(CT(BB)&CT(EB))              │ -(A1)    │ 3(A2)    │ -        │
├──────────────────────────────────────┼──────────┼──────────┼──────────┤
│ 'RESET' /* FROM DFC_RESET */         │ -        │ 1        │ 1        │
└──────────────────────────────────────┴──────────┴──────────┴──────────┘

┌──────────┬──────────────────────────────────────────────────────────────┐
│ OUTPUT   │ FUNCTION                                                     │
│  CODE    │                                                              │
├──────────┼──────────────────────────────────────────────────────────────┤
│   A1     │ CALL #FSM_HDX;          /* PAGE 5-82 OR &CA5F13          */ │
├──────────┼──────────────────────────────────────────────────────────────┤
│   A2     │ SNC_HDX_SENT=SNC;       /* SAVE THE SENSE                */ │
├──────────┼──────────────────────────────────────────────────────────────┤
│   A3     │ MU_PTR_SAVE=MU_PTR;     /* SAVE CURRENT MSG UNIT PTR     */ │
│          │ CREATE MU;              /* CREATE A TEMPORARY MSG UNIT   */ │
│          │ MUCB.DIRECTION=SEND;                                         │
│          │ RRI=RSP;                /* BUILD A ...                   */ │
│          │ RU_CTGY=FMD;            /* ... RSP IN ...                */ │
│          │ RTI=NEG;                /* ... TEMPORARY ...             */ │
│          │ SDI=SD;                 /* ... MSG UNIT                  */ │
│          │ SNC=SNC_HDX_SENT;       /* USE SAVED SENSE               */ │
│          │ CALL #FSM_HDX;          /* FSM GETS TEMP RSP             */ │
│          │                         /* (PAGE 5-82 TO 5-84)           */ │
│          │ DISCARD MU;             /* DISCARD TEMPORARY RSP         */ │
│          │ MU_PTR=MU_PTR_SAVE;     /* RESTORE CURRENT MSG UNIT      */ │
└──────────┴──────────────────────────────────────────────────────────────┘
END FSM_CONTROL_HDX_RSP_SEND;
```

```
FSM_CONTROL_HDX_RSP_SEND_ERP_DL: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO IDENTIFY THE ERP SYNCHRONIZATION EVENT (THE RESPONSE TO CHASE)
│                THAT MARKS THE POINT AT WHICH THE HDX FSM IS TO MAKE ITS ERP
│                TRANSITION.  WHEN THE EVENT OCCURS, THIS FSM CALLS FSM_HDX_FF TO
│                CAUSE THE ERP TRANSITION.  WHEN A NEGATIVE RESPONSE IS SENT, THE
│                SENSE CODE IS SAVED SO THAT WHEN THE SYNC EVENT OCCURS A TEMPORARY
│                RESPONSE CAN BE USED TO CALL FSM_HDX_FF WITH THE CORRECT SENSE CODE.
│                AS WITH THE OTHER FSM_CONTROL_HDX_RSP_XXXX MACHINES, THIS FSM
│                SERIALIZES VARIOUS RACE CONDITIONS THAT CAN OCCUR AND SHIELDS
│                FSM_HDX_FF FROM THE COMPLEXITIES THAT THEY CREATE.  THE BASIC IDEA
│                IS TO PRESENT ALL RESPONSES TO FSM_HDX_FF AT THE END OF A PERIOD OF
│                ACTIVITY SO THAT FSM_HDX_FF WILL NOT HAVE TO CONTAIN NUMEROUS
│                PENDING STATES OR COMPLICATED CHECKING OF PENDING STATES IN OTHER
│                FSMS.
│
│                THERE ARE THREE MAJOR STATES IN THIS FSM:
│
│                •  1&2:  RESET
│
│                •  3&6:  THE SYNC EVENT HAS BEEN RECEIVED BUT ITS RESPONSE HAS NOT
│                   BEEN SENT.
│
│                •  4&5:  A NEGATIVE RESPONSE HAS BEEN SENT, BUT THE SYNC EVENT HAS
│                   NOT YET BEEN RECEIVED.
│
│                THIS FSM IS USED ONLY WHEN SYMMETRIC ERROR RECOVERY IS BEING USED
│                AND THE OTHER HALF-SESSION IS USING DELAYED REQUEST MODE.
│
│     REFERS TO THE FOLLOWING PROCEDURE(S):
│                FSM_HDX_FF                          PAGE 5-84
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----->	RESET BETC	RESET INC	RCVDSE BETC CHASE	WAITSE BETC -RSP SENT	WAITSE INC -RSP SENT	RCVDSE BETC CHASE -RSP SENT
INPUTS	1	2	3	4	5	6
R,RQ,¬EC	2	-	>(R)	5	-	>(R)
R,RQ, EC,¬CHASE	-	1	>(R)	-	4	>(R)
R,RQ, CHASE	3	/NOTE	>(R)	6	/NOTE	>(R)
S,+RSP,¬CHASE	-	-	-	-	-	-
S,-RSP,¬CHASE,¬(CT(BB)&CT(EB))	4(A2)	5(A2)	6(A2)	-	-	-
S,+RSP, CHASE	-	-	1	-	-	1(A3)
S,-RSP, CHASE	-	-	1(A1)	-	-	1(A3)
'RESET' /* FROM DFC_RESET */	-	1	1	1	1	1

OUTPUT CODE	FUNCTION
A1	CALL FSM_HDX_FF; /* PAGE 5-84 */
A2	SNC_HDX_SENT=SNC; /* SAVE THE SENSE */
A3	MU_PTR_SAVE=MU_PTR; /* SAVE CURRENT MSG UNIT PTR */ CREATE MU; /* CREATE A TEMPORARY MSG UNIT */ MUCB.DIRECTION=SEND; RRI=RSP; /* BUILD A ... */ RU_CTGY=FMD; /* ... RSP IN ... */ RTI=NEG; /* ... TEMPORARY ... */ SDI=SD; /* ... MSG UNIT */ SNC=SNC_HDX_SENT; /* USE SAVED SENSE */ CALL FSM_HDX_FF; /* FSM GETS TEMP RSP(PAGE 5-84) */ DISCARD MU; /* DISCARD TEMPORARY RSP */ MU_PTR=MU_PTR_SAVE; /* RESTORE CURRENT MSG UNIT */
R	RECEIVE_CHECK_SENSE=X'200C'; /* ERP SYNC STATE ERROR */

```
/* NOTE: THIS CONDITION IS DETECTED AS RECEIVE ERROR BY FSM_CHAIN_RCV (PAGE 5-78) */

END FSM_CONTROL_HDX_RSP_SEND_ERP_DL;
```

```
FSM_CONTROL_HDX_RSP_SEND_ERP_IM: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO IDENTIFY THE ERP SYNCHRONIZATION EVENT (SYNC EVENT RESPONSE TO │
│                RQD OR RQE WITH CD) THAT MARKS THE POINT AT WHICH THE HDX FSM IS TO │
│                MAKE ITS ERP TRANSITION. WHEN THAT EVENT OCCURS, THIS FSM CALLS │
│                FSM_HDX_FF TO CAUSE THE ERP TRANSITION.  WHEN A NEGATIVE SYNC EVENT │
│                RESPONSE IS SENT AND DFC IS BETWEEN CHAINS, FSM_HDX_FF IS CALLED │
│                IMMEDIATELY. WHEN A RESPONSE IS SENT AND A SYNC EVENT REQUEST (RQD │
│                OR RQE WITH CD) HAS NOT BEEN RECEIVED THE SENSE CODE FROM THE │
│                RESPONSE IS SAVED SO THAT WHEN THE SYNC EVENT RESPONSE IS SENT A │
│                TEMPORARY RESPONSE CAN BE USED TO CALL FSM_HDX_FF WITH THE CORRECT │
│                SENSE CODE.  AS WITH THE OTHER FSM_CONTROL_HDX_RSP_XXXX MACHINES, │
│                THIS FSM SERIALIZES VARIOUS RACE CONDITIONS THAT CAN OCCUR AND │
│                SHIELDS FSM_HDX_FF FROM THE COMPLEXITIES THAT THEY CREATE.  THE │
│                BASIC IDEA IS TO PRESENT ALL RESPONSES TO FSM_HDX_FF AT THE END OF A │
│                PERIOD OF ACTIVITY SO THAT FSM_HDX_FF WILL NOT HAVE TO CONTAIN │
│                NUMEROUS PENDING STATES OR COMPLICATED CHECKING OF PENDING STATES IN │
│                OTHER FSMS.                                               │
│                                                                          │
│                THERE ARE THREE MAJOR STATES IN THIS FSM:                 │
│                                                                          │
│                • 1&2:  RESET                                             │
│                • 3,4,8,&9:  THE SYNC EVENT REQUEST HAS BEEN RECEIVED BUT THE │
│                    RESPONSE TO IT HAS NOT BEEN SENT.  THIS RESPONSE MAY BE │
│                    POSITIVE OR NEGATIVE.  IF THE SYNC EVENT REQUEST WAS RQE WITH │
│                    CD THEN IT WILL BE A NEGATIVE RESPONSE.               │
│                • 5,6,&7:  A NEGATIVE RESPONSE HAS BEEN SENT, BUT THE SYNC EVENT │
│                    REQUEST HAS NOT YET BEEN RECEIVED.                    │
│                                                                          │
│                THIS FSM IS USED ONLY WHEN SYMMETRIC ERROR RECOVERY IS BEING USED │
│                AND THE OTHER HALF-SESSION IS USING IMMEDIATE REQUEST MODE. │
│                                                                          │
│     REFERS TO THE FOLLOWING PROCEDURE(S):                                │
│                FSM_CHAIN_RCV                    PAGE 5-72                │
│                FSM_HDX_FF                       PAGE 5-84                │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----->	RESET BETC	RESET INC	RCVDSE BETC NOTCAN	RCVDSE BETC CANCEL	WAITSE BETC -RSP SENT	WAITSE INC -RSP SENT CURCHN	WAITSE INC -RSP SENT PRECHN	RCVDSE BETC NOTCAN -RSP SENT	RCVDSE BETC CANCEL -RSP SENT
INPUTS	1	2	3	4	5	6	7	8	9
R,RQ,¬CANCEL,¬EC	2	-	>(R)	>(R)	7	-	-	>(R)	>(R)
R,RQ,¬CANCEL, EC,¬(RQD|(RQE&CD))	-	1	>(R)	>(R)	-	5	5	>(R)	>(R)
R,RQ,¬CANCEL, EC, RQE, CD	3	3	>(R)	>(R)	8(E)	1(A3)	8(E)	>(R)	>(R)
R,RQ,¬CANCEL, EC, RQD	3	3	>(R)	>(R)	8	1(A3)	8	>(R)	>(R)
R,RQ, CANCEL	/NOTE1	4	>(R)	>(R)	/NOTE1	9	9	>(R)	>(R)
S,+RSP,¬CANCEL, TO_CURRENT_CHAIN	-	-	1	-	-	-	-	1(A3)	-
S,-RSP,¬CANCEL, TO_CURRENT_CHAIN, ¬(CT(BB)&CT(EB))	5(A2)	6(A2)	1(A1)	9(A2)	-	-	6	1(A3)	-
S,-RSP,¬CANCEL,¬TO_CURRENT_CHAIN, ¬(CT(BB)&CT(EB))	5(A2)	7(A2)	8(A2)	9(A2)	-	-	-	-	-
S,+RSP, CANCEL	-	-	-	1	-	-	-	-	1(A3)
S,-RSP, CANCEL	-	-	-	1(A1)	-	-	-	-	1(A3)
S,RQ	-	/NOTE2	1	>(S)	/NOTE2	/NOTE2	/NOTE2	>(S)	>(S)
'RESET' /* FROM DFC_RESET */	-	1	1	1	1	1	1	1	1

OUTPUT CODE	FUNCTION	
A1	CALL FSM_HDX_FF;	/* PAGE 5-84 */
A2	SNC_HDX_SENT=SNC;	/* SAVE THE SENSE */
A3	MU_PTR_SAVE=MU_PTR;	/* SAVE CURRENT MSG UNIT PTR */
	CREATE MU;	/* CREATE A TEMPORARY MSG UNIT */
	MUCB.DIRECTION=SEND;	
	RRI=RSP;	/* BUILD A ... */
	RU_CTGY=FMD;	/* ... RSP IN ... */
	RTI=NEG;	/* ... TEMPORARY ... */
	SDI=SD;	/* ... MSG UNIT */
	SNC=SNC_HDX_SENT;	/* USE SAVED SENSE */
	CALL FSM_HDX_FF;	/* FSM GETS TEMP RSP(PAGE 5-84) */
	DISCARD MU;	/* DISCARD TEMPORARY RSP */
	MU_PTR=MU_PTR_SAVE;	/* RESTORE CURRENT MSG UNIT */
E	IF SDI¬=SD & FSM_CHAIN_RCV¬=PURGE THEN	/* PAGE 5-72 */
	CALL CHANGE_MU_TO_EXR(X'0867');	/* MUST SEND -RSP TO SYNC EVENT */
S	SEND_CHECK_SENSE=X'200C';	/* ERP SYNC STATE ERROR */
R	RECEIVE_CHECK_SENSE=X'200C';	/* ERP SYN STATE ERROR */

```
/* NOTE1: THIS CONDITION IS DETECTED AS RECEIVE ERROR BY FSM_CHAIN_RCV (PAGE 5-72) */
/* NOTE2: THIS CONDITION IS DETECTED AS SEND ERROR BY FSM_HDX_FF (PAGE 5-84) */
END FSM_CONTROL_HDX_RSP_SEND_ERP_IM;
```

```
FSM_EBCD_RCV: FSM_DEFINITION CONTEXT(SCB);

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO ENFORCE THAT EB CHAINS DO NOT HAVE CD SET ON END OF CHAIN.│
  │                                                                              │
  │      NOTE:      THE IMPLEMENTATION OF  THIS FSM IS OPTIONAL BECAUSE IT  IS ONLY USED
  │                 TO CHECK FOR RECEIVE ERROR CONDITIONS.                       │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  ┌──────────────────────────────────┬──────────┬──────────┐
  │                    STATE NAMES-->│ RESET    │ INC      │
  │ INPUTS                           │ 1        │ 2        │
  ├──────────────────────────────────┼──────────┼──────────┤
  │ R,RQ, BC,¬EC, EB                 │ 2        │ /NOTE1   │
  │ R,RQ,¬BC, EC,     CD             │ -        │ >(R)     │
  │ R,RQ,¬BC, EC,    ¬CD             │ -        │ 1        │
  ├──────────────────────────────────┼──────────┼──────────┤
  │ R,RQ,CANCEL, CD                  │ -        │ >(R)     │
  │ R,RQ,CANCEL,¬CD                  │ -        │ 1        │
  ├──────────────────────────────────┼──────────┼──────────┤
  │ 'RESET' /* FROM DFC_RESET */     │ -        │ 1        │
  └──────────────────────────────────┴──────────┴──────────┘

  ┌──────────┬──────────────────────────────────────────────┐
  │ OUTPUT   │ FUNCTION                                     │
  │  CODE    │                                              │
  ├──────────┼──────────────────────────────────────────────┤
  │    R     │ RECEIVE_CHECK_SENSE=X'400D'; /* CD NOT ALLOWED */ │
  └──────────┴──────────────────────────────────────────────┘
  /* NOTE1: THIS CONDITION IS DETECTED AS A RECEIVE ERROR BY
     FSM_CHAIN_RCV (PAGE 5-72) */

END FSM_EBCD_RCV;



FSM_EBCD_SEND: FSM_DEFINITION CONTEXT(SCB);

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO ENFORCE THAT EB CHAINS DO NOT HAVE CD SET ON END OF CHAIN.│
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  ┌──────────────────────────────────┬──────────┬──────────┐
  │                    STATE NAMES-->│ RESET    │ INC      │
  │ INPUTS                           │ 1        │ 2        │
  ├──────────────────────────────────┼──────────┼──────────┤
  │ S,RQ, BC,¬EC, EB                 │ 2        │ /NOTE    │
  │ S,RQ,¬BC, EC,     CD             │ -        │ >(S)     │
  │ S,RQ,¬BC, EC,    ¬CD             │ -        │ 1        │
  ├──────────────────────────────────┼──────────┼──────────┤
  │ S,RQ,CANCEL, CD                  │ -        │ >(S)     │
  │ S,RQ,CANCEL,¬CD                  │ -        │ 1        │
  ├──────────────────────────────────┼──────────┼──────────┤
  │ 'RESET' /* FROM DFC_RESET */     │ -        │ 1        │
  └──────────────────────────────────┴──────────┴──────────┘

  ┌──────────┬──────────────────────────────────────────────┐
  │ OUTPUT   │ FUNCTION                                     │
  │  CODE    │                                              │
  ├──────────┼──────────────────────────────────────────────┤
  │    S     │ SEND_CHECK_SENSE=X'400D';  /* CD NOT ALLOWED  */ │
  └──────────┴──────────────────────────────────────────────┘
 /* NOTE: THIS CONDITION IS DETECTED AS SEND ERROR BY
    FSM_CHAIN_SEND (PAGE 5-72) */

END FSM_EBCD_SEND;
```

```
FSM_HDX_CONT_LOSER: FSM_DEFINITION CONTEXT(SCB),
MULTIPLE_ACTION_CODES(2);
                                                                        /*
```

```
     FUNCTION:  TO ENFORCE THE HALF-DUPLEX CONTENTION SEND/RECEIVE MODE PROTOCOL FOR
                THE CONTENTION  LOSER.  SEE  "SEND/RECEIVE MODE  PROTOCOLS" ON  PAGE
                5-12 FOR PROSE DESCRIPTION.

                THE STATES ARE:

                •   CONT (CONTENTION STATE):   MEANS A CHAIN IS NOT  IN THE PROCESS
                    OF BEING SENT  OR RECEIVED.  THIS STATE HAS  THE ATTRIBUTES S,R
                    (SEND, RECEIVE), WHICH MEANS A REQUEST  MAY BE SENT OR RECEIVED
                    IN THIS STATE.

                •   SEND (SEND STATE):  MEANS A CHAIN IS CURRENTLY  IN THE PROCESS
                    OF BEING  SENT BY  THE CONTENTION  LOSER.  THIS  STATE HAS  THE
                    ATTRIBUTES S,¬R (SEND, NOT RECEIVE),  WHICH MEANS A REQUEST MAY
                    BE SENT BUT  NOT RECEIVED.  WHILE IN THIS  STATE THE CONTENTION
                    LOSER CANNOT RECEIVE REQUESTS.

                    ALL NORMAL-FLOW  RECEIVED REQUESTS ARE ENQUEUED  ON Q_TC_TO_DFC
                    BEFORE COMING TO  THE DFC.RCV PROCEDURE (PAGE  5-50).  THEY MAY
                    ONLY   BE   DEQUEUED   AND   PASSED   TO   DFC.RCV,   BY   THE
                    DEQUEUE.Q_TC_TO_DFC  PROCEDURE  (PAGE  5-40),  WHEN  THE  STATE
                    ATTRIBUTE  IS *S,R  ("DON'T  CARE"  ABOUT THE  SEND  ATTRIBUTE,
                    RECEIVE).

                •   RCV (RECEIVE STATE):  MEANS A CHAIN IS CURRENTLY IN THE PROCESS
                    OF BEING RECEIVED  FROM THE CONTENTION WINNER.   THIS STATE HAS
                    THE ATTRIBUTES ¬S,R (NOT SEND,  RECEIVE), WHICH MEANS A REQUEST
                    MAY BE RECEIVED BUT NOT SENT.

     REFERENCED BY THE FOLLOWING PROCEDURE(S):
                DEQUEUE.Q_TC_TO_DFC                 PAGE 5-40

     REFERS TO THE FOLLOWING PROCEDURE(S):
                DEQUEUE.Q_TC_TO_DFC                 PAGE 5-40
                DFC.RCV                             PAGE 5-50
```

```
                                                                        */
```

STATE ATTRIBUTES-----> / STATE NAMES----------> / INPUTS	S,R CONT 01	S,¬R SEND 02	¬S,R RCV 03
S,RQ, EC, CD /* NOTE1 */	3	3	>(S)
S,RQ, EC,¬CD	-	1	>(S)
S,RQ,¬EC	2	-	>(S)
R,-RSP,¬(080B|0813|0814|081B)	-	-	1
R,RQ, EC	-	/NOTE2	1
R,RQ,¬EC	3	/NOTE2	-
S,-RSP,¬(080B|0813|0814|081B)	2,-	-	-
'BETB' /* FROM #FSM_BSM */	-	1	1
'RESET_CONT'/* FROM DFC_RESET */	-	1	1

MULTIPLE_ACTION_CODE	DETERMINING CONDITION
1	SCB.RECOVERY_RESP=LOSER_RESPONSIBLE
2	SCB.RECOVERY_RESP=SYMMETRIC

OUTPUT CODE	FUNCTION
S	SEND_CHECK_SENSE=X'2004'; /* HDX STATE ERROR */

```
  /* NOTE1: THE CDI INDICATOR IS USED (SET) ONLY BY LU-LU SESSION TYPE 1. */
  /* NOTE2: REQUESTS ARE QUEUED ON Q_TC_TO_DFC IN THIS SITUATION. */

END FSM_HDX_CONT_LOSER;
```

```
FSM_HDX_CONT_WINNER: FSM_DEFINITION CONTEXT(SCB),
MULTIPLE_ACTION_CODES(2);
                                                                        /*
```

FUNCTION: TO ENFORCE THE HALF-DUPLEX CONTENTION SEND/RECEIVE MODE PROTOCOL FOR THE CONTENTION WINNER. SEE "SEND/RECEIVE MODE PROTOCOLS" ON PAGE 5-12 FOR PROSE DESCRIPTION.

THE STATES ARE:

- CONT (CONTENTION STATE): MEANS A CHAIN IS NOT IN THE PROCESS OF BEING SENT OR RECEIVED. WHEN IN THIS STATE, A REQUEST MAY BE SENT OR RECEIVED.
- SEND (SEND STATE): MEANS A CHAIN IS CURRENTLY IN THE PROCESS OF BEING SENT BY THE CONTENTION WINNER. REQUESTS RECEIVED FROM THE CONTENTION LOSER, WHILE IN THIS STATE, ARE CHANGED INTO AN EXR (EXCEPTION REQUEST) WITH SENSE CODE, 081B, CONTENTION ERROR. THIS CAUSES A NEGATIVE RESPONSE (WITH SENSE CODE 081B) TO BE SENT TO THE REQUEST.
- RCV (RECEIVE STATE): MEANS A CHAIN IS CURRENTLY IN THE PROCESS OF BEING RECEIVED FROM THE CONTENTION LOSER. THE CONTENTION WINNER MAY NOT SEND ANY NORMAL-FLOW REQUESTS WHILE IN THIS STATE.

REFERS TO THE FOLLOWING PROCEDURE(S):
FSM_CHAIN_RCV PAGE 5-72

```
                                                                        */
```

STATE ATTRIBUTES-----> / STATE NAMES----------> / INPUTS	S CONT 01	S SEND 02	S RCV 03
S,RQ, EC, CD /* NOTE */	3	3	>(S)
S,RQ, EC,¬CD	-	1	>(S)
S,RQ,¬EC	2	-	>(S)
R,-RSP,¬(080B|0813|0814|081B|0846)	3,-	-	-
R,RQ, EC	-	-(C)	1
R,RQ,¬EC	3	-(C)	-
'BETB' /* FROM #FSM_BSM */	-	1	1
'RESET_CONT' /* FROM DFC_RESET */	-	1	1

MULTIPLE_ACTION_CODE	DETERMINING CONDITION
1	SCB.RECOVERY_RESP=LOSER_RESPONSIBLE
2	SCB.RECOVERY_RESP=SYMMETRIC

OUTPUT CODE	FUNCTION
S	SEND_CHECK_SENSE=X'2004'; /* HDX STATE ERROR */
C	IF SDI=¬SD & FSM_CHAIN_RCV¬=PURGE THEN /*PAGE 5-72 */ CALL CHANGE_MU_TO_EXR(X'081B');/* CONTENTION ERROR */

```
/* NOTE: THE CDI BIT IS USED (SET) ONLY BY LU-LU SESSION TYPE 1. */

END FSM_HDX_CONT_WINNER;
```

```
FSM_HDX_FF: FSM_DEFINITION CONTEXT(SCB),

MULTIPLE_ACTION_CODES(3),,,;
                                                                        /*
```

FUNCTION: TO ENFORCE THE HALF-DUPLEX FLIP-FLOP PROTOCOL (WITH AND WITHOUT BRACKETS).

HDX-FF WITHOUT BRACKETS: WHEN BRACKETS ARE NOT BEING USED, THIS FSM USES ONLY STATES 4-8. THE FSM IS RESET TO SEND (4) OR RECEIVE (5) STATE. THE INPUT SIGNALS, BETB AND RESET_CONT, ARE NEVER USED; THEREFORE, STATES 1-3 ARE NEVER ENTERED. THE MOST SIGNIFICANT STATES USED ARE THE SEND (4) AND RECEIVE (5) STATES. THESE STATES CONTROL WHEN A HALF-SESSION MAY SEND AND RECEIVE NORMAL-FLOW REQUESTS. THE CHANGE DIRECTION INDICATOR (CDI) IS USED TO ALTERNATE BETWEEN THESE TWO STATES. THE 081B RECEIVE STATE (6) IS ENTERED AS A CONSEQUENCE OF RECEIVING A NEGATIVE RESPNSE WITH SENSE CODE 081B (RESOURCE UNAVAILABLE). THIS STATE ALLOWS ONLY REQUESTS TO BE RECEIVED. THE ERROR RECOVERY STATES (ERPS (7) AND ERPR (8)) ARE USED WHEN ERRORS OCCUR. ENTERING THESE STATES IS DEPENDENT ON THE TYPE OF ERROR RECOVERY (E.G., SYMMETRIC) BEING USED BY THE HALF-SESSION AND THE SENSE CODE (E.G., 0846) ON THE NEGATIVE RESPONSE. ERPR STATE ALLOWS SENDING OF LUSTAT (WITH ¬CD) AND RECEIVING OF ANY NORMAL-FLOW REQUESTS. ERPS STATE ALLOWS RECEIVING OF LUSTAT (WITH ¬CD) AND SENDING OF ANY NORMAL-FLOW REQUESTS.

HDX-FF WITH BRACKETS: WHEN BRACKETS ARE BEING USED ALL 8 STATES OF THIS FSM ARE USED. THERE IS A TIGHT COUPLING BETWEEN THE STATES OF THIS FSM AND THE STATES OF THE BRACKET FSM (FSM_BSM_BIDDER (PAGE 5-68) OR FSM_BSM_FSP (PAGE 5-70)). THE CONTENTION STATES (1-3) ARE USED WHEN THE BRACKET FSM IS IN BETWEEN-BRACKETS (BETB) STATE. THE OTHER STATES (4-8) ARE USED WHEN THE BRACKET FSM IS IN IN-BRACKET (INB) STATE. THE INPUT SIGNALS--BETB, INB_SEND, AND INB_RCV--FROM THE BRACKET FSM COORDINATE THESE STATE COUPLINGS. THE CONTENTION BETWEEN CHAIN STATE (1) IS ENTERED WHEN BETWEEN CHAINS AND BETWEEN BRACKETS. THIS IS ALSO THE RESET STATE. THE CONTENTION IN-CHAIN STATES (2-3) ARE USED FOR REMEMBERING WHEN IN THE MIDDLE OF SENDING OR RECEIVING A CHAIN. THE DESCRIPTION OF STATES 4-8 IS THE SAME AS FOR HDX-FF WITHOUT BRACKETS.

NOTE: RECEIVED AND SENT RESPONSES COME TO THIS FSM FROM FSM_CONTROL_HDX_RSP_RCV (PAGE 5-75 OR 5-76 OR 5-77) AND FSM_CONTROL_HDX_RSP_SEND (PAGE 5-78 OR 5-79 OR 5-80). THESE FSMS SERIALIZE VARIOUS RACE CONDITIONS THAT CAN OCCUR AND SHIELD FSM_HDX_FF FROM THE COMPLEXITIES THAT THEY CREATE. THE BASIC IDEA IS TO PRESENT ALL RESPONSES TO FSM_HDX_FF AT THE END OF A PERIOD OF ACTIVITY SO THAT FSM_HDX_FF WILL NOT HAVE TO CONTAIN NUMEROUS PENDING STATES OR COMPLICATED CHECKING OF PENDING STATES IN OTHER FSMS.

REFERENCED BY THE FOLLOWING PROCEDURE(S):

DEQUEUE.Q_TC_TO_DFC	PAGE 5-40
FSM_CONTROL_HDX_RSP_RCV_ERP_DL	PAGE 5-76
FSM_CONTROL_HDX_RSP_RCV_ERP_IM	PAGE 5-77
FSM_CONTROL_HDX_RSP_SEND_ERP_DL	PAGE 5-79
FSM_CONTROL_HDX_RSP_SEND_ERP_IM	PAGE 5-80

REFERS TO THE FOLLOWING PROCEDURE(S):

FSM_BSM_BIDDER	PAGE 5-68
FSM_BSM_FSP	PAGE 5-70
FSM_CHAIN_RCV	PAGE 5-72
FSM_CONTROL_HDX_RSP_RCV	PAGE 5-75
FSM_CONTROL_HDX_RSP_SEND	PAGE 5-78


```
                                                                        */
```

STATE ATTRIBUTES-------> / STATE NAMES----------> / INPUTS	S,R CONT 1	S,¬R CONT SEND 2	¬S,R CONT RCV 3	S,¬R SEND 4	¬S,R RCV 5	¬S,R RCV 081B 6	S,R ERPS 7	S,R ERPR 8
S,RQ,FMD|CANCEL , CD	-	1	>(S)	5	>(S)	>(S)	5	>(S)
S,RQ,CHASE|LUSTAT|QC, CD	>(S)	>(S)	>(S)	5	>(S)	>(S)	5	>(S)
S,RQ,FMD|CANCEL ,¬CD, EC	-	1	>(S)	-	>(S)	>(S)	-	>(S)
S,RQ,LUSTAT, EB ,¬CD	-	>(S)	>(S)	-	>(S)	>(S)	-	>(S)
S,RQ,LUSTAT,¬EB ,¬CD	-	>(S)	>(S)	-	>(S)	>(S)	-	-
S,RQ,REST_OF_DFC ,¬CD	-	>(S)	>(S)	-	>(S)	>(S)	-	>(S)
S,RQ, ,¬EC	2	-	>(S)	-	>(S)	>(S)	-	>(S)
R,-RSP, 081B	-	-	-	6	6	-	-	-
R,-RSP, 0846, SYMMETRIC	-	-	-	8	8	-	-	-
R,-RSP, 0846,¬SYMMETRIC,RECOVERER	-	-	-	7	7	-	-	-
R,-RSP, 0846,¬SYMMETRIC,NOT_RECOVERER	-	-	-	5	-	-	-	-
R,-RSP,¬RACE, SYMMETRIC	-	-	-	7	7	-	-	-
R,-RSP,¬RACE,¬SYMMETRIC,RECOVERER	-	-	-	7	7	-	-	-
R,-RSP,¬RACE,¬SYMMETRIC,NOT_RECOVERER	-	-	-	5	-	-	-	-
R,RQ,FMD|LUSTAT|CANCEL,CT(BB), CD	-	>(R),>(R),-(C)	1	>(R),>(R),-	>(R),4,-	-	-	-
R,RQ,FMD|LUSTAT|CANCEL,CT(BB),¬CD, EC	-	>(R),>(R),-(C)	1	>(R),>(R),-	>(R),-,-	-	-	-
R,RQ,FMD ,CT(BB), ¬EC	3	>(R),>(R),-(C)	-	>(R),>(R),-	>(R),-,-	-	-	-
R,RQ,FMD|CANCEL ,CT(¬BB), CD	-	>(R),>(R),-(C)	1	>(R),>(R),>(R)	4	4	>(R)	4
R,RQ,CHASE|LUSTAT|QC ,CT(¬BB), CD	>(R)	>(R),>(R),>(R)	>(R)	>(R),>(R),>(R)	4	4	>(R)	4
R,RQ,FMD|CANCEL ,CT(¬BB),¬CD, EC	-	>(R),>(R),-(C)	1	>(R),>(R),>(R)	-	4	>(R)	-
R,RQ,LUSTAT ,CT(¬BB),¬CD	-	>(R),>(R),-(C)	>(R)	>(R),>(R),-	-	4	-	-
R,RQ,REST_OF_DFC ,CT(¬BB),¬CD	-	>(R),>(R),-(C)	>(R)	>(R),>(R),-	-	4	>(R)	-
R,RQ,FMD ,CT(¬BB), ¬EC	3	>(R),>(R),-(C)	-	>(R),>(R),>(R)	-	-	>(R)	-
S,-RSP, 0846, SYMMETRIC	-	-	-	7	7	-	-	-
S,-RSP, 0846,¬SYMMETRIC,RECOVERER	-	-	-	-	4	-	-	-
S,-RSP, 0846,¬SYMMETRIC,NOT_RECOVERER	-	-	-	8	8	-	-	-
S,-RSP,¬RACE, SYMMETRIC	-	-	-	8	8	-	-	-
S,-RSP,¬RACE,¬SYMMETRIC,RECOVERER	-	-	-	-	4	-	-	-
S,-RSP,¬RACE,¬SYMMETRIC,NOT_RECOVERER	-	-	-	8	8	-	-	-
'BETB' /*FROM #FSM_BSM */	-	-	-	1	1	1	1	1
'INB_SEND' /*FROM #FSM_BSM */	4	4	4	-	-	-	-	-
'INB_RCV' /*FROM #FSM_BSM */	5	5	5	-	-	-	-	-
'RESET_CONT' /*FROM DFC_RESET*/	-	1	1	1	1	1	1	1
'RESET_SEND' /*FROM DFC_RESET*/	4	4	4	-	4	4	4	4
'RESET_RCV' /*FROM DFC_RESET*/	5	5	5	5	-	5	5	5

MULTIPLE_ACTION_CODE	DEFINING CONDITION
1	SCB.USING_BRACKETS=NO /* NO BRACKETS */
2	SCB.USING_BRACKETS=YES & SCB.FIRST_SPEAKER=NO /* BIDDER */
3	SCB.USING_BRACKETS=YES & SCB.FIRST_SPEAKER=YES /* FIRST SPEAKER */

OUTPUT CODE	FUNCTION
C	IF SDI=¬SD & FSM_CHAIN_RCV¬=PURGE THEN /* PAGE 5-72 */ CALL CHANGE_MU_TO_EXR(X'081B'); /* CONTENTION ERROR */
S	SEND_CHECK_SENSE=X'2004'; /* HDX STATE ERROR */
R	RECEIVE_CHECK_SENSE=X'2004'; /* HDX STATE ERROR */

END FSM_HDX_FF;

```
FSM_IMM_RQ_MODE_RCV: FSM_DEFINITION CONTEXT(SCB);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO ENFORCE THE IMMEDIATE REQUEST MODE  PROTOCOL FOR NORMAL FLOW (SEE
   │                 CHAPTER 4 FOR PROSE DESCRIPTION).                        │
   │                                                                          │
   │      NOTE:      THE IMPLEMENTATION OF  THIS FSM IS OPTIONAL BECAUSE IT  IS USED ONLY
   │                 TO CHECK FOR RECEIVE ERROR CONDITIONS.                   │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */

   ┌──────────────────────────┬──────┬──────────┬──────┬──────────┬──────────┐
   │          STATE NAMES---->│RESET │RCVD RQD  │ INC  │INC RSP   │RCVD RQD  │
   │                          │      │¬CANCEL   │      │SENT      │CANCEL    │
   │ INPUTS                   │ 1    │ 2        │ 3    │ 4        │ 5        │
   ├──────────────────────────┼──────┼──────────┼──────┼──────────┼──────────┤
   │ R,RQ,¬CANCEL, EC, RQD    │ 2    │ >(R)     │ 2    │ 1        │ >(R)     │
   │ R,RQ,¬CANCEL, EC,¬RQD    │ -    │ >(R)     │ 1    │ 1        │ >(R)     │
   │ R,RQ,¬CANCEL,¬EC         │ 3    │ >(R)     │ -    │ -        │ >(R)     │
   │ R,RQ, CANCEL             │ 5    │ >(R)     │ 5    │ 5        │ >(R)     │
   ├──────────────────────────┼──────┼──────────┼──────┼──────────┼──────────┤
   │ S, RSP,¬CANCEL,TO_CURRENT_CHAIN │ - │ 1     │ 4    │ -        │ -        │
   │ S, RSP, CANCEL,TO_CURRENT_CHAIN │ - │ -     │ -    │ -        │ 1        │
   ├──────────────────────────┼──────┼──────────┼──────┼──────────┼──────────┤
   │ 'RESET' /* FROM DFC_RESET */ │ - │ 1        │ 1    │ 1        │ 1        │
   ├──────────────────────────┴──────┴──────────┴──────┴──────────┴──────────┤
   │                                                                         │
   ├──────────┬──────────────────────────────────────────────────────────────┤
   │ OUTPUT   │ FUNCTION                                                     │
   │  CODE    │                                                              │
   ├──────────┼──────────────────────────────────────────────────────────────┤
   │    R     │ RECEIVE_CHECK_SENSE=X'200A'; /* IMMEDIATE RQ MODE ERROR   */ │
   └──────────┴──────────────────────────────────────────────────────────────┘
END FSM_IMM_RQ_MODE_RCV;


FSM_IMM_RQ_MODE_SEND: FSM_DEFINITION CONTEXT(SCB);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO ENFORCE THE IMMEDIATE REQUEST  MODE PROTOCOL FOR NORMAL-FLOW (SEE
   │                 CHAPTER 4 FOR PROSE DESCRIPTION).                        │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */

   ┌──────────────────────────┬──────┬──────────┬──────┬──────────┬──────────┐
   │          STATE NAMES---->│RESET │SENT RQD  │ INC  │INC RSP   │SENT RQD  │
   │                          │      │¬CANCEL   │      │RCVD      │CANCEL    │
   │ INPUTS                   │ 1    │ 2        │ 3    │ 4        │ 5        │
   ├──────────────────────────┼──────┼──────────┼──────┼──────────┼──────────┤
   │ S,RQ,¬CANCEL, EC, RQD    │ 2    │ >(S)     │ 2    │ 1        │ >(S)     │
   │ S,RQ,¬CANCEL, EC,¬RQD    │ -    │ >(S)     │ 1    │ 1        │ >(S)     │
   │ S,RQ,¬CANCEL,¬EC         │ 3    │ >(S)     │ -    │ -        │ >(S)     │
   │ S,RQ, CANCEL             │ 5    │ >(S)     │ 5    │ 5        │ >(S)     │
   ├──────────────────────────┼──────┼──────────┼──────┼──────────┼──────────┤
   │ R, RSP,¬CANCEL,TO_CURRENT_CHAIN │ - │ 1     │ 4    │ -        │ -        │
   │ R, RSP, CANCEL,TO_CURRENT_CHAIN │ - │ 1     │ -    │ -        │ 1        │
   ├──────────────────────────┼──────┼──────────┼──────┼──────────┼──────────┤
   │ 'RESET' /* FROM DFC_RESET */ │ - │ 1        │ 1    │ 1        │ 1        │
   ├──────────────────────────┴──────┴──────────┴──────┴──────────┴──────────┤
   │                                                                         │
   ├──────────┬──────────────────────────────────────────────────────────────┤
   │ OUTPUT   │ FUNCTION                                                     │
   │  CODE    │                                                              │
   ├──────────┼──────────────────────────────────────────────────────────────┤
   │    S     │ SEND_CHECK_SENSE=X'200A'; /* IMM RQ MODE STATE ERROR    */ │
   └──────────┴──────────────────────────────────────────────────────────────┘
END FSM_IMM_RQ_MODE_SEND;
```

```
FSM_QEC_RCV: FSM_DEFINITION CONTEXT(SCB);
                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO ENFORCE THE  QUIESCE PROTOCOL FOR THE HALF-SESSION  THAT IS BEING │
  │                 QUIESCED (RECEIVED  QEC).  SEE "QUIESCE  PROTOCOL" ON PAGE  5-20 FOR │
  │                 PROSE DESCRIPTION.                                   │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES-----> / INPUTS	RESET 01	PEND QC 02	QUIESCED 03
R,RQ,EXP,QEC	-	>(R)	>(R)
S,+RSP,QEC	2	-	-
R,RQ,EXP,RELQ	-	-	-
S,+RSP,RELQ	-	1	1
S,RQ,NORM,QC	>(S1)	3	>(S1)
R,+RSP,QC	-	-	-
S,RQ,NORM, CANCEL	-	-	>(S2)
S,RQ,NORM,¬CANCEL, BC	-	>(S2)	>(S2)
S,RQ,NORM,¬CANCEL,¬BC	-	-	>(S2)
'RESET' /* FROM DFC_RESET */	-	1	1

OUTPUT CODE	FUNCTION
S1	SEND_CHECK_SENSE=X'0809'; /* MODE INCONSISTENCY */
S2	SEND_CHECK_SENSE=X'2006'; /* DATA TRAFFIC QUIESCED */
R	RECEIVE_CHECK_SENSE=X'0809'; /* MODE INCONSISTENCY */

```
END FSM_QEC_RCV;
```

```
FSM_QEC_SEND: FSM_DEFINITION CONTEXT(SCB);
                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO ENFORCE THE QUIESCE PROTOCOL FOR  THE HALF-SESSION THAT SENT QEC. │
  │                 SEE "QUIESCE PROTOCOL" ON PAGE 5-20 FOR PROSE DESCRIPTION. │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES-----> / INPUTS	RESET 01	PEND QC 02	QUIESCED 03
S,RQ,EXP,QEC	-	>(S)	>(S)
R,+RSP,QEC	2	-	-
S,RQ,EXP,RELQ	-	-	-
R,+RSP,RELQ	-	1	1
R,RQ,NORM,QC	-	3	>(R1)
S,+RSP,QC	-	-	-
R,RQ,NORM, CANCEL	-	-	>(R2)
R,RQ,NORM,¬CANCEL, BC	-	-	>(R2)
R,RQ,NORM,¬CANCEL,¬BC	-	-	>(R2)
'RESET' /*FROM DFC_RESET*/	-	1	1

OUTPUT CODE	FUNCTION
S	SEND_CHECK_SENSE=X'0809'; /* MODE INCONSISTENCY */
R1	RECEIVE_CHECK_SENSE=X'0809'; /* MODE INCONSISTENCY */
R2	RECEIVE_CHECK_SENSE=X'2006'; /* DATA TRAFFIC QUIESCED */

```
END FSM_QEC_SEND;
```

```
FSM_QRI_CHECK_SEND: FSM_DEFINITION CONTEXT(SCB),

MULTIPLE_ACTION_CODES(2);
                                                                              /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO  ENFORCE  THE  PROTOCOL  FOR  SENDING  REQUESTS  USING  THE  QRI      │
   │                 INDICATOR.  WHEN  RUNNING DELAYED RESPONSE MODE  AND THIS FSM  IS IN     │
   │                 QR_SENT STATE, REQUESTS OTHER THAN CHASE MAY BE SENT WITH ¬QR.  THIS      │
   │                 IS BECAUSE RESPONSES  MAY COME BACK IN ANY ORDER  WHEN USING DELAYED      │
   │                 RESPONSE MODE.                                                           │
   │                                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                           │
   │                 DEQUEUE.Q_TC_TO_DFC                PAGE 5-40                             │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                              */

   ┌──────────────────────────────────┬──────┬──────┐
   │                    STATE NAMES-----> │ RESET │ QR   │
   │                                  │      │ SENT │
   │ INPUTS                           │ 1    │ 2    │
   ├──────────────────────────────────┼──────┼──────┤
   │ S,RQ,¬RQN, QR                    │ 2    │ -    │
   │ S,RQ,¬RQN,¬QR, CHASE             │ -    │ >(S) │
   │ S,RQ,¬RQN,¬QR,¬CHASE             │ -    │ >(S),- │
   ├──────────────────────────────────┼──────┼──────┤
   │ 'NO_OUTSTANDING_RQS'             │ -    │ 1    │
   │ 'RESET'   /* FROM DFC_RESET */   │ -    │ 1    │
   └──────────────────────────────────┴──────┴──────┘

   ┌──────────────────┬──────────────────────────────────────────────┐
   │ MULTIPLE_ACTION_CODE │ DEFINING CONDITION                       │
   ├──────────────────┼──────────────────────────────────────────────┤
   │        1         │ SCB.PARTNER_HALF_SESSION_RSP_MODE=IMMEDIATE  │
   ├──────────────────┼──────────────────────────────────────────────┤
   │        2         │ SCB.PARTNER_HALF_SESSION_RSP_MODE=DELAYED    │
   └──────────────────┴──────────────────────────────────────────────┘

   ┌──────────┬──────────────────────────────────────────────────────┐
   │ OUTPUT   │ FUNCTION                                             │
   │  CODE    │                                                      │
   ├──────────┼──────────────────────────────────────────────────────┤
   │    S     │ SEND_CHECK_SENSE=X'200B'; /* QRI STATE ERROR    */  │
   └──────────┴──────────────────────────────────────────────────────┘
END FSM_QRI_CHECK_SEND;


FSM_QRI_CHAIN_RCV: FSM_DEFINITION CONTEXT(SCB);
                                                                              /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS FSM ENFORCES THE SETTING OF THE  QRI INDICATOR IN THE RH.  THIS      │
   │                 INDICATOR IS SET THE SAME FOR ALL RU'S IN A CHAIN, I.E., ALL RU'S IN     │
   │                 A CHAIN HAVE QRI=QR  OR ALL RU'S IN A CHAIN HAVE QRI=¬QR.                 │
   │                                                                                          │
   │      NOTE:      THE IMPLEMENTATION OF  THIS FSM IS OPTIONAL BECAUSE IT  IS USED ONLY      │
   │                 TO DETECT RECEIVE ERROR CONDITIONS.                                      │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                              */

   ┌──────────────────────────────────┬──────┬──────┬──────┐
   │                    STATE NAMES-----> │ RESET │ INC  │ INC  │
   │                                  │      │ QR   │ ¬QRI │
   │                                  │ 1    │ 2    │ 3    │
   ├──────────────────────────────────┼──────┼──────┼──────┤
   │ R,RQ, QR, EC                     │ -    │ 1    │ >(R) │
   │ R,RQ, QR,¬EC                     │ 2    │ -    │ >(R) │
   ├──────────────────────────────────┼──────┼──────┼──────┤
   │ R,RQ,¬QR, EC                     │ -    │ >(R) │ 1    │
   │ R,RQ,¬QR,¬EC                     │ 3    │ >(R) │ -    │
   ├──────────────────────────────────┼──────┼──────┼──────┤
   │ 'RESET' /* FROM DFC_RESET */     │ -    │ 1    │ 1    │
   └──────────────────────────────────┴──────┴──────┴──────┘

   ┌──────────┬──────────────────────────────────────────────────────┐
   │ OUTPUT   │ FUNCTION                                             │
   │  CODE    │                                                      │
   ├──────────┼──────────────────────────────────────────────────────┤
   │    R     │ RECEIVE_CHECK_SENSE=X'200B';/*QRI STATE ERROR*/      │
   └──────────┴──────────────────────────────────────────────────────┘
END FSM_QRI_CHAIN_RCV;
```

```
FSM_QRI_CHAIN_SEND: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS FSM ENFORCES THE SETTING OF THE  QRI INDICATOR IN THE RH.  THIS  │
   │                 INDICATOR MUST BE  SET THE SAME FOR  ALL RU'S IN A  CHAIN, I.E., ALL  │
   │                 RU'S IN A CHAIN HAVE QRI=QR OR ALL RU'S IN A CHAIN HAVE QRI=¬QR.      │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES-----> / INPUTS	RESET 1	INC QR 2	INC ¬QRI 3
S,RQ, QR, EC	-	1	>(S)
S,RQ, QR,¬EC	2	-	>(S)
S,RQ,¬QR, EC	-	>(S)	1
S,RQ,¬QR,¬EC	3	>(S)	-
'RESET' /* FROM DFC_RESET */	-	1	1

OUTPUT CODE	FUNCTION
S	SEND_CHECK_SENSE=X'200B';/* QRI STATE ERROR*/

```
END FSM_QRI_CHAIN_SEND;


FSM_RES: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  FSM  ENFORCES  THAT  NORMAL-FLOW REQUESTS  NOT  BE  SENT  WHEN  │
   │                 RESOURCES ARE UNAVAILABLE.                           │
   │                                                                      │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
   │                 FSM_CHAIN_RCV                    PAGE 5-72           │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES-----> / INPUTS	AVL 01	UNAVL 02
'UNAVL' /* NOTE */	2	-
'AVL' /* NOTE */	-	1
P,RQ,NORM	-	-(C)
'RESET' /* FROM DFC_RESET */	-	1

OUTPUT CODE	FUNCTION
C	IF SDI=¬SD & FSM_CHAIN_RCV¬=PURGE THEN /*P. 5-72 */ CALL CHANGE_MU_TO_EXR(X'081B'); /* NO RESOURCE */

```
 /* NOTE: THE INPUTS TO THIS FSM ARE SIGNALS FROM UPM_RES(PAGE 5-67).
   UNAVL INDICATES THAT RESOURCES NECESSARY FOR HANDLING
       NORMAL-FLOW DATA ON THIS HALF-SESSION ARE UNAVAIABLE.
   AVL INDICATES THAT RESOURCES ARE AVAILABLE. */

END FSM_RES;
```

```
FSM_RTR_BIDDER: FSM_DEFINITION CONTEXT(SCB);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO ENFORCE RTR PORTION OF THE  BRACKET PROTOCOL FOR THE BIDDER.  SEE │
   │                 "BRACKETS PROTOCOL" ON PAGE 5-14 FOR PROSE DESCRIPTION.  │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */

   ┌──────────────────────────────────┬──────────┬──────────┐
   │                  STATE NAMES----->│ RESET    │ PEND     │
   │ INPUTS                           │ 01       │ 02       │
   ├──────────────────────────────────┼──────────┼──────────┤
   │ S,RQ,BB                          │ -        │ >(S)     │
   ├──────────────────────────────────┼──────────┼──────────┤
   │ R,+RSP,BID                       │ -        │ 1        │
   │ R,-RSP,BID,0814                  │ 2        │ -        │
   │ R,-RSP,BID,¬0814                 │ -        │ 1        │
   │ R,-RSP,CT(BB),0814               │ 2        │ -  . . . │
   ├──────────────────────────────────┼──────────┼──────────┤
   │ R,RQ,RTR                         │ -        │ 1        │
   ├──────────────────────────────────┼──────────┼──────────┤
   │ 'RESET' /* FROM DFC_RESET */     │ -        │ 1        │
   ├──────────────────────────────────┴──────────┴──────────┤
   │                                                       │
   ├──────────┬────────────────────────────────────────────┤
   │ OUTPUT   │ FUNCTION                                   │
   │  CODE    │                                            │
   ├──────────┼────────────────────────────────────────────┤
   │    S     │ SEND_CHECK_SENSE=X'2003'; /* BRACKET ERROR */ │
   └──────────┴────────────────────────────────────────────┘
END FSM_RTR_BIDDER;


FSM_RTR_FSP: FSM_DEFINITION CONTEXT(SCB);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO ENFORCE RTR PORTION OF THE BRACKET PROTOCOL FOR THE FIRST │
   │                 SPEAKER.  SEE "BRACKETS PROTOCOL" ON PAGE 5-14 FOR PROSE │
   │                 DESCRIPTION.                                             │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */

   ┌──────────────────────────────────┬──────────┬──────────┐
   │                  STATE NAMES----->│ RESET    │ PEND     │
   │ INPUTS                           │ 01       │ 02       │
   ├──────────────────────────────────┼──────────┼──────────┤
   │ S,+RSP,BID                       │ -        │ 1        │
   │ S,-RSP,BID,0814                  │ 2        │ -        │
   │ S,-RSP,BID,¬0814                 │ -        │ 1        │
   │ S,-RSP,CT(BB),0814               │ 2        │ -        │
   ├──────────────────────────────────┼──────────┼──────────┤
   │ S,RQ,RTR                         │ -        │ 1        │
   ├──────────────────────────────────┼──────────┼──────────┤
   │ 'RESET' /* FROM DFC_RESET */     │ -        │ 1        │
   └──────────────────────────────────┴──────────┴──────────┘
END FSM_RTR_FSP;
```

```
FSM_SBI_RCV: FSM_DEFINITION CONTEXT(SCB);
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO ENFORCE THE STOP-BRACKET-INITIATION PROTOCOL FOR THE SBI  │
   │                 RECEIVER.  SEE "STOP-BRACKET-INITIATION PROTOCOL" ON PAGE 5-19 FOR │
   │                 PROSE DESCRIPTION.                                   │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

   ┌──────────────────────────────────┬──────┬──────┬──────┐
   │                  STATE NAMES---->│ RESET│ PEND │ NOBB │
   │                                  │      │ NOBB │      │
   │ INPUTS                           │ 01   │ 02   │ 03   │
   ├──────────────────────────────────┼──────┼──────┼──────┤
   │ R,RQ,EXP,SBI                     │ -    │ >(R) │ >(R) │
   │ S,+RSP,SBI                       │ 2    │ -    │ -    │
   ├──────────────────────────────────┼──────┼──────┼──────┤
   │ S,RQ,NORM,BIS                    │ -    │ -    │ >(S1)│
   │ R,+RSP,BIS                       │ 3    │ 3    │ -    │
   ├──────────────────────────────────┼──────┼──────┼──────┤
   │ S,RQ,NORM,BID                    │ -    │ -    │ >(S2)│
   │ S,RQ,NORM,BB                     │ -    │ -    │ >(S2)│
   ├──────────────────────────────────┼──────┼──────┼──────┤
   │ 'RESET' /* FROM DFC_RESET */     │ -    │ 1    │ 1    │
   └──────────────────────────────────┴──────┴──────┴──────┘

   ┌──────────┬──────────────────────────────────────────────────────┐
   │ OUTPUT   │ FUNCTION                                             │
   │  CODE    │                                                      │
   ├──────────┼──────────────────────────────────────────────────────┤
   │   S1     │ SEND_CHECK_SENSE=X'0809'; /* MODE INCONSISTENCY  */ │
   ├──────────┼──────────────────────────────────────────────────────┤
   │   S2     │ SEND_CHECK_SENSE=X'2008'; /* NO BEGIN BRACKET    */ │
   ├──────────┼──────────────────────────────────────────────────────┤
   │   R      │ RECEIVE_CHECK_SENSE=X'0809'; /* MODE INCONSISTENCY */ │
   └──────────┴──────────────────────────────────────────────────────┘
END FSM_SBI_RCV;


FSM_SBI_SEND: FSM_DEFINITION CONTEXT(SCB);
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO ENFORCE THE STOP-BRACKET-INITIATION PROTOCOL FOR THE SBI SENDER. │
   │                 SEE "STOP-BRACKET-INITIATION PROTOCOL" ON PAGE 5-19 FOR PROSE │
   │                 DESCRIPTION.                                         │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

   ┌──────────────────────────────────┬──────┬──────┬──────┐
   │                  STATE NAMES---->│ RESET│ PEND │ NOBB │
   │                                  │      │ NOBB │      │
   │ INPUTS                           │ 01   │ 02   │ 03   │
   ├──────────────────────────────────┼──────┼──────┼──────┤
   │ S,RQ,EXP,SBI                     │ -    │ >(S) │ >(S) │
   │ R,+RSP,SBI                       │ 2    │ -    │ -    │
   ├──────────────────────────────────┼──────┼──────┼──────┤
   │ R,RQ,NORM,BIS                    │ -    │ -    │ >(R1)│
   │ S,+RSP,BIS                       │ 3    │ 3    │ -    │
   ├──────────────────────────────────┼──────┼──────┼──────┤
   │ R,RQ,NORM,BID                    │ -    │ -    │ >(R2)│
   │ R,RQ,NORM,BB                     │ -    │ -    │ >(R2)│
   ├──────────────────────────────────┼──────┼──────┼──────┤
   │ 'RESET' /* FROM DFC_RESET */     │ -    │ 1    │ 1    │
   └──────────────────────────────────┴──────┴──────┴──────┘

   ┌──────────┬──────────────────────────────────────────────────────┐
   │ OUTPUT   │ FUNCTION                                             │
   │  CODE    │                                                      │
   ├──────────┼──────────────────────────────────────────────────────┤
   │   S      │ SEND_CHECK_SENSE=X'0809'; /* MODE INCONSISTENCY  */ │
   ├──────────┼──────────────────────────────────────────────────────┤
   │   R1     │ RECEIVE_CHECK_SENSE=X'0809'; /* MODE INCONSISTENCY */ │
   ├──────────┼──────────────────────────────────────────────────────┤
   │   R2     │ RECEIVE_CHECK_SENSE=X'2008'; /* NO BEGIN BRACKET  */ │
   └──────────┴──────────────────────────────────────────────────────┘
END FSM_SBI_SEND;
```

```
FSM_SHUTD_RCV: FSM_DEFINITION CONTEXT(SCB);
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO ENFORCE THE SHUTDOWN PROTOCOL FOR  THE HALF-SESSION THAT IS BEING  │
   │                 SHUT DOWN  (RECEIVES SHUTD).  SEE  "SHUTDOWN PROTOCOL" ON  PAGE 5-21  │
   │                 FOR PROSE DESCRIPTION.                               │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES-----> / INPUTS	RESET 01	PEND SHUTC 02	QUIESCED 03
R,RQ,EXP,SHUTD S,+RSP,SHUTD	- 2	>(R) -	>(R) -
R,RQ,EXP,RELQ S,+RSP,RELQ	- -	- 1	- 1
S,RQ,EXP,SHUTC R,+RSP,SHUTC	>(S1) -	- 3	>(S1) -
S,RQ,NORM	-	-	>(S2)
'RESET' /* FROM DFC_RESET */	-	1	1

OUTPUT CODE	FUNCTION
S1	SEND_CHECK_SENSE=X'0809'; /* MODE INCONSISTENCY */
S2	SEND_CHECK_SENSE=X'2006'; /* DATA TRAFFIC QUIESCED */
R	RECEIVE_CHECK_SENSE=X'0809'; /* MODE INCONSISTENCY */

```
END FSM_SHUTD_RCV;
```

```
FSM_SHUTD_SEND: FSM_DEFINITION CONTEXT(SCB);
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO ENFORCE  THE SHUTDOWN  PROTOCOL FOR  THE HALF-SESSION  THAT SENDS  │
   │                 SHUTD.  SEE "SHUTDOWN PROTOCOL" ON PAGE 5-21 FOR PROSE DESCRIPTION.  │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES-----> / INPUTS	RESET 01	PEND SHUTC 02	QUIESCED 03
S,RQ,EXP,SHUTD R,+RSP,SHUTD	- 2	>(S) -	>(S) -
S,RQ,EXP,RELQ R,+RSP,RELQ	- -	- 1	- 1
R,RQ,EXP,SHUTC S,+RSP,SHUTC	>(R) -	- 3	>(R) -
R,RQ,NORM	-	-	-
'RESET' /* FROM DFC_RESET */	-	1	1

OUTPUT CODE	FUNCTION
S	SEND_CHECK_SENSE=X'0809'; /* MODE INCONSISTENCY */
R	RECEIVE_CHECK_SENSE=X'0809'; /* MODE INCONSISTENCY */

```
END FSM_SHUTD_SEND;
```

```
|     THE SYMBOLS  USED IN THE "INPUTS"  COLUMN OF THE STATE-TRANSITION  MATRICES ARE
|     DEFINED BELOW.
*/

'AVL'                       FSMINPUT='AVL';
 BB                         BBI=BB;
 BC                         BCI=BC;
'BETB'                      FSMINPUT='BETB';
 BID                        RU_CTGY=DFC & RQ_CODE=BID;
 BIS                        RU_CTGY=DFC & RQ_CODE=BIS;
 CANCEL                     RU_CTGY=DFC & RQ_CODE=CANCEL;
 CD                         CDI=CD;
 CHASE                      RU_CTGY=DFC & RQ_CODE=CHASE;
 CHASE|LUSTAT|QC            RU_CTGY=DFC & RQ_CODE=(CHASE|LUSTAT|QC);
 CT(BB)                     CT_BBI=BB;
(CT(BB)&CT(EB))             CT_BBI=BB & CT_EBI=EB;
 CT(CD)                     CT_CDI=CD;
 CT(EB)                     CT_EBI=EB;
 EB                         EBI=EB;
 EC                         ECI=EC;
 EXP                        EFI=EXP;
 FMD                        RU_CTGY=FMD;
 FMD|CANCEL                 RU_CTGY=FMD | (RU_CTGY=DFC & RQ_CODE=CANCEL);
 FMD|EBDFC                  RU_CTGY=FMD | (RU_CTGY=DFC & RQ_CODE=(CHASE|LUSTAT|QC));
 FMD|LUSTAT                 RU_CTGY=FMD | (RU_CTGY=DFC & RQ_CODE=LUSTAT);
 FMD|LUSTAT|CANCEL          RU_CTGY=FMD | (RU_CTGY=DFC & RQ_CODE=(LUSTAT|CANCEL));
'INB_RCV'                   FSMINPUT='INB_RCV';
'INB_SEND'                  FSMINPUT='INB_SEND';
 LUSTAT                     RU_CTGY=DFC & RQ_CODE=LUSTAT;
'NO_OUTSTANDING_RQS'        FSMINPUT='NO_OUTSTANDING_RQS';
 NORM                       EFI=NORM;
 NOT_RECOVERER              (SCB.HALF_SESSION=PRI & SCB.CONT_WIN=PRI) |
                            (SCB.HALF_SESSION=SEC & SCB.CONT_WIN=SEC);
 OTHERDFC                   RU_CTGY=DFC & RQ_CODE¬=(BID|CANCEL|RTR);
 QC                         RU_CTGY=DFC & RQ_CODE=QC;
 QEC                        RU_CTGY=DFC & RQ_CODE=QEC;
 QR                         QRI=QR;
 R                          MUCB.DIRECTION=RECEIVE;
 RACE                       SNC(0:15)=(X'080B'|X'0813'|X'0814'|X'181B'|X'0846');
 RECOVERER                  (SCB.HALF_SESSION=PRI & SCB.CONT_WIN=SEC) |
                            (SCB.HALF_SESSION=SEC & SCB.CONT_WIN=PRI);
 RELQ                       RU_CTGY=DFC & RQ_CODE=RELQ;
'RESET'                     FSMINPUT='RESET';
'RESET_BETB'                FSMINPUT='RESET_BETB';
'RESET_CONT'                FSMINPUT='RESET_CONT';
'RESET_INB'                 FSMINPUT='RESET_INB';
'RESET_RCV'                 FSMINPUT='RESET_RCV';
'RESET_SEND'                FSMINPUT='RESET_SEND';
 REST_OF_DFC                RU_CTGY=DFC & RQ_CODE=(BID|BIS|CHASE|QC|RTR);
 RQ                         RRI=RQ;
 RQD                        RQD;
(RQD|(RQE&CD))              RQD | (RQE & CDI=CD);
 RQE                        RQE;
 RQN                        RQN;
 RSP                        RRI=RSP;
+RSP                         RRI=RSP & RTI=POS;
-RSP                         RRI=RSP & RTI=NEG;
 RTR                        RU_CTGY=DFC & RQ_CODE=RTR;
 S                          MUCB.DIRECTION=SEND;
 SBI                        RU_CTGY=DFC & RQ_CODE=SBI;
 SHUTC                      RU_CTGY=DFC & RQ_CODE=SHUTC;
 SHUTD                      RU_CTGY=DFC & RQ_CODE=SHUTD;
 SYMMETRIC                  SCB.RECOVERY_RESP=SYMMETRIC;
 SYNC_EVENT                 RQD | (RQE & CDI=CD);
 TO_CURRENT_CHAIN           ((LAST_ENTRY(CT_PTR)->CT_END_SNF -
                                 LAST_ENTRY(CT_PTR)->CT_BEG_SNF>=0) &
                            (SNF>=LAST_ENTRY(CT_PTR)->CT_BEG_SNF &
                                 SNF<=LAST_ENTRY(CT_PTR)->CT_END_SNF)) |
                           ((LAST_ENTRY(CT_PTR)->CT_END_SNF -
                                 LAST_ENTRY(CT_PTR)->CT_BEG_SNF<0) &
                            (SNF>=LAST_ENTRY(CT_PTR)->CT_BEG_SNF |
                                 SNF<=LAST_ENTRY(CT_PTR)->CT_END_SNF));
'UNAVL'                     FSMINPUT='UNAVL';
 080B|0813|0814|081B        SNC(0:15)=(X'080B'|X'0813'|X'0814'|X'081B');
 080B|0813|0814|081B|0846   SNC(0:15)=(X'080B'|X'0813'|X'0814'|X'081B'|X'0846');
 0814                       SNC(0:15)=X'0814';
 081B                       SNC(0:15)=X'081B';
 0846                       SNC(0:15)=X'0846';

END FSM_INPUT_DEFINITION;
```

```
                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│                                CT_RCV_RQ_EXP                             │
│                                                                          │
│       THIS CORRELATION TABLE CONTAINS INFORMATION FOR ALL EXPEDITED-FLOW │
│       REQUESTS RECEIVED.  IT IS USED TO ENFORCE PROPER SENDING OF        │
│       RESPONSES TO THESE REQUESTS.                                       │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
ENTITY(CT_RCV_RQ_EXP_ENTRY),
      2 CT_RCV_RQ_EXP_ID FIXED(15) BIN,
      2 CT_RCV_RQ_EXP_DFC_RQ_CODE BIT(8),
      2 CT_RCV_RQ_EXP_EXR_SENSE BIT(16);




                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│                                CT_SEND_RQ_EXP                            │
│                                                                          │
│       THIS CORRELATION TABLE CONTAINS INFORMATION FOR ALL EXPEDITED-FLOW │
│       REQUESTS SENT.  IT IS USED TO CHECK PROPER RECEIVING OF            │
│       RESPONSES TO THESE REQUESTS.                                       │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
 ENTITY(CT_SEND_RQ_EXP_ENTRY),
       2 CT_SEND_RQ_EXP_ID FIXED(15) BIN,
       2 CT_SEND_RQ_EXP_DFC_RQ_CODE BIT(8);




                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│                                   CT_NORM                                │
│                                                                          │
│       THIS IS THE FORMAT OF THE SEND AND RECEIVE NORMAL-FLOW             │
│       CORRELATION TABLE ENTRIES                                          │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
 ENTITY(CT_NORM_ENTRY),
       2 CT_ENTRY_TYPE BIT(2),
       2 CT_CONTROL_INFO,
         3 CT_BEG_SNF FIXED(15) BIN,
         3 CT_END_SNF FIXED(15) BIN,
         3 CT_RSP_TO_NOT_CANCEL BIT(1),
         3 CT_EXR_SENSE_FOR_NOT_CANCEL BIT(32),
         3 CT_EXR_SENSE_FOR_CANCEL BIT(32),
       2 CT_RH_RU_INFO,
         3 CT_RU_CTGY BIT(2),
         3 CT_RQI,
           4 CT_DR1I BIT(1),
           4 CT_DR2I BIT(1),
           4 CT_ERI,
             5 CT_RTI BIT(1),
         3 CT_QRI BIT(1),
         3 CT_BBI BIT(1),
         3 CT_EBI BIT(1),
         3 CT_CDI BIT(1),
         3 CT_DFC_RQ_CODE BIT(8);
```

CHAPTER 6. OVERVIEW OF NETWORK SERVICES

This chapter provides an overview of Chapters 7, 8, and 9, which present definitions for the SSCP and LU services layers consisting of SSCP and LU services managers, session network services, and undefined protocol machines. An overview of the PU services layer is given in Chapter 10, and detailed descriptions of the PU services components are given in Chapters 11, 12, and 13.

NAU.SVC

The NAU services components of the SSCPs and of the PUs and LUs in the network interact to monitor and control LUs, links, link connections, and routing tables. The interaction is based on the division of the network into domains. Each domain consists of an SSCP and the PUs, LUs, link stations, links and associated resources that the SSCP controls by having the capability to activate them (e.g., via ACTPU, ACTLU, and ACTLINK).

For a given NAU, the services manager and the session network services (SNS) component for its various half-sessions jointly form a NAU services layer.

- NAU services managers control network operation by exchanging network services RUs with one another, using SSCP based sessions, i.e., SSCP-SSCP, SSCP-PU, and SSCP-LU sessions.

- Session network services (SNS) are located in half-sessions and record state information on a half-session basis, to enforce the correct ordering of network services RUs.

The specific requests and responses flowing between the NAU services components described in the following chapters are called network services (NS) requests and responses. Both same-domain (SSCP-PU and SSCP-LU) and cross-domain (SSCP-SSCP) network services flows are described. These flows are illustrated in Figure 6-5 on page 6-7.

Figures 6-1 through 6-4 provide an overview of nodes, emphasizing NAU services. Each node type is shown in a separate figure.

SSCP.SVC

The SSCP.SVC layer includes the SSCP.SVC_MGR and SNS components. SSCP.SVC_MGR exchanges network services RUs with a corresponding services manager--SSCP.SVC_MGR (using the SSCP-SSCP session), LU.SVC_MGR (using the SSCP-LU

session), or PU.SVC_MGR (using the SSCP-PU session). For configuration services, the SNS component consists only of SNS.RCV and SNS.SEND. Figure 6-6 illustrates the structure of SSCP.SVC.

LU.SVC

The LU.SVC layer includes the LU.SVC_MGR and SNS components. LU.SVC_MGR exchanges network services RUs with a corresponding SSCP.SVC_MGR on the associated SSCP-LU session. Figure 6-7 illustrates the structure of LU.SVC.

PU.SVC

PU.SVC consists of the PU.SVC_MGR and SNS components. PU.SVC_MGR exchanges network services RUs with a corresponding SSCP.SVC_MGR on the associated SSCP-PU session. The SNS component of PU.SVC consists only of SNS.RCV and SNS.SEND. Figure 6-8 illustrates the structure of PU.SVC.

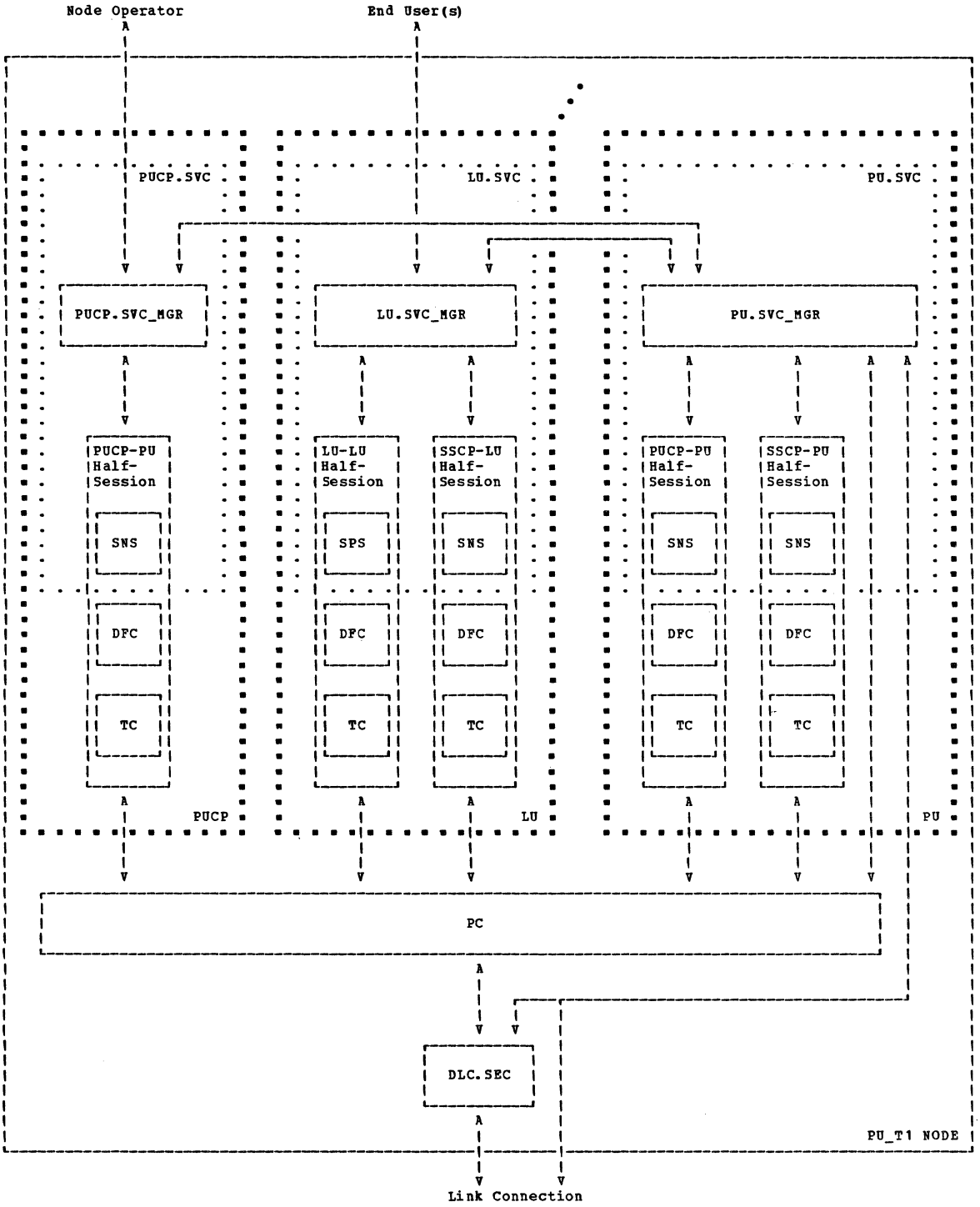
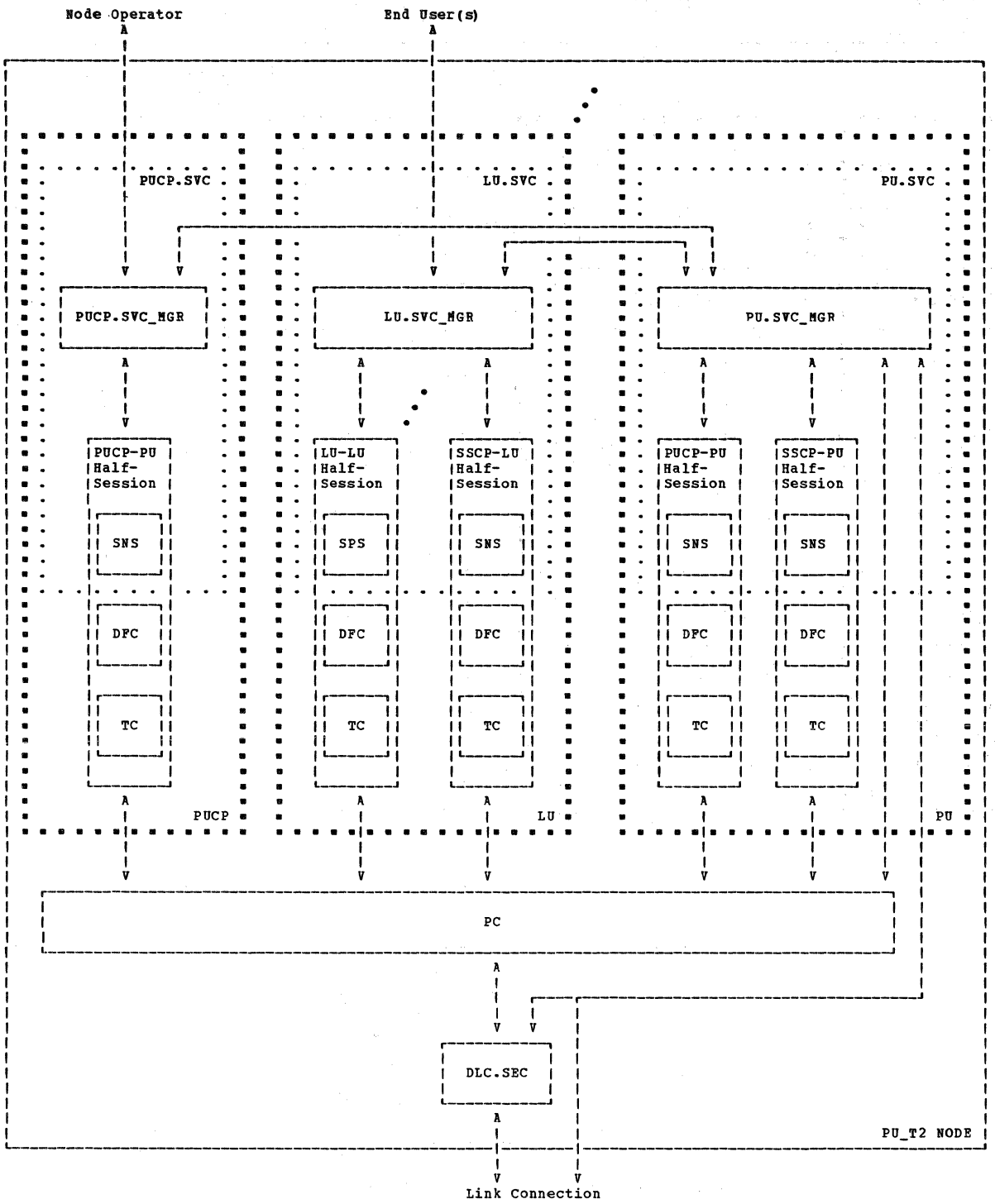
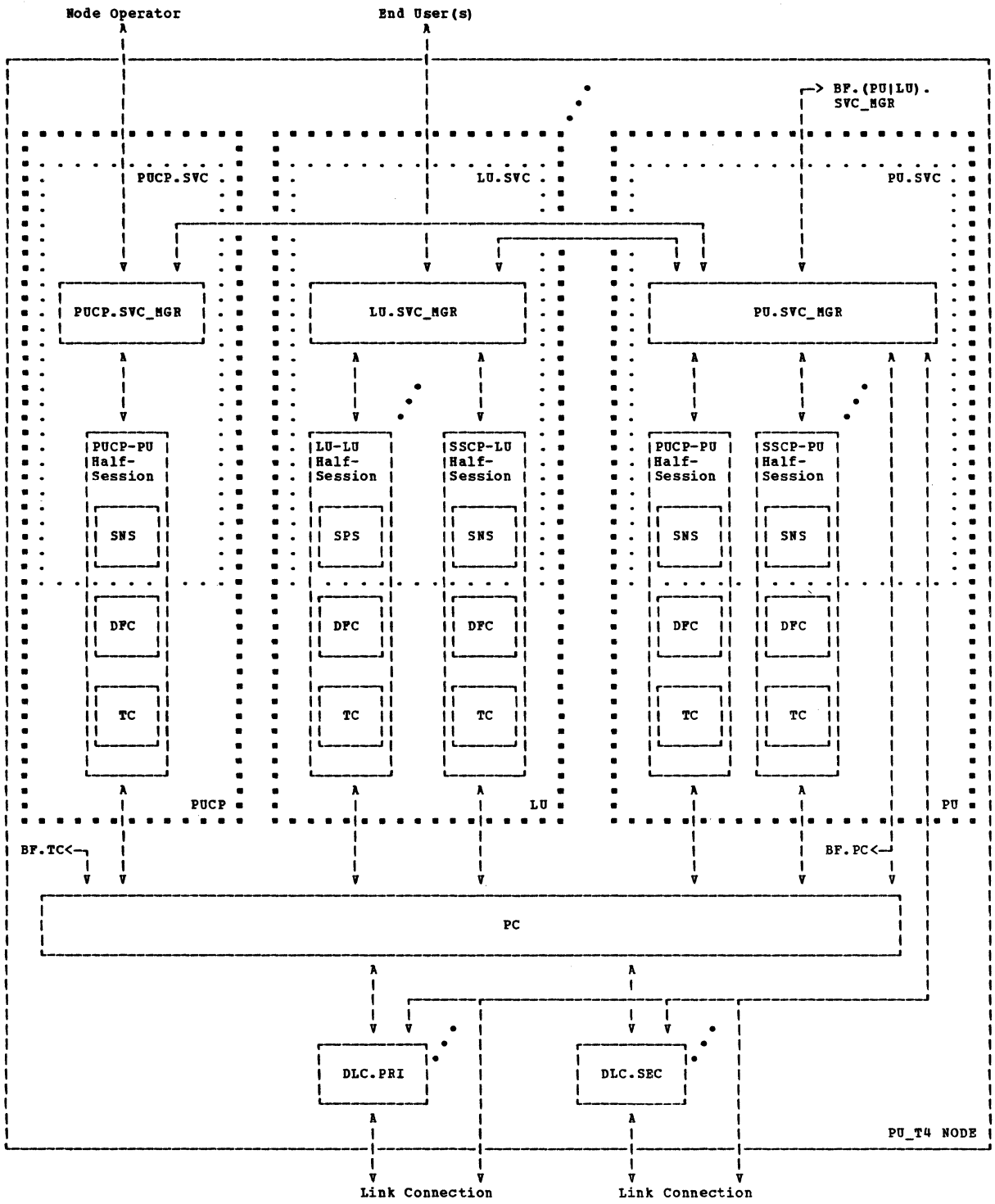


Figure 6-1. Overview of a PU_T1 Peripheral Node, Emphasizing NAU Services



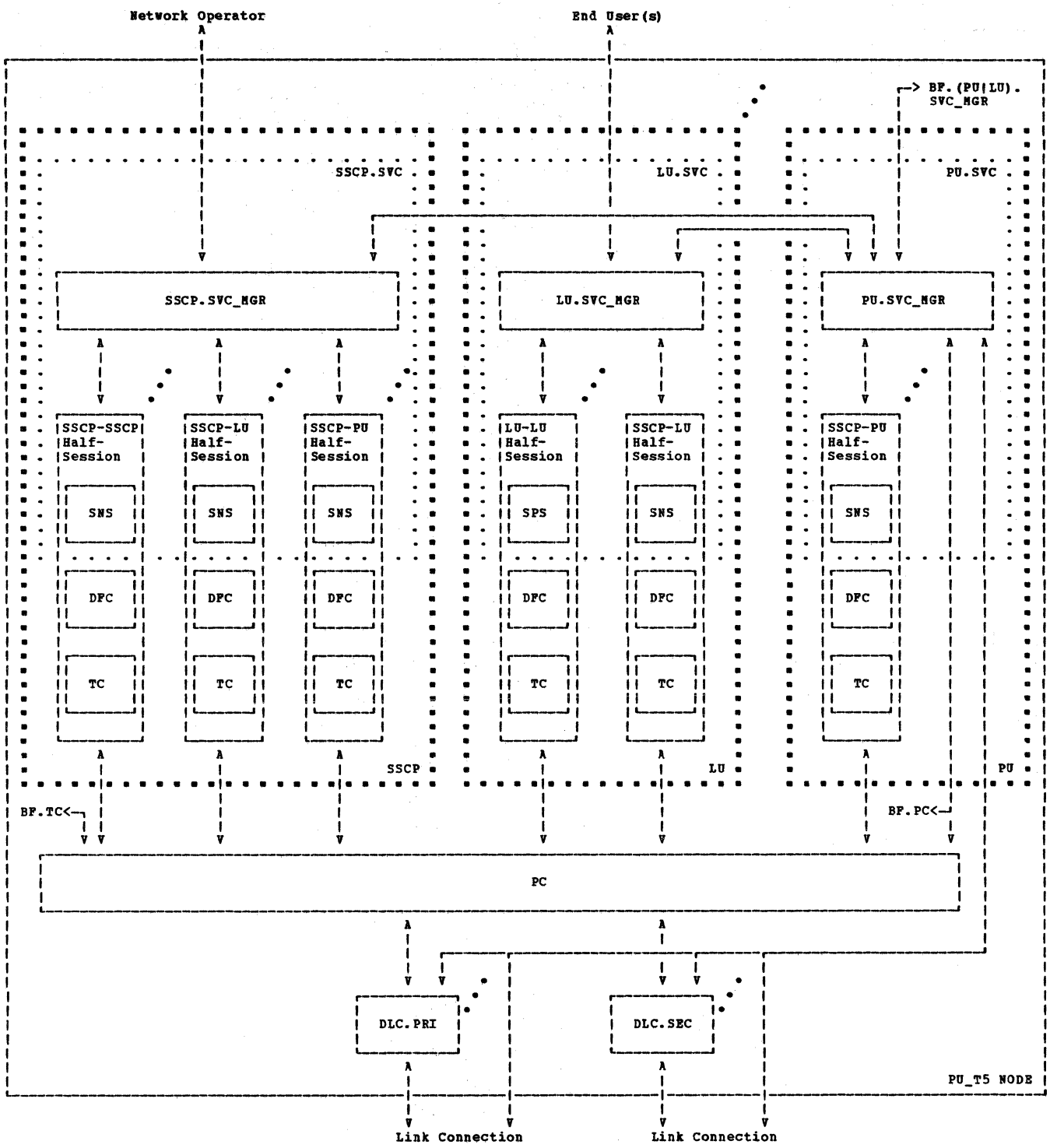
Note:
A PU_T2 node differs from a PU_T1 node in that each LU in a PU_T2 node has the capability to support multiple LU-LU sessions.

Figure 6-2. Overview of a PU_T2 Peripheral Node, Emphasizing NAU Services



Note:
The structural overview of the boundary function, including BF.(PU|LU).SVC_MGR, BF.TC, and BF.PC, is illustrated in Chapter 1.

Figure 6-3. Overview of a PU_T4 Subarea Node, Emphasizing NAU Services



Notes:

1. The structural overview of the boundary function, including BF.(PU|LU).SVC_MGR, BF.TC, and BF.PC, is illustrated in Chapter 1.

2. A PU_T5 node differs from a PU_T4 node in that a PU_T5 node contains an SSCP instead of a PUCP.

Figure 6-4. Overview of a PU_T5 Subarea Node, Emphasizing NAU Services

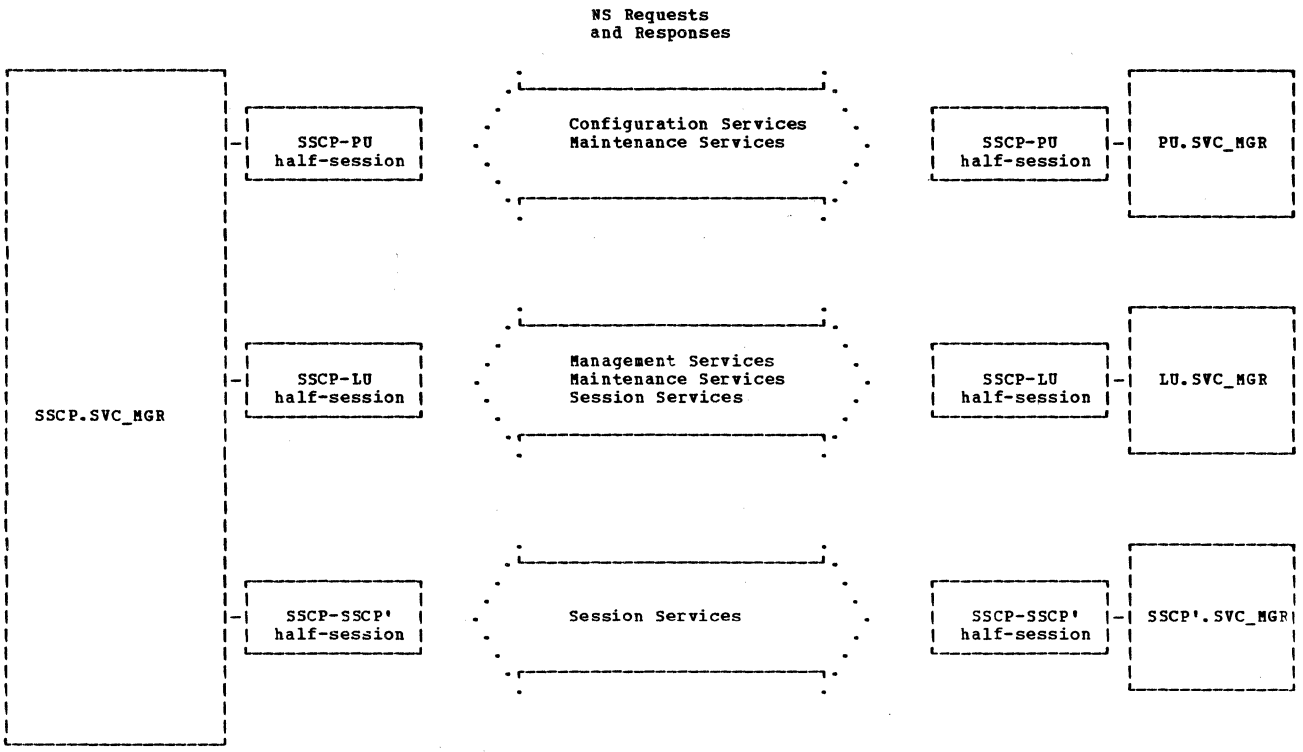


Figure 6-5. Relationship of SSCP, PU, and LU Services Managers to Network Services Request and Response Flows

NETWORK SERVICES CATEGORIES

The NS requests and responses, described in Chapters 7, 8 and 9, are classified into the following categories:

- Configuration services (See Chapter 7)
- Session services (See Chapter 8)
- Maintenance services (See Chapter 9)
- Management services (See Chapter 9)

Figure 6-5 on page 6-7 illustrates, by category, which RUs flow between the NAU services managers.

CONFIGURATION SERVICES

Configuration services protocol machines are distributed between the SSCPs and all the PUs in the network on a domain basis; there are no configuration services in LUs. Configuration services support the control of link-level procedures such as the activation and deactivation of switched and nonswitched links, control of link station contact, and the loading and dumping of nodes.

The configuration services within the SSCP have such implementation- and installation-dependent information as the network addresses and characteristics of all PUs within its domain, and the appropriate telephone numbers associated with switched link connection operations. Each PU has available to it implementation- and installation-dependent information about the network, such as its own network address and characteristics.

SESSION SERVICES

Session services protocol machines are distributed between the SSCPs and LUs in the network; there are no session services in PUs. Session services provide facilities for the SSCP to support LUs in initiating and terminating LU-LU sessions.

MAINTENANCE AND MANAGEMENT SERVICES

Maintenance services protocol machines are distributed between the SSCPs and both the PUs and LUs of the network, or between communication network management (CNM) components as described below. These protocols support the execution of link-level traces, the testing of various network resources (e.g., a link or an LU), and the reporting of network resource status.

Management services protocol machines are distributed between SSCPs and LUs that support CNM applications in order to support maintenance services that are operating as part

of CNM. Management services allow the CNM application, associated with the LU, to use the existing LU-SSCP and SSCP-PU sessions to access the CNM component associated with a specific node. This CNM component access is accomplished using network names. The SSCP translates the network name to a network address. Chapter 9 describes communication network management in more detail.

NETWORK SERVICES FORMATS

All NS requests and responses are sent on the normal flow with the RU category indicating FMD. When responses are requested or returned, the Definite Response 1 indicator (DR1I) in the RH is set to DR1.

NS requests flowing in SSCP-LU sessions from the LU to the SSCP may, in general, be field-formatted (RH Format indicator set to NSH) or character-coded (RH Format indicator set to ¬NSH); NS requests flowing from the SSCP to the LU, or in SSCP-SSCP or SSCP-PU sessions are always field-formatted.

Field-formatted NS requests consist of an initial three-byte NS header, followed by additional fields that vary by request. The NS header has the following format.

- The first byte has two subfields:

  -Bits 0-1 denote whether the request involves a service related to a PU, an LU, or possibly either:

  00 not specified (may be either)
  01 PU
  10 LU
  11 reserved

  -Bits 2-7 currently have only one value defined:

  000001 network services

  The second byte has three subfields:

  -Bit 0 denotes whether the request is used on a same-domain session or on a cross-domain session:

  0 same-domain, i.e., SSCP-LU or SSCP-PU sessions
  1 cross-domain, i.e., SSCP-SSCP sessions

  -Bit 1 is reserved

  -Bits 2-7 indicate the NS category; values currently defined are listed below, with the values of bits 0-1 (although these are separate subfields) merged with those for bits 2-7 to give byte values:

same-domain	cross-domain	
X'02'	X'82'	configuration services
X'03'	X'83'	maintenance services
X'04'	X'84'	measurement services
X'05'	X'85'	network operator services
X'06'	X'86'	session services
X'08'	-	management services

Two of these categories--measurement services and network operator services--are not described in this book.

- The third byte indicates the particular request code (e.g., CONTACT) within the NS category.

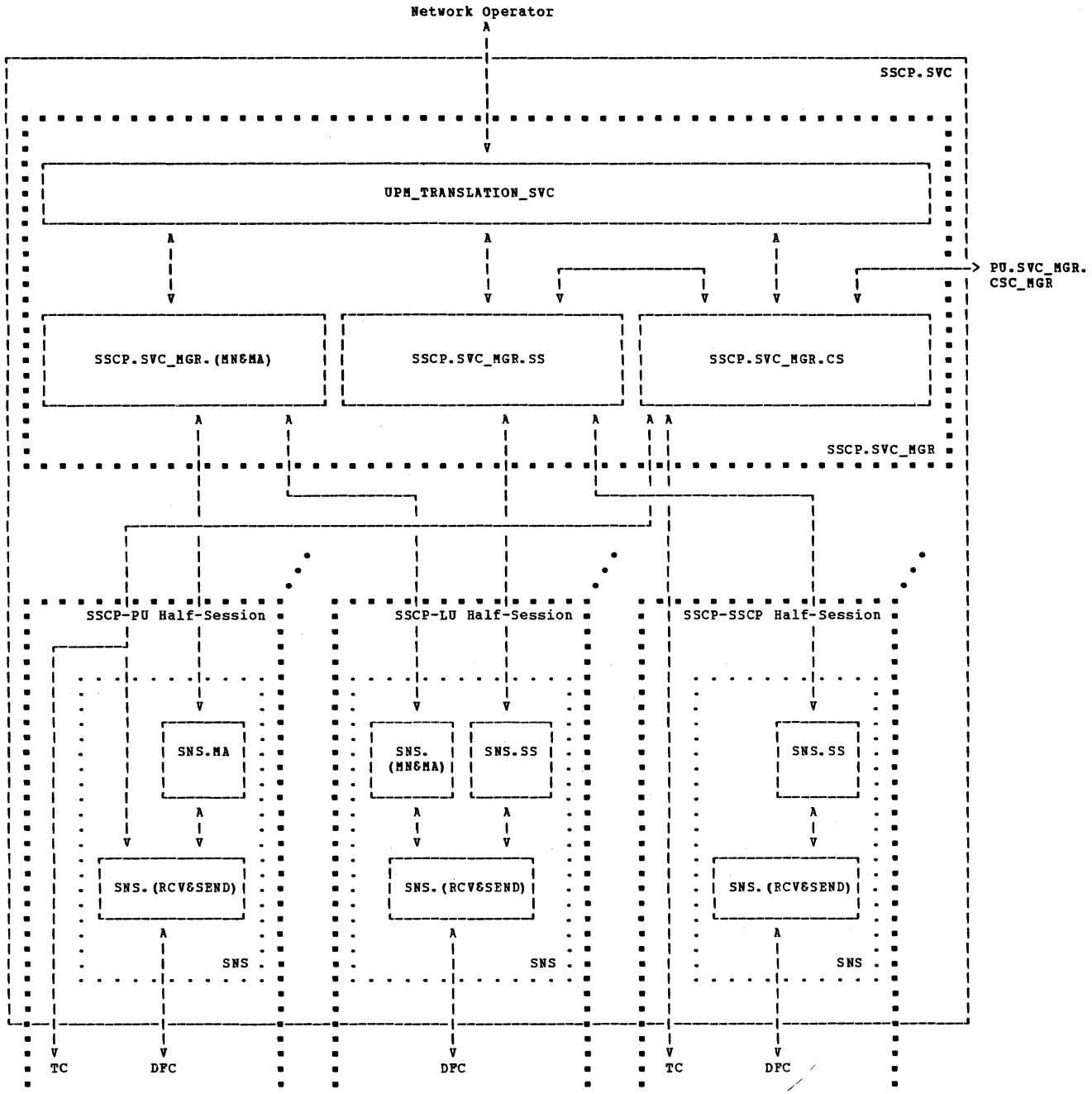
Full details on the RU formats for field-formatted NS requests (and responses) are given in Appendix E.

Control vectors and control lists are maintained at NAUs, and are set or accessed by other NAUs using specific requests and their responses. Control vectors are referred to by key, while control lists are referred to by type. Appendix E defines the formats and uses within RUs of control vectors and control lists. The SETCV function is discussed in Chapters 7 and 9, and the DSRLST function is discussed in Chapter 8.

A network name is the name by which a PU, an LU, a link, or a link station is known within NTWK.SNA. Network names used across domains must be unique within the multiple-domain network. Network Name fields and Uninterpreted Name fields (described in Chapter 8) include a Type field (denoting the resource type), a Length field, and the name itself. The following values are defined for the Type field:

hex value	type name identified
X'00'	none (no name present)
X'F1'	PU name
X'F3'	LU name
X'F5'	test procedure name
X'F7'	adjacent link station name
X'F9'	link name

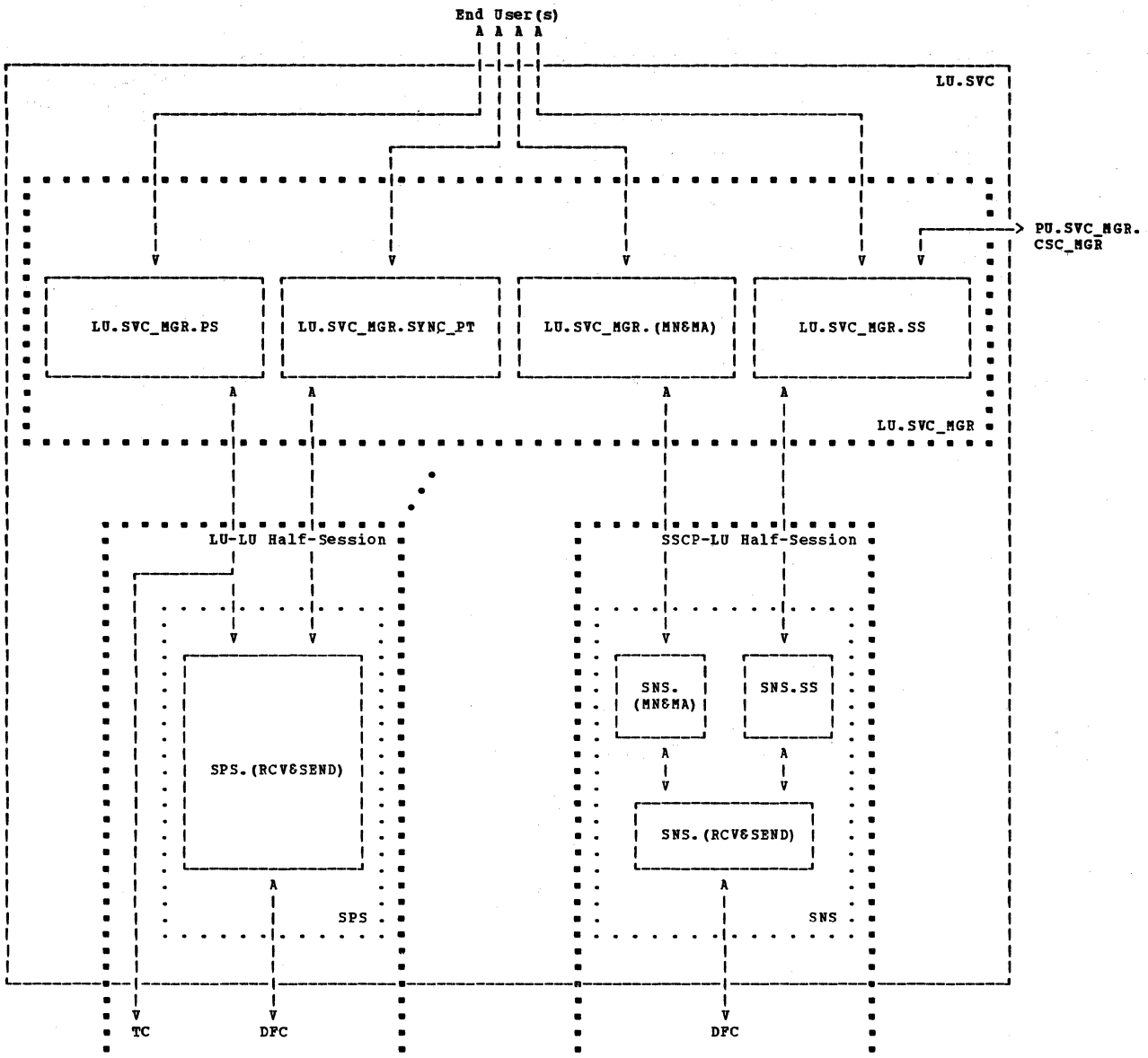
Character-coded NS requests contain RUs consisting of character strings that can be translated into equivalent field-formatted RUs. A translation protocol is provided by the SNS.RCV component of SSCP.SVC to translate the character-coded requests received from DFC into field-formatted requests.



Note:
The components of SSCP.SVC are described in the following chapters:

Component	Chapter	Component	Chapter	Component	Chapter
SNS	6	SNS.SEND	6	SSCP.SVC_MGR.MA	9
SNS.MA	9	SNS.SS	8	SSCP.SVC_MGR.MN	9
SNS.MN	9	SSCP.SVC_MGR	6	SSCP.SVC_MGR.SS	8
SNS.RCV	6	SSCP.SVC_MGR.CS	7	UPM_TRANSLATION_SVC	6

Figure 6-6. Structure of SSCP.SVC

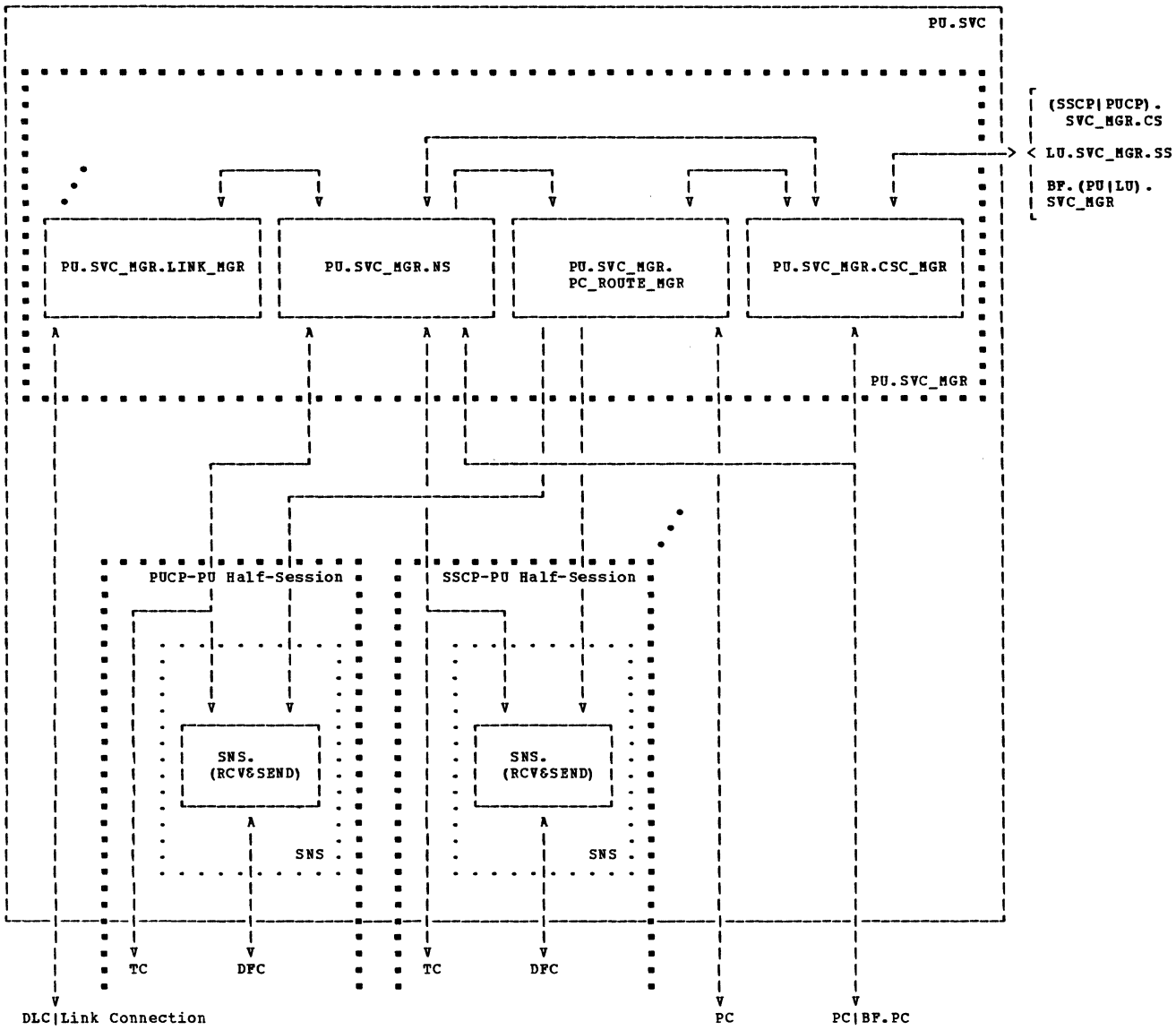


Note:
The components of LU.SVC are described in the following chapters:

Component	Chapter	Component	Chapter	Component	Chapter
LU.SVC_MGR	6	LU.SVC_MGR.SYNC_PT	**	SNS.SEND	6
LU.SVC_MGR.MA	9	SNS	6	SNS.SS	8
LU.SVC_MGR.MN	9	SNS.MA	9	SPS	*
LU.SVC_MGR.PS	*	SNS.MN	9	SPS.RCV	*
LU.SVC_MGR.SS	8	SNS.RCV	6	SPS.SEND	*

* These components are described in SNA LU-LU Session Types.
**Details of the LU.SVC_MGR.SYNC_PT are not defined in this book.

Figure 6-7. Structure of LU.SVC



Note:
The components of PU.SVC are described in the following chapters:

Component	Chapter	Component	Chapter	Component	Chapter
PU.SVC_MGR	10	PU.SVC_MGR.NS	11	SNS.RCV	6
PU.SVC_MGR.CSC_MGR	13	PU.SVC_MGR.PC_ROUTE_MGR	12	SNS.SEND	6
PU.SVC_MGR.LINK_MGR	*	SNS	6		

* The PU.SVC_MGR.LINK_MGR is not described in this book.

Figure 6-8. Structure of PU.SVC

This page intentionally left blank

FAPL PROCEDURES

SNS.RCV

SNS.RCV is shown on page 6-17, and is basically a router. It determines on which session the request/response is flowing and routes the request/response to the appropriate SNS or services manager component based upon RU category. The same router is used for all half-sessions.

SNS.SEND

SNS.SEND shown on page 6-18 is the procedure to which the services managers send requests and responses to be forwarded to DFC.SEND.

UPM_TRANS_TO_FIELD_FORMATTED

This procedure is called by SNS.RCV to translate character coded requests, received from DFC, into field-formatted requests that can be processed by the NAU.SVC components.

UPM_TRANSLATION_SVC

This procedure translates input from the network operator into requests that are to be processed by the SSCP services manager components (Chapters 7, 8, and 9). Any responses that result from these requests are sent to this UPM. This UPM is defined in Chapter 6 because it is common to Chapters 7, 8, and 9.

```
SNS.RCV: PROCEDURE;

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  ROUTES CURRENT NS RQ OR RSP TO CORRECT NETWORK SERVICES PROCEDURE.
 │
 │      INPUT:     RQ OR RSP FROM DFC.RCV
 │
 │      OUTPUT:    -RSP TO DFC.SEND OR RQ/RSP TO PROCEDURE THAT PROCESSES THE REQUEST.
 │
 │      NOTE:      NETWORK  OPERATOR SERVICES  AND  MEASUREMENT  SERVICES RUS  ARE  NOT
 │                 DEFINED IN THIS BOOK.
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
 SELECT ANYORDER(SCB.TYPE_OF_SESSION);
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │   HANDLE SSCP-SSCP SESSION REQUEST/RESPONSES   │
                  └──────────────────────────────────────────┘
                                                                                */
 . WHEN(SSCP_SSCP)
 .   IF NS_CATEGORY(1:7) = SESSION_SERVICES THEN
 .     SEND MU TO SNS.SS.RCV;                          /* CHAPTER 8            */
 .   ELSE
 .     DO;
 .     . IF RRI = RQ THEN                              /* IF REQUEST  SEND -RSP */
 .     .   DO;                                         /* CATEGORY NOT SUPPORTED */
 .     .   . CALL CHANGE_MU_TO_NEG_RSP(X'1007');       /* APPENDIX B           */
 .     .   . SEND MU TO DFC.SEND;                      /* -RSP TO CHAPTER 5    */
 .     .   END;
 .     . ELSE                                          /* IF RESPONSE, LOG     */
 .     .   DO;                                         /* AND DISCARD          */
 .     .   . CALL UPM_LOG;                             /* APPENDIX B           */
 .     .   . DISCARD MU;
 .     .   END;
 .     END;
 .
 .
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │    HANDLE SSCP-LU SESSION REQUEST/RESPONSES    │
                  └──────────────────────────────────────────┘
                                                                                */
 . WHEN(SSCP_LU)
 .   DO;
 .   . IF FI = ¬NSH THEN                               /* IF NOT FIELD FORMATTED */
 .   .   CALL UPM_TRANS_TO_FIELD_FORMATTED;            /* PAGE 6-19            */
 .   . ELSE
 .   .   SELECT ANYORDER;
 .   .   .
 .   .   .
 .   .   . WHEN(NS_CATEGORY(1:7) = MANAGEMENT_SERVICES)
 .   .   .
 .   .   .   SEND MU TO SNS.MN.RCV;                   /* CHAPTER 9            */
 .   .   .
 .   .   . WHEN(NS_CATEGORY(1:7) = MAINTENANCE_SERVICES)
 .   .   .   SEND MU TO SNS.MA.RCV;                   /* CHAPTER 9            */
 .   .   .
 .   .   .
 .   .   . WHEN(NS_CATEGORY(1:7) = SESSION_SERVICES)
 .   .   .   SEND MU TO SNS.SS.RCV;                   /* CHAPTER 8            */
 .   .   .
 .   .   .
 .   .   . OTHERWISE
 .   .   .   DO;
 .   .   .   . IF RRI = RQ THEN                       /* IF REQUEST SEND -RSP */
 .   .   .   .   DO;                                  /* CATEGORY NOT SUPPORTED */
 .   .   .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'1007'); /* APPENDIX B         */
 .   .   .   .   . SEND MU TO DFC.SEND;              /* -RSP TO CHAPTER 5    */
 .   .   .   .   END;
 .   .   .   . ELSE                                  /* IF RESPONSE, LOG     */
 .   .   .   .   DO;                                 /* AND DISCARD          */
 .   .   .   .   . CALL UPM_LOG;                     /* APPENDIX B           */
 .   .   .   .   . DISCARD MU;
 .   .   .   .   END;
 .   .   .   END;
```

```
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │ HANDLE SSCP-PU SESSION REQUEST/RESPONSES │
                    └──────────────────────────────────────┘
                                                                        */
   .   .   . WHEN(SSCP_PU)
   .   .   .   SELECT ANYORDER;
   .   .   .   .
   .   .   .   .
   .   .   .   . WHEN(NS_CATEGORY(1:7) = CONFIGURATION_SERVICES)
   .   .   .   .   IF DEF = 0 THEN                    /* PU IS DESTINATION     */
   .   .   .   .     SEND MU TO PU.SVC_MGR.NS.RCV;     /* CHAPTER 11            */
   .   .   .   .   ELSE                               /* SSCP IS DESTINATION   */
   .   .   .   .     SEND MU TO SSCP.SVC_MGR.CS.RCV;   /* CHAPTER 7             */
   .   .   .   .
   .   .   .   . WHEN(NS_CATEGORY(1:7) = MAINTENANCE_SERVICES)
   .   .   .   .   IF DEF = 0 THEN                    /* PU IS DESTINATION     */
   .   .   .   .     SEND MU TO PU.SVC_MGR.NS.RCV;     /* CHAPTER 11            */
   .   .   .   .   ELSE                               /* SSCP IS DESTINATION   */
   .   .   .   .     SEND MU TO SNS.MA.RCV;            /* CHAPTER 9             */
   .   .   .   .
   .   .   .   .
   .   .   .   . OTHERWISE
   .   .   .   .   DO;
   .   .   .   .   . IF RRI = RQ THEN                 /* IF REQUEST SEND -RSP  */
   .   .   .   .   .   DO;                             /* CATEGORY NOT SUPPORTED*/
   .   .   .   .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'1007');  /* APPENDIX B   */
   .   .   .   .   .   . SEND MU TO DPC.SEND;         /* -RSP TO CHAPTER 5     */
   .   .   .   .   .   . END;
   .   .   .   .   . ELSE                             /* IF RESPONSE, LOG      */
   .   .   .   .   .   DO;                             /* AND DISCARD           */
   .   .   .   .   .   . CALL UPM_LOG;                /* APPENDIX B            */
   .   .   .   .   .   . DISCARD MU;
   .   .   .   .   .   END;
   .   .   .   .   END;
   .   .   .   END;
   .   .   END;
   .   END;
   END;
   RETURN;
END SNS.RCV;
```

```
SNS.SEND: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  ROUTES CURRENT NETWORK SERVICES RQ OR RSP TO DFC.SEND.       │
  │                                                                              │
  │      INPUT:     RQ OR RSP FROM NAU SERVICES COMPONENT.                       │
  │                                                                              │
  │      OUTPUT:    RQ OR RSP TO DFC.SEND.                                       │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SEND NU TO DFC.SEND;                                  /* CHAPTER 5            */
  RETURN;
END SNS.SEND;
```

```
UPM_TRANS_TO_FIELD_FORMATTED: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TRANSLATES RECEIVED CHARACTER-CODED REQUESTS INTO FIELD-FORMATTED │
  │                  REQUESTS.                                                   │
  │                                                                              │
  │      INPUT:     RQ FROM SNS.RCV                                              │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                                                /*
                    ┌──────────────────────────────────────┐
                    │          TRANSLATE TO FIELD FORMATTED          │
                    └──────────────────────────────────────┘
                                                                                */
  FI = NSH                                  /* INDICATE RU IS FIELD FORMATTED   */
       RETURN;
END UPM_TRANS_TO_FIELD_FORMATTED;
```

```
UPM_TRANSLATION_SVC: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TRANSLATES  INPUT FROM  THE  NETWORK OPERATOR  INTO  REQUESTS TO  BE  │
   │                 PASSED  TO CS.SEND  (CHAPTER 7),  SS.SEND (CHAPTER  8), MA.SEND,  OR  │
   │                 MN.SEND (CHAPTER  9) TO BE  PROCESSED.  RESPONSES TO  THESE REQUESTS  │
   │                 ARE RECEIVED  FROM CS.RCV, SS.RCV, MA.RCV,  OR MN.RCV AND  PASSED TO  │
   │                 THE NETWORK OPERATOR.                                                 │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  RETURN;
END UPM_TRANSLATION_SVC;
```

SSCP.SVC_MGR.CS GENERAL DESCRIPTION

Every network contains one or more system services control points (SSCPs), each of which manages a portion of the network called its domain. Each SSCP has a corresponding SSCP services manager with a configuration services component, the SSCP.SVC_MGR.CS. The function of the SSCP.SVC_MGR.CS is to control the physical configuration of its domain by managing the link connections, link stations, and physical units within the domain. The SSCP.SVC_MGR.CS initially activates the domain at start-up time as specified by the network operator, modifies it subsequently, restarts elements of the domain, and shuts down the domain.

The SSCP.SVC_MGR.CS is assisted by the PU.SVC_MGRs within its domain, through the exchange of configuration services requests and responses.

A PU control point (PUCP) exists in a non-PU_T5 node and is a functional subset of an SSCP. The subset of functions that the PUCP provides is implementation-defined. Minimally, a PUCP provides sufficient capability to activate the PU in the node in which the PUCP resides and a locally-attached link. Basically, the functions that the PUCP provides from within the node are those that the SSCP.SVC_MGR.CS also provides. (See Chapter 11 for more information on the PUCP.)

Information about each resource within the SSCP's domain is contained in the domain resource list (see Appendix A), which is created by an implementation- and installation-dependent process. The domain resource list is managed by the SSCP.SVC_MGR.CS, and a representation of a domain resource can be added to, or deleted from, the domain resource list at any time.

A PU, link, or link station may be activated by more than one SSCP, in which case that resource is said to be under shared control This sharing of resources is, however, transparent to the SSCP, which is aware only of its own control over the resource. The PU.SVC_MGR.NS supervises the sharing of the resources associated with its node (see Chapter 11).

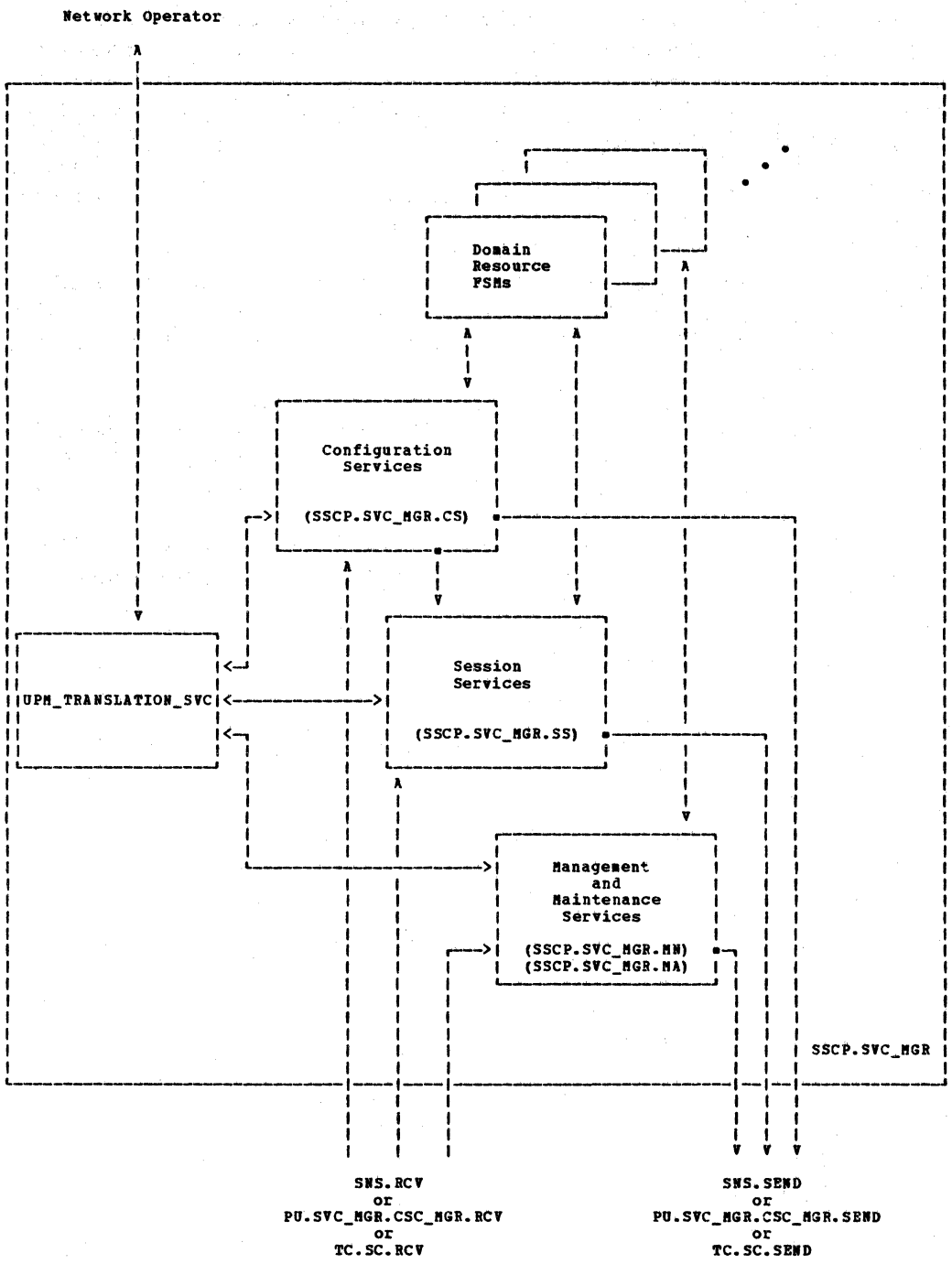


Figure 7-1. SSCP.SVC_MGR Structure

The SSCP.SVC_MGR.CS accesses domain resource (DOM_RES) finite-state machines, each of which represents a resource of the domain and provides information about the state of the resource in relation to the SSCP. The domain resource FSMs describe the interaction of the SSCP with given resources. A single domain resource FSM of a particular kind exists for each resource in the SSCP's domain (e.g., one FSM_LINK_ACT_DOM_RES for a link or one FSM_ALS_CONTACT_DOM_RES for an adjacent link station represented in a given node of the domain).

In contrast to domain resource FSMs, node resource FSMs describe the interaction of multiple control points with given resources (see Figure 7-2). Node resource FSMs are discussed in Chapter 11.

The states of the node resource FSM for a given shared resource are coupled with those of the domain resource FSMs representing the interaction of different SSCPs with the same shared resource, e.g., the node resource FSM goes active when the first corresponding domain resource FSM does, and is reset only after all corresponding domain resource FSMs are reset.

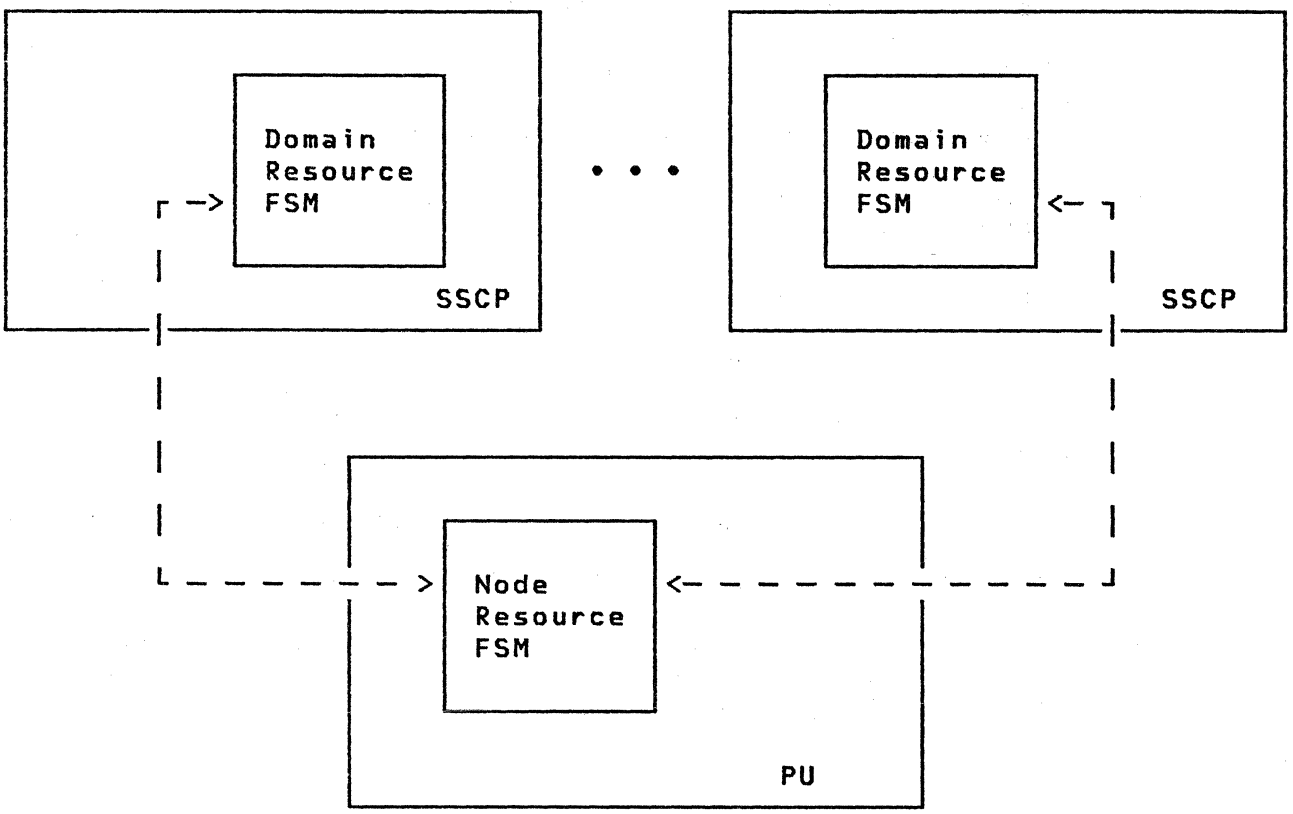


Figure 7-2. Relationships Between Domain Resource FSMs in SSCPs and Node Resource FSMs in PUs

SSCP.SVC_MGR STRUCTURE

The SSCP.SVC_MGR (Figure 7-1) is composed of the following network services elements:

- Configuration services (Chapter 7)
- Session services (Chapter 8)
- Management and maintenance services (Chapter 9)
- Domain resource finite-state machines (Chapters 7, 8, and 9)
- UPM_TRANSLATION_SVC (Chapter 6)

The configuration services (CS) component (Figure 7-3) is composed of the following elements:

- CS.SEND, which handles the sending of all requests, responses, and other signals to SNS.SEND (Chapter 6) and to PU.SVC_MGR.CSC_MGR.SEND (Chapter 13)
- CS.RCV, which handles the receiving of all requests, responses, and other signals from SNS.RCV (Chapter 6), from PU.SVC_MGR.CSC_MGR.RCV (Chapter 13), and from TC.SC.RCV (Chapter 4)

Session network services (SNS) is a router for requests and responses flowing between CS and DFC. (For more information on SNS, see Chapter 6.)

The network operator drives the SSCP.SVC_MGR.CS by means of an undefined protocol machine (UPM), called UPM_TRANSLATION_SVC. UPM_TRANSLATION_SVC receives input from the network operator and passes the input to CS.SEND. It also receives input from CS.RCV and routes this input to the network operator.

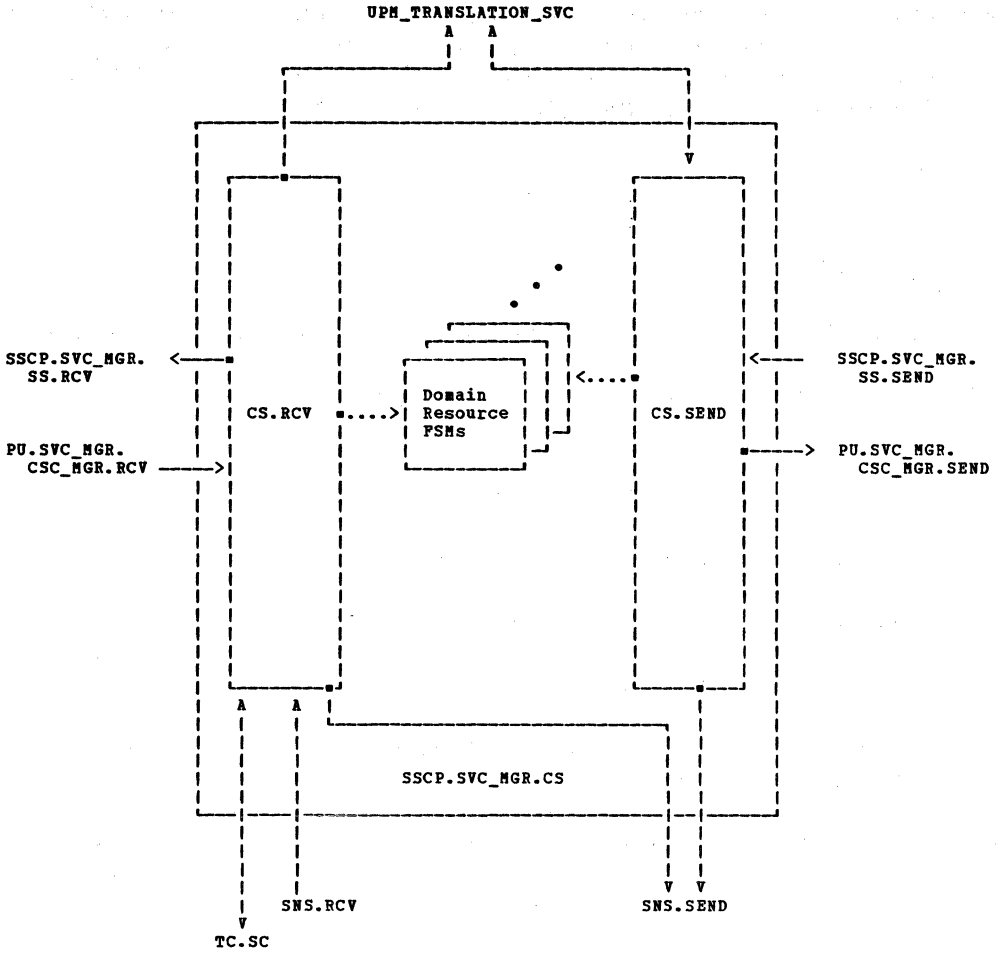


Figure 7-3. SSCP.SVC_MGR.CS Structure

SSCP.SVC_MGR.CS PROTOCOL BOUNDARIES

The protocol boundary information for the SSCP.SVC_MGR.CS depends on the sender of the RU. The specific protocol boundary information for the receipt of RUs is contained in CS.SEND (page 7-48) and CS.RCV (page 7-50).

SSCP.SVC_MGR.CS FUNCTIONS

The SSCP.SVC_MGR.CS coordinates the following functions:

- Dialing-out and enabling for dial-in over switched link connections
- Activating and deactivating links
- Link-level contacting and discontacting of nodes
- Initial program loading of nodes
- Dumping of stored data to the SSCP
- Resetting of appropriate domain resource FSMs upon receipt of an INOP
- Adding entries to, and deleting entries from, the domain resource list

CONFIGURATION SERVICES DATA BASE STRUCTURE

The configuration services data base consists of the node control block (NCB) and the domain resource list, and is structured as shown in Figure 7-4. This structure describes the hierarchy of the many resources within the domain of an SSCP. Details of this structure are given in Appendix A.

The node control block contains the element address of the SSCP and the SSCP identification used in resolving ACTCDRM contention.

The domain resource list consists of a domain resource entry for each resource in the domain. A domain resource entry contains such information as the network name and network address of the resource it is representing.

Every resource is hierarchically associated with its next higher level. This is represented in the domain resource list through the use of associated (backward) resource pointers.

Each link attached to a subarea node is associated with that node's PU. The domain resource entry for the link points to the entry for the subarea PU with which it is associated.

For a given subarea PU, a link may have one or more adjacent link stations associated with it. Only one adjacent link station is associated with a switched link connection. At a subarea node containing a secondary link station, only one adjacent link station (the primary station) is represented. At a subarea node containing the primary link station for a multipoint link, there may be multiple adjacent link stations. Each domain resource entry for an adjacent link station points to the entry for the specific link resource with which it is associated.

The domain resource entry for a peripheral PU points to the entry for the subarea PU's adjacent link station with which the peripheral PU is associated. The network address carried in the entry for the peripheral PU is identical to that for its associated adjacent link station. (The two entries have different resource category values.) Peripheral nodes require boundary function support in a subarea node. The domain resource entry for a peripheral PU contains the local form of the peripheral PU address, as known to the boundary function, and the PU type.

The resource entry for a peripheral LU points to the entry for the peripheral PU with which it is associated. A boundary function LU in a subarea node is required for every LU that exists in a peripheral node attached to the subarea node. The entry for the peripheral LU contains the local form of the peripheral LU address, as known to the boundary function.

The domain resource entry for a subarea LU points either to a subarea PU entry or to another subarea LU entry. If the LU does not support parallel sessions, then the entry points to the LU's associated PU. If the LU does support parallel sessions, then it is represented by a single secondary LU address and by multiple primary LU addresses. The secondary LU entry points to its associated PU, while the primary LU entries point to the secondary LU.

Also included in a domain resource list entry is the SAVE_MU_FOR_RETRY_LIST. This is a pointer to a list that contains requests that are being held in the list pending the activation of a given resource. For example, if the SSCP.SVC_MGR.CS receives an ACTLINK request and the target link's associated PU has not yet been activated (i.e., sent ACTPU), then the ACTLINK is inserted into that PU's list. All requests on this list are removed and reissued after the PU becomes active (i.e., a positive response to ACTPU is received). For more information about this list, see Figure 7-5.

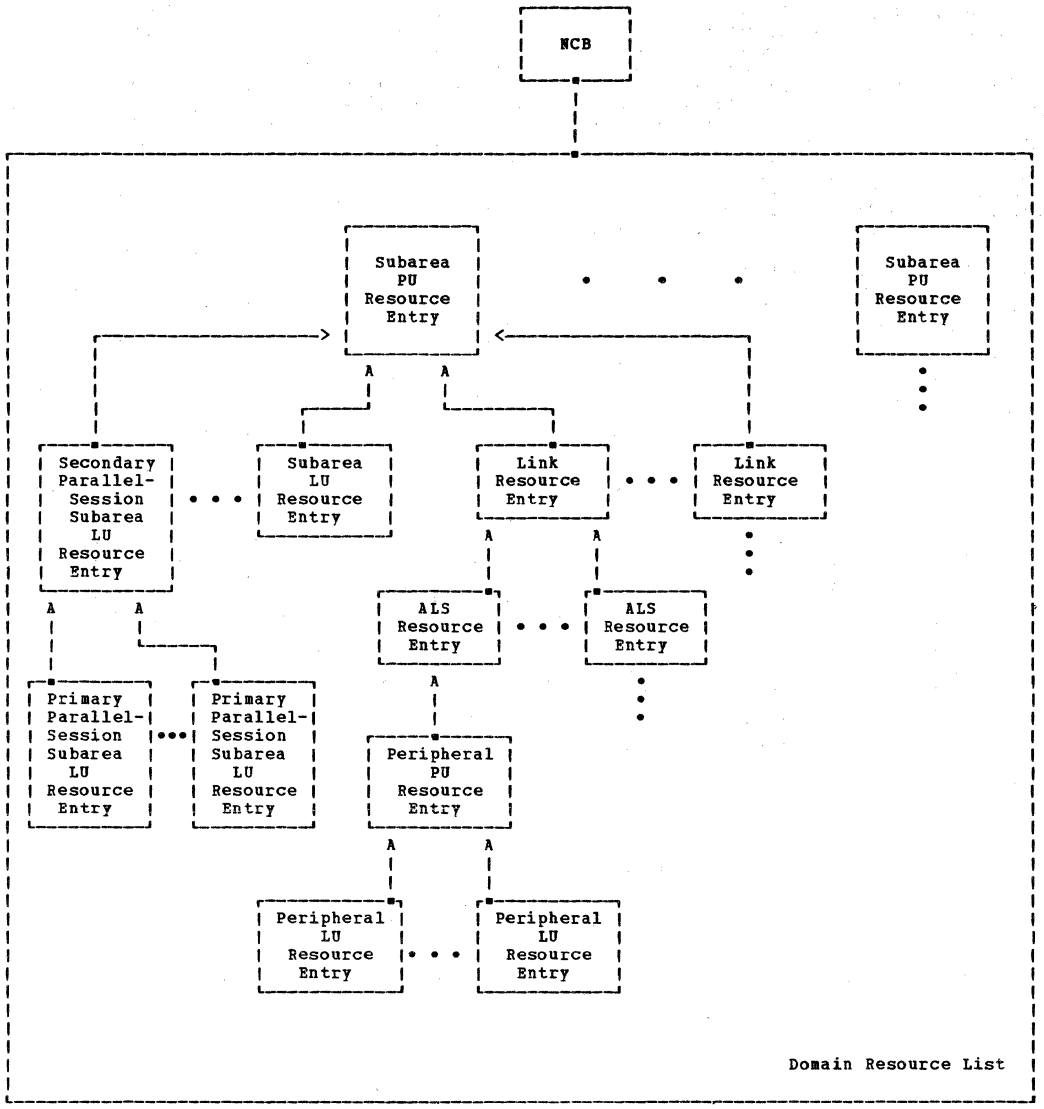


Figure 7-4. Structure of the Domain Resource Data Base

REQUEST	TARGET RESOURCE	RESOURCE CHECKED	(FOR)	LIST USED
ACTLU	Subarea or Peripheral LU	Subarea or Peripheral PU	(ACTPU)	PU
DACTLU	Subarea or Peripheral LU	Subarea or Peripheral PU	(ACTPU)	PU
ACTLINK	Link	Subarea PU	(ACTPU)	PU
DACTLINK	Link	Subarea PU	(ACTPU)	PU
ACTCONNIN	Link	Link	(ACTLINK)	Link
DACTCONNIN	Link	Link	(ACTLINK)	Link
CONNOUT	Link	Link	(ACTLINK)	Link
ABCONNOUT	Link	Link	(ACTLINK)	Link
ABCONN	Link	Link	(ACTLINK)	Link
CONTACT	ALS	Link	(ACTLINK)	Link
DISCONTACT	ALS	Link	(ACTLINK)	Link
IPLINIT	ALS	Link	(ACTLINK)	Link
DUMPINIT	ALS	Link	(ACTLINK)	Link
RPO	ALS	Link	(ACTLINK)	Link
INITPROC	Peripheral PU	ALS	(CONTACT)	ALS
RNAA	Link	Link	(ACTLINK)	Link
	ALS	ALS	(CONTACT)	ALS
	Subarea LU	Subarea LU	(ACTLU)	LU
ADDLINK	Subarea PU	Subarea PU	(ACTPU)	PU
ADDLINKSTA	Subarea PU	Subarea PU	(ACTPU)	PU
DELETENR	Link	Subarea PU	(ACTPU)	PU
	ALS	Subarea PU	(ACTPU)	PU
FNA	Link	Link	(ACTLINK)	Link
	Subarea or Peripheral PU	Subarea or Peripheral PU	(ACTPU)	PU
	Subarea LU	Subarea LU	(ACTLU)	LU

REQUEST OR RESPONSE	TARGET RESOURCE	LIST EMPTIED
+RSP(ACTPU)	Subarea or Peripheral PU	PU
+RSP(ACTLU)	Subarea or Peripheral LU	LU
+RSP(ACTLINK)	Link	Link
CONTACTED(LOADED)	ALS	ALS

Example: When the SSCP.SVC_MGR.CS receives an ACTLINK request, the target link's associated PU is checked to see if it has been sent ACTPU. If it has not, the ACTLINK request is placed on the PU's SAVE_MU_FOR_RETRY_LIST. The requests on this PU's list are removed and reissued after a positive response to ACTPU is received.

Figure 7-5. Summary of Activity Involving the SAVE_MU_FOR_RETRY_LIST

RESET HIERARCHY

The domain resource FSMs contained in the SSCP.SVC_MGR.CS lie in a reset hierarchy shown in Figure 7-6.

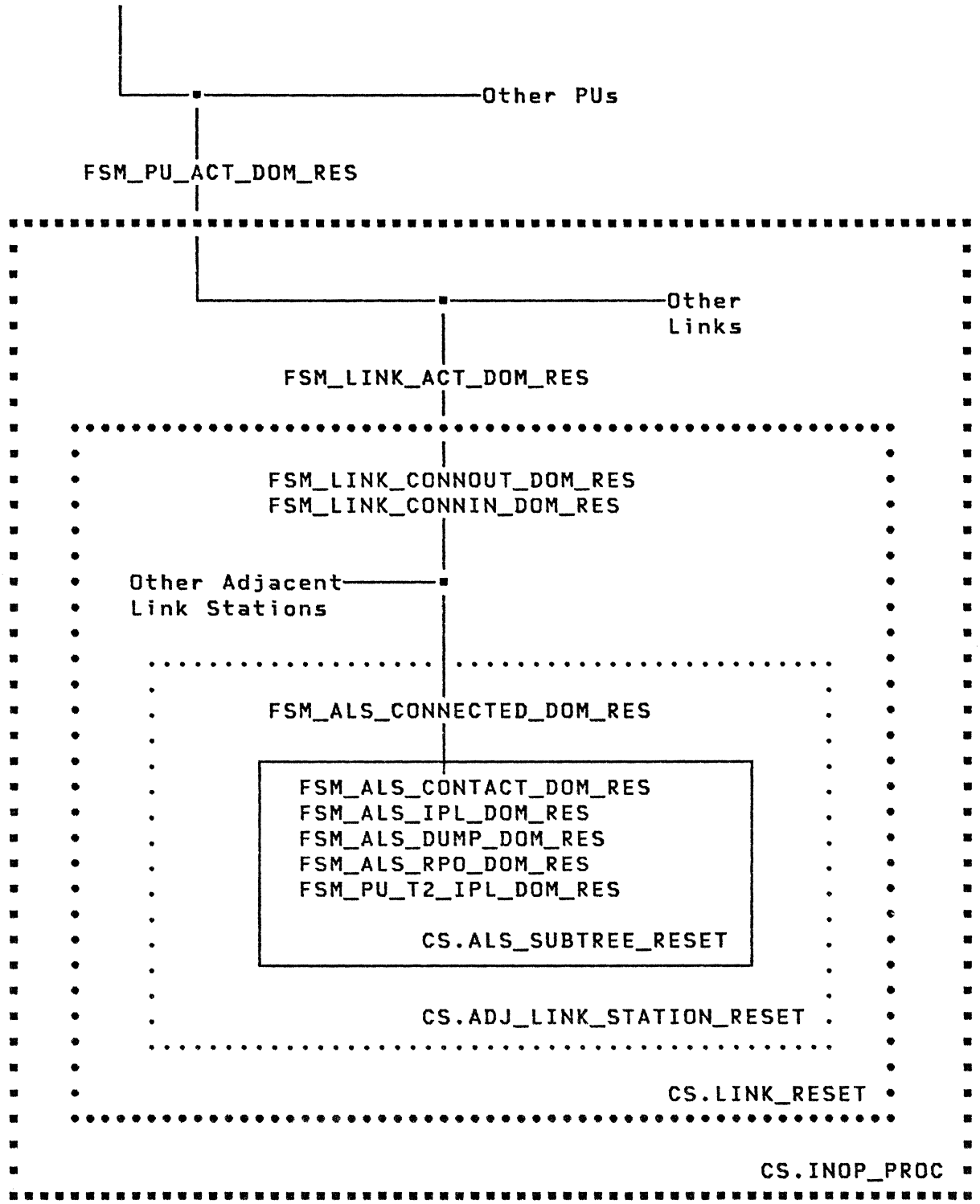


Figure 7-6. The Reset Hierarchy of Domain Resource FSMs in an SSCP

SWITCHED LINK CONNECTION OPERATION

BASIC CONCEPTS

Switched link connection operation involves the activation of a link connection between an SDLC link station in a subarea node and an adjacent SDLC link station in a peripheral node over communication common-carrier switched facilities. Switched link connection allows a station to connect to the network through different switched links at different times.

A call initiating the activation of a switched link connection may originate at either the link station residing in the subarea node or at the adjacent link station residing in the peripheral node. In either case, the call initiation and answering functions may be manual, involving operator assistance, or may be automatic. Switched link connection requires that the subarea PU already have an active session with the SSCP prior to a connection being made with other nodes.

The establishment of a switched link connection requires certain functions to be performed by the PU.SVC_MGR.NS in the subarea PU, and by UPM_TRANSLATION_SVC and the SSCP.SVC_MGR.CS in the SSCP. (The subarea PU and the SSCP may reside in the same node, but are not required to do so.) These functions involve link management, switched link selection and dynamic assignment of network addresses, and network integrity.

Link Management

Link management is a function of the PU.SVC_MGR.NS and includes the following subfunctions:

- Enabling the link connection, upon receipt of a CONNOUT or an ACTCONNIN request from the SSCP.SVC_MGR.CS, so that an outgoing or incoming call is possible

- Placing a call to an adjacent link station, or answering a call that originated at that link station. The telephone number to be dialed is supplied either automatically or manually to the dial equipment. In the event of a manual outgoing call operation, the phone number is provided to the operator by the SSCP.SVC_MGR.CS

- Disabling the link connection from making or answering a call, upon receipt of an ABCONNOUT or a DACTCONNIN request from the SSCP.SVC_MGR.CS

Switched Link Selection and Dynamic Address Assignment

Switched link selection and the dynamic assignment of network addresses is carried out by the SSCP.SVC_MGR.CS and UPM_TRANSLATION_SVC in conjunction with the PU.SVC_MGR.NS. Switched link selection requires knowledge of the physical characteristics of the link connections that are available at each node versus the characteristics required of the link connection in order to contact the specified adjacent link station (characteristics such as line speed, answer capability, and dial capability), and of the types of communication common-carrier services available at each node in the network.

UPM_TRANSLATION_SVC maintains this knowledge of the link stations in the network and of the alternate switched links over which the link stations can connect to the network. Link considerations are transparent to the network operator. Since an LU in a peripheral node can be connected to the network via different links and appear as different network addresses at different times, the SSCP remembers the network address currently in use for a named LU, and provides this name-to-address translation for the network operator.

When UPM_TRANSLATION_SVC receives an activation request from the network operator for an LU in a node connectable to the network via a switched link, the UPM chooses an appropriate switched link to the LU's node (see Figure 7-7). UPM_TRANSLATION_SVC sends to the SSCP.SVC_MGR.CS an ACTLINK and a CONNOUT request.

Later in the call sequence, the SSCP.SVC_MGR.CS issues to the PU.SVC_MGR.NS an RNAA request that carries the local addresses of all the LUs in the node in which the adjacent link station resides. The PU.SVC_MGR.NS responds with the corresponding LU network addresses, which the SSCP.SVC_MGR.CS stores in the domain resource control block entries for the LUs.

Network Integrity

Network integrity requires checking whether a PU that is to be connected via a switched link actually belongs in the network and has been defined to the network. The SSCP.SVC_MGR.CS inspects the XID information field carried in the REQCONT request to see if the PU is part of the domain controlled by the SSCP and is represented by an entry in the domain resource list. Identification is exchanged between the PU and the PU.SVC_MGR.NS via the SDLC XID command and response. In addition to the checking performed by the SSCP, the SSCP.SVC_MGR.CS may provide its SSCP identification in the ACTPU request to allow further integrity checking.

A nonswitched link connection between two link stations in a network can be replaced temporarily by a switched link connection, allowing backup of the nonswitched connection for increased availability. When a link connection that is nonswitched temporarily becomes switched, UPM_TRANSLATION_SVC is responsible for changing the value of the DRCB.SWITCHED_LINK field of the link station and ALS entries to indicate that the link connection is currently switched. UPM_TRANSLATION_SVC changes the field back to the original nonswitched indication when the connection once again becomes nonswitched.

ESTABLISHMENT OF A SWITCHED LINK CONNECTION

Figure 7-8 (page 7-20) illustrates the sequence of RU flows necessary for establishing a switched link connection. Figure 7-9 (page 7-21) shows the configuration services procedures involved during the sequence, and gives a brief summary of the switched link connection operation functions performed by the procedures.

The network operator sends UPM_TRANSLATION_SVC an activation request that carries the name of an LU in the node in which the adjacent link station to be connected resides. UPM_TRANSLATION_SVC issues an ACTLINK request along with a CONNOUT request to the SSCP.SVC_MGR.CS, which processes the requests and forwards them to the PU.SVC_MGR.NS. During the processing of a CONNOUT request, if the CONNOUT RU specifies that the outgoing call operation is to be manual, the SSCP.SVC_MGR.CS sends the operator the phone number of the node to be dialed. This number is maintained in the DRCB.DIAL_DIGITS field of the domain resource list entry for the peripheral PU associated with the LU that is the target of the activation request.

Upon receipt by the PU.SVC_MGR.NS of the CONNOUT request, the outgoing call operation is performed, and the SDLC XID command and response are exchanged between the adjacent PUs.

A switched link connection operation may also be initiated by a peripheral node placing an incoming call to a subarea node. In this case, the switched link over which the call is made has already been activated and the link station in the subarea node has been placed in the enable-answer mode prior to the receipt of the incoming call (i.e., ACTLINK and ACTCONNIN requests have been received and processed by the PU.SVC_MGR.NS). After the incoming call operation is performed and the XID command and response are exchanged, the sequence proceeds exactly like that for an outgoing call.

Following the XID exchange, the PU.SVC_MGR.NS sends a REQCONT request to the SSCP.SVC_MGR.CS. The REQCONT contains the XID information field and indicates the PU type and ID of the peripheral node. The SSCP.SVC_MGR.CS verifies that the information contained in the XID is valid and updates the information contained in the domain resource control block as described below.

Entries for adjacent link stations, PUs, and LUs residing in a node that can be connected to the network via switched links exist in the domain resource list prior to the initiation of the dial sequence; however, the network address field in the peripheral PU and LU entries, and the associated resource pointer field in the peripheral PU entry are not initialized (see Figure 7-7). The network address

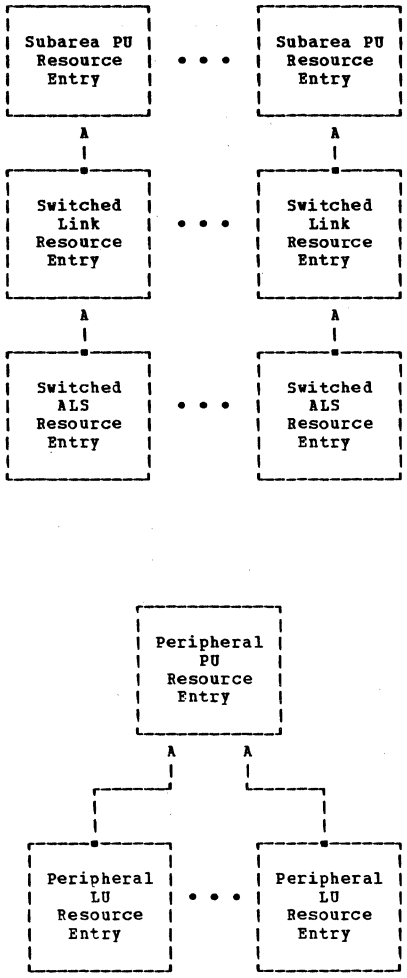
field in the switched ALS entry, however, is initialized at system generation time to be equal to the value of the address of the target link plus one. When the SSCP.SVC_MGR.CS receives a REQCONT, the value of the associated resource pointer field in the peripheral PU entry is set to point to its associated ALS entry. (The ALS entry already points to its associated link, and the entries for the peripheral LUs that are subordinate to the peripheral PU already point to the PU entry.) Also at this time, the network address field in the peripheral PU entry is initialized to be equal to the network address of its associated ALS entry, i.e., the value of the address of the target link plus one. (The SSCP.SVC_MGR.CS generates an RNAA request later in the call sequence to obtain the network addresses of the peripheral LUs, as described below.)

After updating the domain resource list, the SSCP.SVC_MGR.CS generates a SETCV request that informs the PU.SVC_MGR.NS of the PU type of the node being attached and provides information to be used in initializing the boundary function, and a CONTACT request containing the network address of the associated adjacent link station.

The CONTACT request causes the PU.SVC_MGR.NS to exchange the normal SDLC SNRM command and UA response between the adjacent PUs. The successful completion of the link-level contact procedure is reported by the PU.SVC_MGR.NS to the SSCP.SVC_MGR.CS via the CONTACTED request. At this point, the link station in the subarea node and the adjacent link station in the peripheral node have established physical communication on the link.

Next, the SSCP.SVC_MGR.CS sends an ACTPU request to the peripheral PU. The ACTPU may carry the ID of the SSCP. The PU verifies that it has reached the correct SSCP by checking the ID.

When the SSCP.SVC_MGR.CS receives a positive ACTPU response, it generates an RNAA request for all LU addresses associated with the peripheral PU. The RNAA request carries the local addresses of the LUs. The PU.SVC_MGR.NS returns an RNAA response that contains the network addresses of the peripheral LUs. The SSCP.SVC_MGR.CS now initializes the network address field in the domain resource list entries for the LUs. After doing this, it issues a SETCV that contains boundary function pacing count information, and an ACTLU for all LUs associated with the peripheral PU.

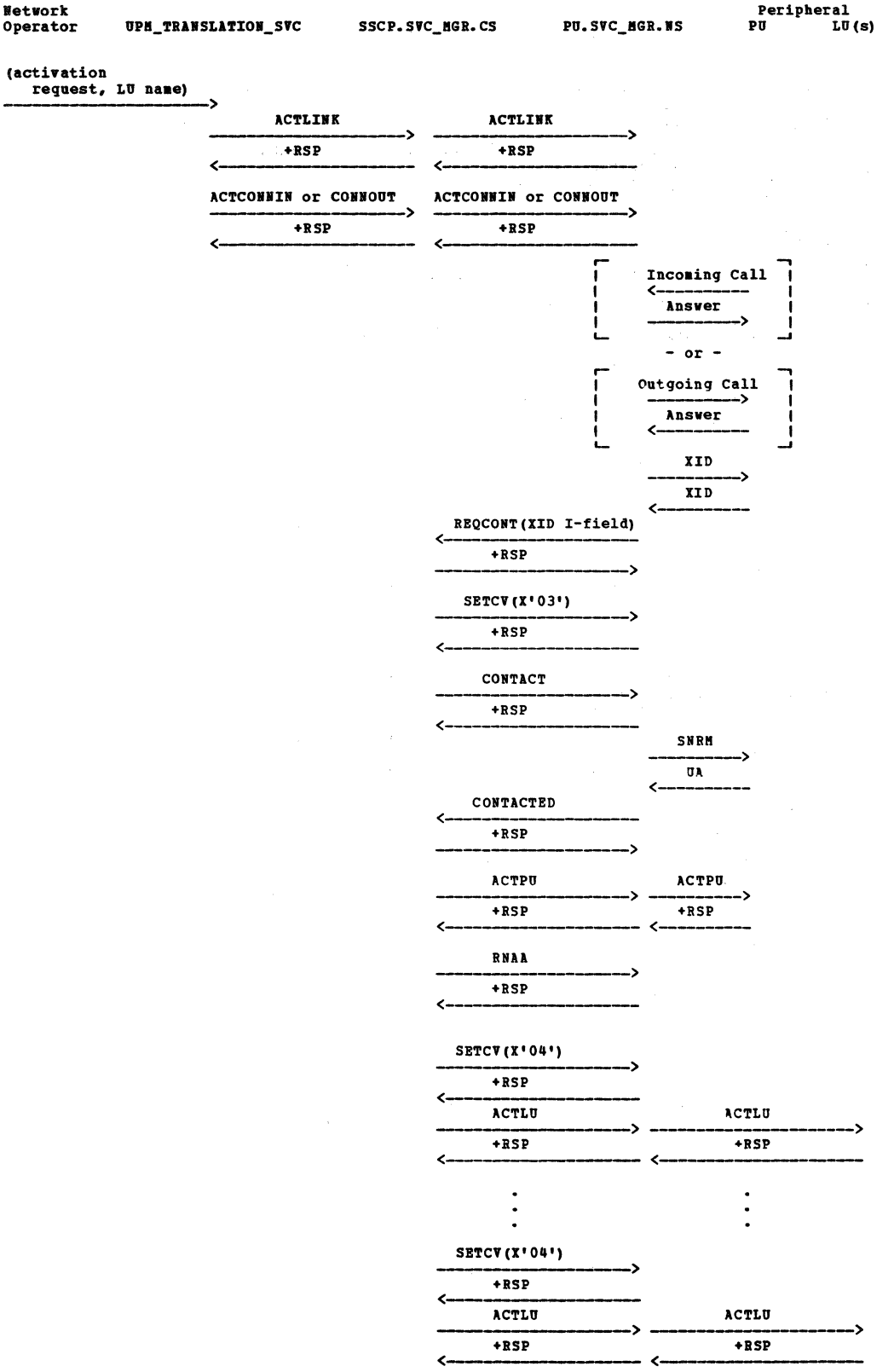


For an outgoing call sequence, the SSCP chooses a specific switched link over which the peripheral PU and LUs will be connected to the network.

For an incoming call sequence initiated at the peripheral node, the switched link selection is made implicitly when the peripheral PU chooses a subarea node to dial into.

Entries for the adjacent link stations and for the peripheral PU and its associated LUs in the figure above exist in the domain resource list prior to the initiation of the switched link connection operation; however, the peripheral PU has not been assigned a network address and the PU domain resource entry does not yet point to an associated adjacent link station entry. This initialization takes place during the switched link connection sequence.

Figure 7-7. Switched Link Selection and the Domain Resource List



Note: The SSCP.SVC_MGR.CS and the PU.SVC_MGR.NS may or may not be in the same node.

Figure 7-8. Establishment of a Switched Link Connection

CONNOUT

When a CONNOUT is received that specifies that a manual dial operation is to be performed, CS.CONN_PROC (page 7-68) calls UPM_MANUAL_DIAL (page 7-126), which sends to the operator the telephone number to be dialed.

REQCONT

When REQCONT is received, CS.REQCONT_REQDISCONT_PROC (page 7-114) checks the XID I-field carried in the REQCONT. If the XID I-field is valid, the procedure:

- initializes the DRCB.ASSOCIATED_RES_PTR and DRCB.NETWORK_ADDRESS fields in the peripheral PU entry
- generates SETCV
- generates CONTACT

If XID I-field is invalid, the procedure generates:

- -RSP(REQCONT)
- DISCONTACT
- ABCONN

CONTACTED

When CS.CONTACTED_PROC (page 7-77) receives a CONTACTED request, the procedure generates:

- +RSP(CONTACTED)
- ACTPU

RSP(ACTPU)

CS.ACTPU_RSP (page 7-54) handles the processing of RSP(ACTPU).

If the response is positive, the procedure generates an RNAA request for all peripheral LU addresses associated with the peripheral PU that received the ACTPU.

If the response is negative, CS.ACTPU_RSP generates DISCONTACT and ABCONN requests.

+RSP(RNAA)

When CS.PERIPHERAL_LU_ADD (page 7-97) receives a +RSP(RNAA), it:

- initializes DRCB.NETWORK_ADDRESS field in the peripheral LU entries
- generates SETCV for all peripheral LUs associated with the peripheral PU
- generates ACTLU for all peripheral LUs associated with the peripheral PU

Figure 7-9. Commentary on Figure 8

COINCIDENCE OF AN OUTGOING CALL AND AN INCOMING CALL

Normally a link station can both send and receive calls, unless it is specifically limited by the available common-carrier facilities to incoming only or outgoing only operations. Collisions between an outgoing call and an incoming call can occur, and may result in various situations depending on the exact timing. Some examples are:

- If an incoming call precedes the connect-out processing, the PU.SVC_MGR.NS reports the unsuccessful dial to the SSCP. In this case, the CONNOUT request was received by the PU.SVC_MGR.NS after the PU had already entered an answer procedure on the same link. This situation occurs only if the link connection was in the enable-answer mode as a result of having received an ACTCONNIN when CONNOUT was issued. The PU.SVC_MGR.NS responds with a negative response to the CONNOUT request with the sense code X'0807' indicating "Resource Not Available."

- If the CONNOUT successfully precedes the incoming call, then neither the SSCP nor the PU.SVC_MGR.NS will be aware that an incoming call came after the outgoing call procedure was initiated at the subarea node. The adjacent link station's call operation is terminated because of a busy signal from the subarea node.

- If the incoming call precedes the actual dialing of the outgoing telephone number (i.e., the telephone at the subarea node goes off-hook for the purpose of dialing the outgoing telephone number and finds the link connection is already active), the XID exchange takes place as normal. If an error has occurred, it is detected by the SSCP when it examines the XID I-field in the REQCONT request.

ERROR-CHECKING AND RECOVERY

If an error occurs on the link during the dial operation, or if the link connection prematurely disconnects after the operation has been completed, the PU.SVC_MGR.NS sends INOP to the SSCP.SVC_MGR.CS, which then processes the INOP and forwards it to UPM_TRANSLATION_SVC. Retrying the link connection operation on another link may involve network operator action.

Figure 7-10 (page 7-24) shows the sequence of events that occurs when the SSCP.SVC_MGR.CS detects an invalid XID I-field in the REQCONT request. The attempted connection is abandoned, but the adjacent link station is informed properly, so that it can release the link connection on its side and become free for any other operation. The SSCP.SVC_MGR.CS sends a DISCONTACT request to the PU.SVC_MGR.NS, which causes the exchange of the SDLC Disconnect command and UA response between the adjacent PUs. The SSCP.SVC_MGR.CS also generates an ABCONN request.

The flow of RUs that result when the peripheral PU detects an invalid SSCP ID in the ACTPU request is shown in Figure 7-11 (page 7-25). The invalid SSCP ID is indicated by a -RSP(ACTPU) with the proper sense code set. The SSCP.SVC_MGR.CS responds, as in the case of an invalid XID I-field, with the DISCONTACT and ABCONN requests.

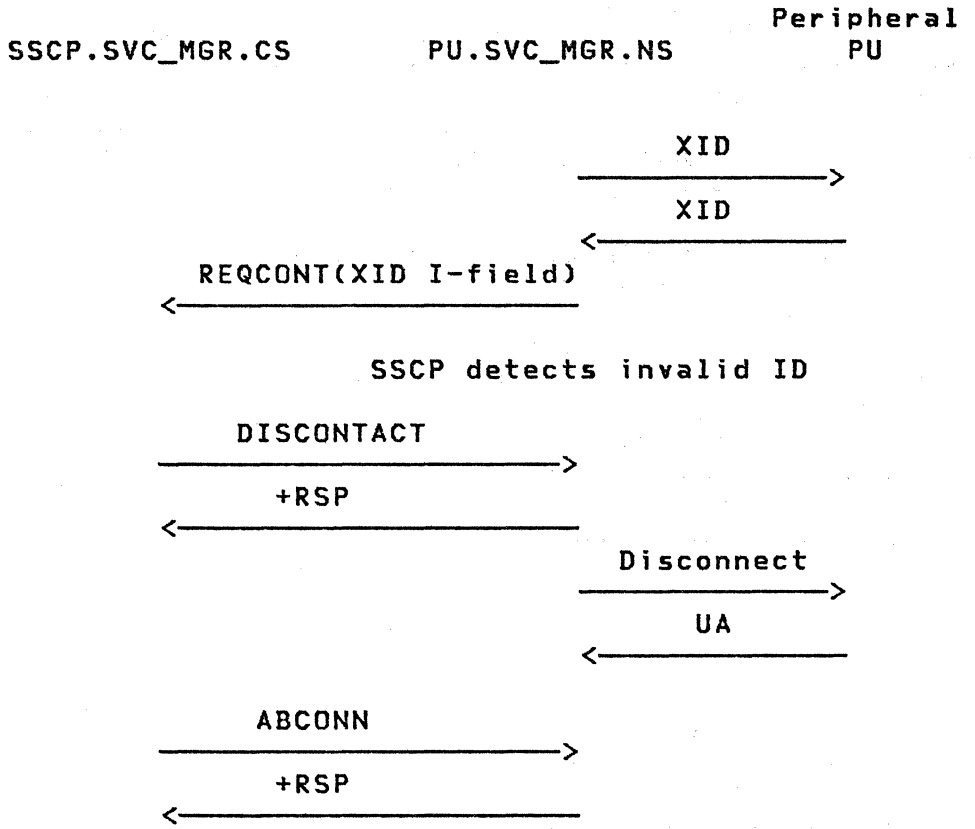


Figure 7-10. SSCP Detects an Invalid XID I-field

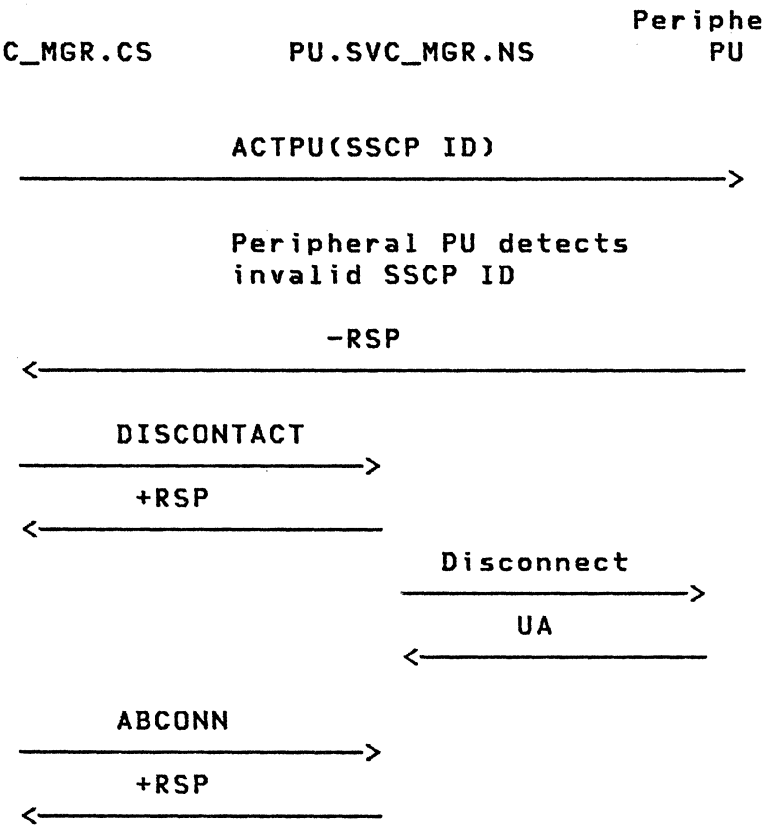


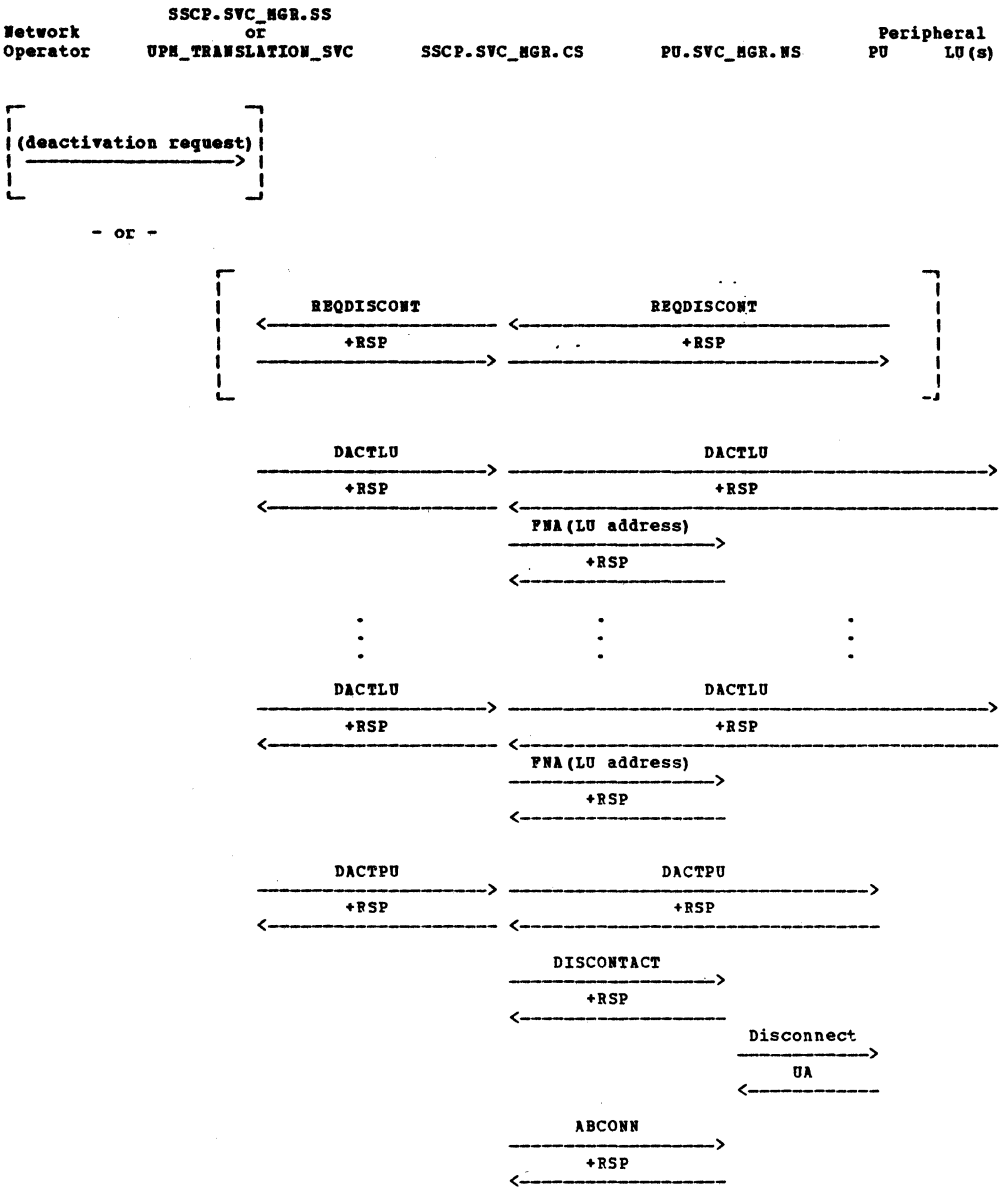
Figure 7-11. Peripheral PU Detects Invalid SSCP ID

DEACTIVATION OF A SWITCHED LINK CONNECTION

A switched link connection deactivation process can be initiated by either the network operator sending to UPM_TRANSLATION_SVC a deactivation request, or by the peripheral PU sending to the SSCP.SVC_MGR.CS a REQDISCONT request. Figure 7-12 (page 7-27) illustrates the sequence of events that takes place when deactivating a switched link connection, while Figure 7-13 (page 7-28) shows the configuration services procedures that are involved in the process.

Sending REQDISCONT from the peripheral PU to the SSCP is optional. If REQDISCONT is sent to the SSCP.SVC_MGR.CS, the request is processed and forwarded to the SSCP.SVC_MGR.SS, which in turn generates multiple DACTLUs and a DACTPU for the LUs and PU in the node to be disconnected, and sends them to the SSCP.SVC_MGR.CS as the active LU-LU sessions become reset. Receipt of the DACTPU causes the call-termination procedure controlled by the SSCP.SVC_MGR.CS to proceed. If REQDISCONT is not sent to the SSCP.SVC_MGR.CS, the DACTLUs and DACTPU are sent to the SSCP.SVC_MGR.CS by UPM_TRANSLATION_SVC. Unless REQDISCONT is received, the decision to terminate the call is made by the network operator.

As the response to each DACTLU is received, the SSCP.SVC_MGR.CS generates an FNA request, and sends the FNA to the PU.SVC_MGR.NS to free the network address associated with the DACTLU's target LU. (Only the address is freed. The LU entry remains in the domain resource list.) Finally, when the response to the DACTPU is received, the SSCP.SVC_MGR.CS sets the network address field in the peripheral PU domain resource entry to 0 and the associated resource pointer field to null, and generates a DISCONTACT followed by an ABCONN, and sends them to the PU.SVC_MGR.NS.



Note: If SSCP.SVC_MGR.CS receives REQDISCONT, it forwards the REQDISCONT TO SSCP.SVC_MGR.SS, which in turn generates the DACTLU(s) and DACTPU. If SSCP.SVC_MGR.CS does not receive REQDISCONT, the DACTLU(s) and DACTPU are issued by UPM_TRANSLATION_SVC as a result of receiving a deactivation request from the network operator.

Figure 7-12. Deactivation of a Switched Link Connection

REQDISCONT

When REQDISCONT is received, CS.REQCONT_REQDISCONT_PROC (page 7-114) sends the REQDISCONT to the SSCP.SVC_MGR.SS (Chapter 8).

+RSP(DACTLU)

When +RSP(DACTLU) is received, CS.LU_RSP (page 7-60) generates an FNA request for the peripheral LU address. The FNA causes the DRCB.NETWORK_ADDRESS field in the LU entry to be set to 0.

+RSP(DACTPU)

When +RSP(DACTPU) is received, CS.DACTPU_RSP (page 7-56):

- sets the DRCB.NETWORK_ADDRESS field in the peripheral PU entry to 0
- sets the DRCB.ASSOCIATED_RES_PTR field in the peripheral PU entry to null
- generates DISCONTACT
- generates ABCONN

Figure 7-13. Commentary on Figure 12

ACTIVATE PHYSICAL UNIT (ACTPU)
DEACTIVATE PHYSICAL UNIT (DACTPU)

Flow: From SSCP|PUCP to PU (Expedited)

Procedure:
CS.PU_PROC (page 7-52)

Principal FSM:
FSM_PU_ACT_DOM_RES (page 7-128)

ACTPU is sent by the SSCP to activate a session with the PU; DACTPU is sent to deactivate a session with the PU. See Chapter 13 for more information on ACTPU and DACTPU.

A PU_T2 node may request to be loaded by setting the Type Activation field in the ACTPU response to indicate IPL required. (See Chapter 11 for a description of the SSCP-PU_T2 and PU_T4|5-PU_T2 load operations.)

ACTIVATE LOGICAL UNIT (ACTLU)
DEACTIVATE LOGICAL UNIT (DACTLU)

Flow: From SSCP to LU (Expedited)

Procedure:
CS.LU_PROC (page 7-58)

Principal FSM:
FSM_LU_ACT_DOM_RES (page 7-128)

ACTLU is sent by the SSCP to activate a session with the LU; DACTLU is sent to deactivate a session with the LU. See Chapter 13 for more information on ACTLU and DACTLU.

Some subarea LUs support parallel sessions. An LU with parallel-session capability has one secondary LU network address and multiple primary LU network addresses associated with it. The primary network addresses are assigned via the RNAA request. While all LUs that do not support parallel sessions are sent ACTLU, an ACTLU request is sent only to the secondary LU address for LUs that do support parallel sessions; the primary LU addresses become active when the secondary LU address does.

ACTIVATE LINK (ACTLINK)
DEACTIVATE LINK (DACTLINK)

Flow: From SSCP to PU_T4|5 or PUCP to PU (Normal)

Procedure:
CS.LINK_PROC (page 7-62)

Principal FSM:
FSM_LINK_ACT_DOM_RES (page 7-129)

ACTLINK initiates a procedure at the PU to activate the protocol boundary between a link station in the node (as specified by the link network address parameter in the request) and the link connection attached to it. Adjacent link stations may be contacted only after a positive response to ACTLINK has been received.

DACTLINK initiates a procedure at the PU to deactivate the protocol boundary between a link station in the node (as specified by the link network address parameter in the request) and the link connection attached to it. It is used after all adjacent link stations on the specified link have been discontacted (see CONTACT, CONTACTED, DISCONTACT, later in this section).

ACTIVATE CONNECT IN (ACTCONNIN)
DEACTIVATE CONNECT IN (DACTCONNIN)

Flow: From SSCP to PU_T4|5 or PUCP to PU (Normal)

Procedure:
CS.CONN_PROC (page 7-68)

Principal FSMs:
FSM_LINK_CONNIN_DOM_RES (page 7-129)
FSM_ALS_CONNECTED_DOM_RES (page 7-133)

ACTCONNIN requests the PU to enable the specified link to accept incoming calls.

DACTCONNIN requests the PU to disable the specified link from accepting incoming calls.

These requests control the incoming-connection answering ability of the link (as reflected in the state of FSM_LINK_CONNIN_DOM_RES) independent of the connection status of the link (as reflected in the state of FSM_ALS_CONNECTED_DOM_RES). This means:

- The connect-in ability may be enabled or disabled while the link connection is active, without any effect on the link connection
- A link connection may be activated (via a connect-out) and/or deactivated without any effect on the connect-in ability

Neither ACTCONNIN nor DACTCONNIN affects an active link connection.

CONNECT OUT (CONNOUT)
ABANDON CONNECT OUT (ABCONNOUT)

Flow: From SSCP to PU_T4|5 or PUCP to PU (Normal)

Procedure:

CS.CONN_PROC	(page 7-68)

Principal FSMs:

FSM_LINK_CONNOUT_DOM_RES	(page 7-130)
FSM_ALS_CONNECTED_DOM_RES	(page 7-133)

CONNOUT requests the PU to initiate a connect-out procedure on the specified link. Included in the request parameters, if a switched-network calling operation is to occur, are the telephone number (in EBCDIC digits and separator characters) and the number of times the calling operation is to be retried. CONNOUT is also used to initiate a connect-out procedure for an X.21 connection, as described in CCITT (Consultative Committee on International Telegraph and Telephone) recommendation X.21. For an X.21 connection with direct call feature, the dial digits are not provided.

ABCONNOUT requests the PU to terminate a connect-out procedure on the designated link.

Neither CONNOUT nor ABCONNOUT affects an active link connection.

REQUEST CONTACT (REQCONT)
ABANDON CONNECTION (ABCONN)

Flow: From PU_T4|5 to SSCP or PU to PUCP (Normal)
for REQCONT;
from SSCP to PU_T4|5 or PUCP to PU (Normal)
for ABCONN

Procedures:

CS.CONN_PROC	(page 7-68)
CS.REQCONT_REQDISCONT_PROC	(page 7-114)

Principal FSM:

FSM_ALS_CONNECTED_DOM_RES	(page 7-133)

REQCONT notifies the SSCP that a connection with an adjacent secondary link station (in a PU_T1|2 node) has been activated via a successful connect-in or connect-out procedure. A DLC-level identification exchange (XID) is required before issuing REQCONT; the XID information field of the adjacent link station is sent as a parameter of REQCONT (see Appendix E).

ABCONN requests the PU to deactivate the link connection for the specified link.

CONTACT
CONTACTED
DISCONTACT

Flow: From SSCP to PU_T4|5 or PUCP to PU (Normal) for CONTACT and DISCONTACT;
from PU_T4|5 to SSCP or PU to PUCP (Normal) for CONTACTED

Procedures:

Principal FSM:

CONTACT requests the initiation of a procedure at the PU to activate DLC-level contact with the adjacent link station specified in the request. The DLC-level contact must be activated before any PIUs can be exchanged with the adjacent node over the link. The contact procedure is DLC-dependent (for example, it may cause an SDLC SNRM command, or UA in response to SNRM, to be issued by the link station). A positive response to CONTACT is issued when the DLC contact procedure begins.

CONTACTED is issued by the PU to indicate to the SSCP the completion of the DLC contact procedure. A status parameter conveyed by this request informs SSCP configuration services whether or not the contact procedure was successful; if not successful, the status indicates, for example, whether an adjacent PU node load is required or whether an error occurred on the contact procedure.

DISCONTACT requests the PU to deactivate DLC-level contact with the specified adjacent node. The discontact procedure is DLC-dependent; if applicable, polling is stopped. DISCONTACT may be used to terminate contact, IPL, or dump procedures before their completion. The PU responds negatively to DISCONTACT if an uninterruptible link-level procedure is in progress at the primary link station of the specified link.

REQUEST DISCONTACT (REQDISCONT)

Flow: From PU_T1|2 to SSCP (Normal)

Procedure:
CS.REQCONT_REQDISCONT_PROC (page 7-114)

Principal FSM: None

With REQDISCONT, the PU_T1|2 requests the SSCP to start a procedure that will ultimately discontact the secondary link station in the PU_T1|2 node.

If the Type parameter on this request indicates "normal," the requested procedure should start when all the sessions involving LUs that are local to the PU_T1|2 node are reset by LU-invoked session deactivation. The requested procedure involves sending DACTLU to each active LU in the PU_T1|2 node, DACTPU to the PU_T1|2, DISCONTACT to the PU_T4|5, and, if the PU_T1|2 is connected via a switched link, ABCONN to the PU_T4|5.

If the Type parameter on this request indicates "immediate," the requested procedure should cause CTERM(Forced) to be sent to all PLUs having an active session with an LU local to the PU_T1|2 node. Each such PLU will send UNBIND (optionally preceded by CLEAR) to its SLU session partner. When the SSCP receives notification that all these sessions are reset, DACTLU is sent to all active LUs in the PU_T1|2 node, DACTPU to the PU_T1|2, DISCONTACT to the PU_T4|5, and, for switched link connections, ABCONN to the PU_T4|5.

For nonswitched link connections, if the CONTACT information field on this request indicates "send CONTACT immediately," CONTACT is sent to the PU_T4|5 after the DISCONTACT as part of the requested procedure. Otherwise, the requested procedure is completed when the DISCONTACT is sent. Sending the CONTACT allows the PU_T1|2 node to resume an active role in the network (i.e., activate DLC-level contact and receive and send PIUs) when it is ready.

For switched link connections, the CONTACT information field is reserved; the requested procedure ends with the ABCONN.

IPL INITIAL (IPLINIT)
IPL TEXT (IPLTEXT)
IPL FINAL (IPLFINAL)

Flow: From SSCP to PU_T4|5 (Normal)

Procedure:
CS.LOAD_PROC (page 7-78)

Principal FSM:
FSM_ALS_IPL_DOM_RES (page 7-131)

IPLINIT initiates a DLC-level load of an adjacent PU_T4 node from the PU_T4|5 node. The node to be loaded is identified by the adjacent link station address contained in the request. IPLINIT resets the IPL, dump, and contact FSMs.

Following an IPLINIT, any number of IPLTEXT commands are valid. IPLTEXT transfers load module information to the PU_T4|5, which passes it in a DLC-level load to the PU_T4 node.

IPLFINAL completes an IPL sequence and supplies the load module entry point to the PU_T4 node. A positive response to IPLFINAL indicates that the PU_T4 node is successfully loaded.

DUMP INITIAL (DUMPINIT)
DUMP TEXT (DUMPTEXT)
DUMP FINAL (DUMPFINAL)

Flow: From SSCP to PU_T4|5 (Normal)

Procedure:
CS.DUMP_PROC (page 7-80)

Principal FSM:
FSM_ALS_DUMP_DOM_RES (page 7-131)

DUMPINIT requests the PU_T4|5 to initiate a DLC-level dump from an adjacent PU_T4 node to the PU_T4|5, for eventual transmission to the SSCP. The node to be dumped is identified by the adjacent link station address contained in the request. Basic dump data, such as register, key, and indicator values, may be returned on the response to this request. DUMPINIT resets the IPL, dump, and contact FSMs.

If further dump data is required, DUMPINIT may be followed by DUMPTEXT. DUMPTEXT causes the dump data specified by the starting-address parameter to be returned to the SSCP on the response. The PU_T4|5 obtains the dump data from the PU_T4 node, using a DLC-level interchange.

DUMPFINAL terminates the dump sequence, whether DUMPTEXT is used or not. A positive response to DUMPFINAL indicates that the dump sequence is complete.

REMOTE POWER OFF (RPO)

Flow: From SSCP to PU_T4|5 (Normal)

Procedure:
CS.RPO_PROC (page 7-83)

Principal FSM:
FSM_RPO_DOM_RES (page 7-132)

RPO causes the receiving PU_T4|5 to initiate a DLC-level power-off sequence to the PU_T4 node specified by the adjacent link station network address conveyed in the request. The PU_T4 node being powered off does not need to have an active SSCP-PU half-session nor be contacted. RPO resets the IPL, dump, and contact FSMs.

INOPERATIVE (INOP)

Flow: From PU_T4|5 to SSCP or PU to PUCP (Normal)

Procedure:
CS.INOP_PROC (page 7-110)

Principal FSM: None

INOP is sent to the SSCP by the PU to report a link-related connection or contact failure involving one or more PU nodes. The target of the INOP is a link or an adjacent link station. The SSCP processes this request by resetting the FSMs within the appropriate link or adjacent link station subtree.

For specific types of INOP requests and their corresponding reason codes, see Appendix E.

LOAD REQUIRED (LDREQD)

Flow: From PU_T2 to SSCP (Normal)

Procedure:
CS.LDREQD_PROC (page 7-86)

Principal FSM: None

The LDREQD request enables the PU_T2 to request a specific load module be moved to its node. Upon receipt of LDREQD, the SSCP inspects the Adjacent PU Load Capability bit. If the bit is set to CAPABLE, the SSCP sends INITPROC to the subarea PU adjacent to the PU_T2 to initiate a PU_T4|5-PU_T2 load operation. If the bit is set to NOT_CAPABLE and the SSCP can load the PU_T2 node, the SSCP sends NS_IPL_INIT to the PU_T2 to begin the load operation. If the bit is set to NOT_CAPABLE and the SSCP cannot load the PU_T2 node, the SSCP sends NS_IPL_ABORT to the PU_T2. See Chapter 11 for a description of the SSCP-PU_T2 and PU_T4|5-PU_T2 load operations.

INITIATE PROCEDURE (INITPROC)
PROCEDURE STATUS (PROCSTAT)

Flow: From SSCP to PU_T4|5 (Normal) for INITPROC;
from PU_T4|5 to SSCP (Normal) for PROCSTAT

Procedures:
CS.INITPROC_PROC (page 7-87)
CS.PROCSTAT_PROC (page 7-89)

Principal FSM:
FSM_PROC_DOM_RES (page 7-132)

INITPROC is sent to the subarea PU adjacent to a PU_T2 in order to initiate a PU_T4|5-PU_T2 load operation. (See Chapter 11 for a description a PU_T4|5-PU_T2 load operation.) INITPROC is sent by the SSCP upon receipt of LDREQD(Adjacent PU Load Capability = CAPABLE) or of +RSP(ACTPU, IPL Required, Adjacent PU Load Capability = CAPABLE), indicating the subarea PU can load the PU_T2 node. If the SSCP receives a negative response to INITPROC, the SSCP tries to perform an SSCP-PU_T2 load operation just as if it received LDREQD(Adjacent PU Load Capability = NOT_CAPABLE) or +RSP(ACTPU, IPL Required, Adjacent PU Load Capability = NOT_CAPABLE), indicating the PU_T4|5 cannot load the PU_T2 node. See Chapter 11 for a description of an SSCP-PU_T2 load operation.

PROCSTAT reports to the SSCP either the successful completion or the failure of the load operation. If the procedure failed, the request code of the failing RU and sense data are included as parameters in the PROCSTAT RU. If a negative response from the PU_T2 was the cause of the failure, the sense data from the negative response is placed in the PROCSTAT. If the PROCSTAT indicates the PU_T2 node has not been loaded, and the load operation was requested via the response to ACTPU, the SSCP sends DACTPU to the PU_T2. The PU_T2 may request another load or may send REQDISCONT if the load operation was requested via LDREQD.

NETWORK SERVICES IPL INITIAL (NS_IPL_INIT)
NETWORK SERVICES IPL TEXT (NS_IPL_TEXT)
NETWORK SERVICES IPL FINAL (NS_IPL_FINAL)
NETWORK SERVICES IPL ABORT (NS_IPL_ABORT)

Flow: From SSCP to PU_T2 (Normal)

Procedure:
CS.PU_T2_LOAD_RSP (page 7-92)

Principal FSM:
FSM_PU_T2_IPL_DOM_RES (page 7-133)

The NS_IPL_INIT request is sent from the SSCP to the PU_T2 to indicate that a particular load module is about to be transmitted to the PU_T2's node. Upon receipt of the +RSP(NS_IPL_INIT), the SSCP starts transmitting the load module by sending NS_IPL_TEXT. When the SSCP receives the response to NS_IPL_TEXT, it may send another NS_IPL_TEXT. Any number of NS_IPL_TEXT requests may be sent depending on the size of the load module.

When the SSCP receives the response to the final NS_IPL_TEXT, it sends NS_IPL_FINAL indicating the load module transfer has been completed. The NS_IPL_FINAL contains the entry-point location for the PU_T2 node to begin execution of the load module.

If, at any time during the load operation, the SSCP receives a negative response, or if the load operation cannot be completed, the SSCP sends NS_IPL_ABORT to the PU_T2. The NS_IPL_ABORT indicates to the PU_T2 that the load operation has been halted. Sense data is included in NS_IPL_ABORT indicating the cause of the failure. If the NS_IPL_ABORT is the result of a negative response from the PU_T2, the sense data in the response may be placed in the sense data of the NS_IPL_ABORT.

ASSIGN NETWORK ADDRESSES (ANA)

Flow: From SSCP to PU_T4|5 (Normal)

Procedure:
UPM_ANA_PROC (page 7-123)

Principal FSM: None

ANA updates the path control routing algorithm in the PU_T4|5 node, such that PIUs with the specified LU network addresses (one or more) will be routed to the specified PU_T1|2 node.

REQUEST NETWORK ADDRESS ASSIGNMENT (RNAA)

Flow: From SSCP to PU_T4|5 (Normal)

Procedure:
CS.RNAA_PROC (page 7-94)

Principal FSM: None

RNAA requests the PU to update its path control routing table and to assign one or more network addresses:

- To one or more adjacent link stations and their BF.PUs, as identified in the RNAA request by a link network address and secondary link station link-level addresses
- To one or more BF.LUs, where the BF.LUs are identified in the RNAA request by an adjacent link station network address and the LU local addresses
- To an LU that supports parallel sessions, where the LU is identified in the RNAA request by the LU network address used for the SSCP-LU session, in order to assign an additional network address

The PU returns the network addresses in the RNAA response.

FREE NETWORK ADDRESSES (FNA)

Flow: From SSCP to PU_T4|5 (Normal)

Procedure:
CS.FNA_PROC (page 7-99)

Principal FSM: None

FNA is sent from an SSCP to request the PU_T4|5 to remove the appropriate entries from the node resource list, thereby freeing the network addresses associated with the corresponding resources in the node. Bytes 3 and 4 of the FNA RU contain the network address associated with the target resource, i.e., the PU_T4|5, LU, link, or BF.PU.

Target resource	Resources to free
PU	LUs identified by network addresses associated with SSCP-LU sessions
LU (identified by the network address associated with an SSCP-LU session)	LU network addresses used as primary network addresses in parallel sessions
Link	BF.PUs and adjacent link stations
BF.PU	BF.LUs

The FNA RU contains the number of network addresses to be freed, followed by the actual network addresses to be removed from the node resource list. If the number to be freed is zero, then no network addresses are present in the FNA RU, and all the node resources associated with the target resource, as indicated in the table above, are to be removed.

The target resource address may be zero. This means that the target resource network address is to be determined by the PU.SVC_MGR receiving the FNA by analyzing the first network address in the list of addresses to be freed. If the network address is for:

- An LU, and the network address is used for an SSCP-LU session, then the target resource is the PU_T4|5
- An LU, and the network address is not used for an SSCP-LU session, then the target resource is the LU network address associated with the SSCP-LU session
- A BF.PU, then the target resource is the link attaching

the node of the PU_T1|2 (represented by the BF.PU) to the BF.PU's node

- A BF.LU, then the target resource is the BF.PU that supports the BF.LU

Upon receiving the positive response from the PU_T4|5, the SSCP removes the resources from the corresponding list maintained in the SSCP.

ADD LINK (ADDLINK)
ADD LINK STATION (ADDLINKSTA)
DELETE NETWORK RESOURCE (DELETENR)

Flow: From SSCP to PU_T4|5 (Normal)

Procedures:

CS.ADDLINK_ADDLINKSTA_PROC	(page 7-106)
CS.DELETENR_PROC	(page 7-108)

Principal FSM: None

ADDLINK is sent from the SSCP to the PU to obtain a link network address that will be mapped to the locally-used link identifier specified in the request. A positive response to ADDLINK will contain a link network address.

ADDLINKSTA is sent from the SSCP to the PU to obtain an adjacent link station network address to be associated with the locally-used link station identifier specified in the request. An additional qualifier is included in this RU to notify the PU of the FID types that may be sent or received by this adjacent link station. A positive response to ADDLINKSTA will contain the requested adjacent link station network address.

DELETENR is sent to free a network address assigned to a link or adjacent link station.

ENTERING SLOWDOWN (ESLOW).
EXITING SLOWDOWN (EXSLOW)

Flow: From PU_T4 to SSCP (Normal)

Procedure:
UPM_SLOW_PROC (page 7-123)

Principal FSM: None

ESLOW informs the SSCP that the node of the sending PU has entered a slowdown state. This state is generally associated with buffer depletion, and requires traffic through the node to be reduced or suspended.

EXSLOW informs the SSCP that the node of the sending PU is no longer in the slowdown state and regular traffic can resume.

REQUEST FREE NETWORK ADDRESSES (REQFNA)

Flow: From PU_T4|5 to SSCP (Normal)

Procedure:
CS.RCV (page 7-50)

Principal FSM: None

REQFNA is sent from a PU_T4|5 to an SSCP to request the SSCP to send FNA to the PU_T4|5 in order to free all addresses for the specified LU. The REQFNA contains a type-of-deactivation field; there are four types of deactivation--Normal, Orderly, Forced, and Cleanup. For each of these deactivation types, the SSCP prevents new sessions from being activated with the LU to be freed and waits until all existing LU-to-LU sessions with the LU have been deactivated; the SSCP then sends DACTLU to the LU followed by FNA to the PU. For the normal type of deactivation, the SSCP does not take any action to cause LU-to-LU session deactivation. For Orderly or Forced types of deactivation, a CTERM Orderly or Forced, respectively, is sent to the PLU for each session to be deactivated; for the Cleanup type of deactivation, CLEANUP is sent to the LU for each session to be deactivated.

REQUEST ACTIVATE LOGICAL UNIT (REQACTLU)

Flow: From PU_T4|5 to SSCP (Normal)

Procedure:
CS.RCV (page 7-50)

Principal FSM: None

REQACTLU is sent from the PU to an SSCP to request that ACTLU be sent to the LU named in the RU. The parameters on the REQACTLU contain the network address and network name of the LU. The LU to be sent ACTLU resides in the PU_T4|5 node; definition of the LU to the PU_T4|5 and the SSCP is implementation- and installation-dependent. When an LU is to be added to a local domain, for local access only, then the SSCP may choose to add the LU without prior definition of the LU name.

When the SSCP receives the REQACTLU from the PU, the LU network name is used to obtain the capabilities of the LU as defined by the implementation- and installation-dependent process. The capabilities of the LU may be modified by the RSP(ACTLU) or, if the LU name is for local access only, the capabilities of the LU are defined at RSP(ACTLU). If the network name of the LU is not recognized by the SSCP (when the SSCP does not choose to add LUs without prior definition), the SSCP lacks resources to support the LU, or the LU is presently active, the SSCP responds with a negative response--Resource Unknown (X'0806'), Insufficient Resources (X'0812'), or Function Active (X'0815'). If the SSCP responds positively, then the SSCP activates the LU by sending ACTLU.

NETWORK SERVICES LOST SUBAREA (NS_LSA)

Flow: From PU_T4|5 to SSCP (Normal)

Procedure:
UPM_NS_LSA_PROC (page 7-124)

Principal FSM: None

NS_LSA is sent by the PU to every SSCP with which it has an active session to report the interruption of routing capability to a set of subareas after originating or propagating a LOST SUBAREA (LSA).

The list of subareas in the NS_LSA request is identical to the list sent by the PU in the LSA request.

SET CONTROL VECTOR (SETCV)

Flow: From SSCP to PU_T4|5 (Normal)

Procedure:
CS.SEND (page 7-48)

Principal FSM: None

SETCV sets a control vector that is maintained by the PU receiving the request and that is associated with the network address specified in the RU.

For SETCV(Intensive Mode), see Chapter 9.

START DATA TRAFFIC (SDT)

Flow: From SSCP to PU_T4|5 (Expedited)

Procedure:
CS.RCV (page 7-50)

Principal FSM: None

SDT is sent by the primary session control to the secondary session control to enable the sending and the receiving of FMD and DFC requests and responses by both half-sessions.

EXPLICIT ROUTE INOPERATIVE (ER_INOP)
VIRTUAL ROUTE INOPERATIVE (VR_INOP)

Flow: From PU_T4|5 to SSCP or PU_T4 to PUCP (Normal)

Procedure:
UPM_ER_VR_INOP_PROC (page 7-127)

Principal FSM: None

ER_INOP and VR_INOP notify the SSCP when an explicit route or a virtual route has become inoperative as the result of a transmission group having become inoperative somewhere in the network. The SSCP displays this information for the network operator.

LOST CONTROL POINT (LCP)

Flow: From PU_T4|5 to SSCP or PU_T4 to PUCP (Normal)

Procedure:
UPM_LCP_PROC (page 7-127)

Principal FSM: None

LCP notifies the SSCP that a subarea PU's session with another SSCP has failed. The SSCP displays this information for the network operator.

```
SSCP.SVC_MGR.CS.SEND: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE  RECEIVES ALL  INPUT TO  THE SSCP.SVC_MGR.CS  THAT IS │
│                 SENT BY UPM_TRANSLATION_SVC (CHAPTER 6), AND ROUTES THE INPUT TO THE │
│                 APPROPRIATE PROCEDURE FOR PROCESSING.                    │
│                                                                          │
│      INPUT:     THE  CURRENT MESSAGE  UNIT,  INCLUDING  THE ORIGIN  AND  DESTINATION │
│                 ADDRESSES IN THE TH FIELDS                               │
│                                                                          │
│      OUTPUT:    REFER TO THE PROCEDURES THAT ARE  CALLED FROM THIS PROCEDURE FOR THE │
│                 SPECIFIC OUTPUTS.                                        │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 CS.ADDLINK_ADDLINKSTA_PROC          PAGE 7-106           │
│                 CS.CONN_PROC                        PAGE 7-68            │
│                 CS.CONTACT_PROC                     PAGE 7-72            │
│                 CS.DELETENR_PROC                    PAGE 7-108           │
│                 CS.DISCONTACT_PROC                  PAGE 7-74            │
│                 CS.DUMP_PROC                        PAGE 7-80            │
│                 CS.FNA_PROC                         PAGE 7-99            │
│                 CS.INITPROC_PROC                    PAGE 7-87            │
│                 CS.LINK_PROC                        PAGE 7-62            │
│                 CS.LOAD_PROC                        PAGE 7-78            │
│                 CS.LU_PROC                          PAGE 7-58            │
│                 CS.PU_PROC                          PAGE 7-52            │
│                 CS.RNAA_PROC                        PAGE 7-94            │
│                 CS.RPO_PROC                         PAGE 7-83            │
│                 FSM_PROC_DOM_RES                    PAGE 7-132           │
│                 UPM_ANA_PROC                        PAGE 7-123           │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL TARGET_NA BIT(48);
  DCL REQUEST_CODE BIT(8);

  IF SERVICE_TYPE = NETWORK_SERVICES THEN
    DO;
    . IF NS_RQ_CODE = (REQDISCONT | LDREQD | NS_LSA | LCP | ER_INOP | VR_INOP) THEN
    .   TARGET_NA = DSAF||DEF;
    . ELSE
    .   TARGET_NA = DSAF||(NSC_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
    .                                                   /* APPENDIX A      */
    . REQUEST_CODE = NS_RQ_CODE;
    .
    . FIND SCB IN SCB_LIST
    .      WHERE(SCB.PARTNER_SA = DSAF &
    .      SCB.PARTNER_EA = 0 &
    .      SCB.THIS_SA = NCB.NODE_SUBAREA_ADDRESS &
    .      SCB.THIS_EA = NCB.SSCP_ELEMENT_ADDRESS);
    END;

  ELSE
    DO;
    . TARGET_NA = DSAF||DEF;
    . REQUEST_CODE = RQ_CODE;
    END;
```

```
IF RRI = RQ THEN
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │          INPUT IS A REQUEST.         │
                  └──────────────────────────────────────┘
                                                                        */
  IF FIND_DOMAIN_RESOURCE(TARGET_NA) = NULL THEN          /* APPENDIX B   */
    SEND SEND_CHECK(X'0806') TO UPM_TRANSLATION_SVC;      /* RESOURCE UNKNOWN */

  ELSE

    SELECT ANYORDER;
    .
    . WHEN(RU_CTGY = SC & REQUEST_CODE = (ACTPU | DACTPU))
    .   CALL CS.PU_PROC;                                  /* PAGE 7-52    */
    .
    . WHEN(RU_CTGY = SC & REQUEST_CODE = (ACTLU | DACTLU))
    .   CALL CS.LU_PROC;                                  /* PAGE 7-58    */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = (ACTLINK | DACTLINK))
    .   CALL CS.LINK_PROC;                                /* PAGE 7-62    */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = (ADDLINK | ADDLINKSTA))
    .   CALL CS.ADDLINK_ADDLINKSTA_PROC;                  /* PAGE 7-106   */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = DELETENR)
    .   CALL CS.DELETENR_PROC;                            /* PAGE 7-108   */
    .
    . WHEN(RU_CTGY = FMD &
    .      REQUEST_CODE = (ACTCONNIN | DACTCONNIN | CONNOUT | ABCONNOUT | ABCONN))
    .   CALL CS.CONN_PROC;                                /* PAGE 7-68    */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = CONTACT)
    .   CALL CS.CONTACT_PROC;                             /* PAGE 7-72    */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = DISCONTACT)
    .   CALL CS.DISCONTACT_PROC;                          /* PAGE 7-74    */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = ANA)
    .   CALL UPM_ANA_PROC;                                /* PAGE 7-123   */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = RNAA)
    .   CALL CS.RNAA_PROC;                                /* PAGE 7-94    */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = FNA)
    .   CALL CS.FNA_PROC;                                 /* PAGE 7-99    */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = (DUMPINIT | DUMPTEXT | DUMPFINAL))
    .   CALL CS.DUMP_PROC;                                /* PAGE 7-80    */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = (IPLINIT | IPLTEXT | IPLFINAL))
    .   CALL CS.LOAD_PROC;                                /* PAGE 7-78    */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = INITPROC)
    .   CALL CS.INITPROC_PROC;                            /* PAGE 7-87    */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = RPO)
    .   CALL CS.RPO_PROC;                                 /* PAGE 7-83    */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = SETCV)
    .   SEND MU TO SNS.SEND;                              /* CHAPTER 6    */
    .
    . OTHERWISE
    .   SEND SEND_CHECK(X'1003') TO UPM_TRANSLATION_SVC;  /* FUNCTION NOT SUPPORTED */
    .
    END;

  ELSE
    IF RRI = RSP THEN
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │          INPUT IS A RESPONSE.        │
                  │                                      │
                  │  (RESPONSE TO INOP, REQDISCONT, REQFNA, ESLOW,  │
                  │  EXSLOW,   PROCSTAT,  LDREQD,   NS_LSA,   LCP,  │
                  │  ER_INOP, VR_INOP, OR REQACTLU.)     │
                  └──────────────────────────────────────┘
                                                                        */
      DO;
      . IF RU_CTGY = FMD & REQUEST_CODE = PROCSTAT THEN
      .   DO;
      .   . DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);   /* APPENDIX B   */
      .   . IF DRCB.RESOURCE_CATEGORY = ALS THEN
      .   .   DRCB_PTR = FIND_SUBORDINATE_DOM_RES(TARGET_NA); /* APPENDIX B */
      .   . CALL FSM_PROC_DOM_RES;                        /* PAGE 7-132   */
      .   END;
      .
      . SEND MU TO SNS.SEND;                              /* CHAPTER 6    */
      END;

END SSCP.SVC_MGR.CS.SEND;
```

```
SSCP.SVC_MGR.CS.RCV: PROCEDURE;

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE  RECEIVES ALL  INPUT TO  THE SSCP.SVC_MGR.CS  THAT IS │
 │                 SENT BY SNS  (CHAPTER 6), BY PU.SVC_MGR.CSC_MGR (CHAPTER  13), OR BY │
 │                 TC.SC (CHAPTER 4), AND ROUTES THE INPUT TO THE APPROPRIATE PROCEDURE │
 │                 FOR PROCESSING.                                              │
 │                                                                              │
 │      INPUT:     THE CURRENT MESSAGE  UNIT; THE SCB POINTER HAS ALREADY  BEEN SET AND │
 │                 IS PASSED TO THIS PROCEDURE AS PART OF THE ENVIRONMENT       │
 │                                                                              │
 │      OUTPUT:    REFER TO THE PROCEDURES THAT ARE  CALLED FROM THIS PROCEDURE FOR THE │
 │                 SPECIFIC OUTPUTS.                                            │
 │                                                                              │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
 │                 CS.ACTPU_RSP                        PAGE 7-54                │
 │                 CS.ADDLINK_ADDLINKSTA_RSP           PAGE 7-107               │
 │                 CS.CONN_RSP                         PAGE 7-70                │
 │                 CS.CONTACT_DISCONTACT_RSP           PAGE 7-76                │
 │                 CS.CONTACTED_PROC                   PAGE 7-77                │
 │                 CS.DACTPU_RSP                       PAGE 7-56                │
 │                 CS.DELETENR_RSP                     PAGE 7-109               │
 │                 CS.FNA_RSP                          PAGE 7-102               │
 │                 CS.INITPROC_RSP                     PAGE 7-88                │
 │                 CS.INOP_PROC                        PAGE 7-110               │
 │                 CS.LDREQD_PROC                      PAGE 7-86                │
 │                 CS.LINK_RSP                         PAGE 7-67                │
 │                 CS.LOAD_DUMP_RPO_RSP                PAGE 7-84                │
 │                 CS.LU_PROC                          PAGE 7-58                │
 │                 CS.LU_RSP                           PAGE 7-60                │
 │                 CS.PROCSTAT_PROC                    PAGE 7-89                │
 │                 CS.PU_PROC                          PAGE 7-52                │
 │                 CS.PU_T2_LOAD_RSP                   PAGE 7-92                │
 │                 CS.REQCONT_REQDISCONT_PROC          PAGE 7-114               │
 │                 CS.RNAA_RSP                         PAGE 7-95                │
 │                 UPM_ANA_RSP                         PAGE 7-123               │
 │                 UPM_ER_VR_INOP_PROC                 PAGE 7-127               │
 │                 UPM_LCP_PROC                        PAGE 7-127               │
 │                 UPM_NS_LSA_PROC                     PAGE 7-124               │
 │                 UPM_SLOW_PROC                       PAGE 7-123               │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL TARGET_NA BIT(48);
  DCL REQUEST_CODE BIT(8);

  IF SERVICE_TYPE = NETWORK_SERVICES THEN
    DO;
    . IF NS_RQ_CODE = (REQDISCONT | LDREQD | NS_LSA | LCP | ER_INOP | VR_INOP) THEN
    .   TARGET_NA = OSAF||OEF;
    . ELSE
    .   TARGET_NA = OSAF||(NSC_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
    .                                                   /* APPENDIX A          */
    . REQUEST_CODE = NS_RQ_CODE;
    END;

  ELSE
    DO;
    . TARGET_NA = OSAF||OEF;
    . REQUEST_CODE = RQ_CODE;
    END;

  IF RRI = RSP THEN
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │              INPUT IS A RESPONSE.        │
                  └──────────────────────────────────────────┘
                                                                                */
    SELECT ANYORDER;
    .
    . WHEN(RU_CTGY = SC & REQUEST_CODE = ACTPU)
    .   CALL CS.ACTPU_RSP;                              /* PAGE 7-54           */
    .
    . WHEN(RU_CTGY = SC & REQUEST_CODE = DACTPU)
    .   CALL CS.DACTPU_RSP;                             /* PAGE 7-56           */
    .
    . WHEN(RU_CTGY = SC & REQUEST_CODE = SDT)
    .   DISCARD MU;
    .
    . WHEN(RU_CTGY = SC & REQUEST_CODE = (ACTLU | DACTLU))
    .   CALL CS.LU_RSP;                                 /* PAGE 7-60           */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = (ACTLINK | DACTLINK))
    .   CALL CS.LINK_RSP;                               /* PAGE 7-67           */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = (ADDLINK | ADDLINKSTA))
    .   CALL CS.ADDLINK_ADDLINKSTA_RSP;                 /* PAGE 7-107          */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = DELETENR)
    .   CALL CS.DELETENR_RSP;                           /* PAGE 7-109          */
    .
    . WHEN(RU_CTGY = FMD &
    .      REQUEST_CODE = (ACTCONNIN | DACTCONNIN | CONNOUT | ABCONNOUT | ABCONN))
    .   CALL CS.CONN_RSP;                               /* PAGE 7-70           */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = (CONTACT | DISCONTACT))
    .   CALL CS.CONTACT_DISCONTACT_RSP;                 /* PAGE 7-76           */
```

```
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = ANA)
    .   CALL UPM_ANA_RSP;                                    /* PAGE 7-123          */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = RNAA)
    .   CALL CS.RNAA_RSP;                                    /* PAGE 7-95           */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = FNA)
    .   CALL CS.FNA_RSP;                                     /* PAGE 7-102          */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = (DUMPINIT | DUMPTEXT | DUMPFINAL))
    .   CALL CS.LOAD_DUMP_RPO_RSP;                           /* PAGE 7-84           */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = (IPLINIT | IPLTEXT | IPLFINAL))
    .   CALL CS.LOAD_DUMP_RPO_RSP;                           /* PAGE 7-84           */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = RPO)
    .   CALL CS.LOAD_DUMP_RPO_RSP;                           /* PAGE 7-84           */
    .
    . WHEN(RU_CTGY = FMD &
    .      REQUEST_CODE = (NS_IPL_INIT | NS_IPL_TEXT | NS_IPL_FINAL | NS_IPL_ABORT))
    .   CALL CS.PU_T2_LOAD_RSP;                              /* PAGE 7-92           */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = INITPROC)
    .   CALL CS.INITPROC_RSP;                                /* PAGE 7-88           */
    .
    . WHEN(RU_CTGY = FMD & REQUEST_CODE = SETCV)
    .   SEND MU TO UPM_TRANSLATION_SVC;                      /* CHAPTER 6           */
    .
    . OTHERWISE
    .   DO;
    .   . CALL UPM_LOG;                                      /* APPENDIX B          */
    .   . DISCARD MU;
    .   END;
    END;

  ELSE
    IF RRI = RQ THEN
                                                                                     /*
                      ┌──────────────────────────────────────┐
                      │          INPUT IS A REQUEST.         │
                      └──────────────────────────────────────┘
                                                                                     */
      IF FIND_DOMAIN_RESOURCE(TARGET_NA) = NULL THEN         /* APPENDIX B          */
        DO;
        . CALL UPM_LOG;                                      /* APPENDIX B          */
        . DISCARD MU;
        END;

      ELSE
        SELECT ANYORDER;
        .
        . WHEN(RU_CTGY = SC & REQUEST_CODE = DACTPU)
        .   CALL CS.PU_PROC;                                 /* PAGE 7-52           */
        .
        . WHEN(RU_CTGY = SC & REQUEST_CODE = DACTLU)
        .   CALL CS.LU_PROC;                                 /* PAGE 7-58           */
        .
        . WHEN(RU_CTGY = FMD & REQUEST_CODE = INOP)
        .   CALL CS.INOP_PROC;                               /* PAGE 7-110          */
        .
        . WHEN(RU_CTGY = FMD & REQUEST_CODE = (REQCONT | REQDISCONT))
        .   CALL CS.REQCONT_REQDISCONT_PROC;                 /* PAGE 7-114          */
        .
        . WHEN(RU_CTGY = FMD & REQUEST_CODE = (ESLOW | EXSLOW))
        .   CALL UPM_SLOW_PROC;                              /* PAGE 7-123          */
        .
        . WHEN(RU_CTGY = FMD & REQUEST_CODE = CONTACTED)
        .   CALL CS.CONTACTED_PROC;                          /* PAGE 7-77           */
        .
        . WHEN(RU_CTGY = FMD & REQUEST_CODE = REQFNA)
        .   SEND MU TO UPM_TRANSLATION_SVC;                  /* CHAPTER 6           */
        .
        . WHEN(RU_CTGY = FMD & REQUEST_CODE = PROCSTAT)
        .   CALL CS.PROCSTAT_PROC;                           /* PAGE 7-89           */
        .
        . WHEN(RU_CTGY = FMD & REQUEST_CODE = LDREQD)
        .   CALL CS.LDREQD_PROC;                             /* PAGE 7-86           */
        .
        . WHEN(RU_CTGY = FMD & REQUEST_CODE = REQACTLU)
        .   SEND MU TO UPM_TRANSLATION_SVC;                  /* CHAPTER 6           */
        .
        . WHEN(RU_CTGY = FMD & REQUEST_CODE = NS_LSA)
        .   CALL UPM_NS_LSA_PROC;                            /* PAGE 7-124          */
        .
        . WHEN(RU_CTGY = FMD & REQUEST_CODE = (ER_INOP | VR_INOP))
        .   CALL UPM_ER_VR_INOP_PROC;                        /* PAGE 7-127          */
        .
        . WHEN(RU_CTGY = FMD & REQUEST_CODE = LCP)
        .   CALL UPM_LCP_PROC;                               /* PAGE 7-127          */
        .
        . OTHERWISE
        .   DO;
        .   . CALL UPM_LOG;                                  /* APPENDIX B          */
        .   . DISCARD MU;
        .   END;
        END;

END SSCP.SVC_MGR.CS.RCV;
```

```
CS.PU_PROC: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE  HANDLES THE ACTIVATION  AND DEACTIVATION  OF SUBAREA
│                 AND PERIPHERAL PU'S.
│
│                 IF THE  TARGET PU  IS A PERIPHERAL  PU, THE  PU'S ASSOCIATED  ALS IS
│                 CHECKED TO SEE IF IT IS ACTIVE.  IF THE ALS FSM IS NOT IN THE ACTIVE
│                 STATE,  THE  PROCEDURE  RESOURCE_ACTIVE_CHECK,  WHICH  PERFORMS  THE
│                 CHECKING,    INSERTS    THE    ACTPU    REQUEST    INTO    THE    ALS'S
│                 SAVE_MU_FOR_RETRY_LIST.   IF THE  ALS FSM  IS IN  THE ACTIVE  STATE,
│                 PROCESSING OF THE REQUEST CONTINUES IMMEDIATELY.
│
│                 WHEN ACTPU IS THE INPUT AND THE FSM  OF THE TARGET PU IS RESET, THIS
│                 PROCEDURE  SENDS   THE  ACTPU   REQUEST  TO  THE   PU  FSM   AND  TO
│                 PU.SVC_MGR.CSC_MGR.SEND.  IF THE PU FSM IS NOT RESET, THIS PROCEDURE
│                 GENERATES A SEND_CHECK, WHICH IS SENT TO UPM_TRANSLATION_SVC.
│
│                 WHEN DACTPU IS THE INPUT, THE TARGET PU  FSM IS CHECKED TO SEE IF IT
│                 IS IN THE RESET  STATE.  IF THE PU FSM IS  NOT RESET, THIS PROCEDURE
│                 SENDS THE DACTPU REQUEST TO  THE FSM AND TO PU.SVC_MGR.CSC_MGR.SEND.
│                 IF THE PU FSM IS RESET, THIS PROCEDURE GENERATES A SEND_CHECK, WHICH
│                 IS SENT TO UPM_TRANSLATION_SVC.
│
│      INPUT:     ACTPU OR DACTPU FROM UPM_TRANSLATION_SVC  (CHAPTER 6); OR ACTPU FROM
│                 CS.CONTACTED_PROC (PAGE  7-77); OR DACTPU  FROM SSCP.SVC_MGR.SS.SEND
│                 (CHAPTER    8),    FROM    CS.PROCSTAT_PROC    (PAGE   7-89),    FROM
│                 CS.PU_T2_IPL_ABORT  (PAGE  7-93),  OR  FROM  CS.DEACTIVATION_CLEANUP
│                 (PAGE 7-119)
│
│      OUTPUT:    ACTPU OR DACTPU  TO PU.SVC_MGR.CSC_MGR.SEND (CHAPTER 13)  AND TO THE
│                 PU FSM AND A COPY OF  THE TARGET ADDRESS TO UPM_SAVE_TARGET_NA (PAGE
│                 7-122),  OR  A   SEND_CHECK  WITH  AN  APPROPRIATE   ERROR  CODE  TO
│                 UPM_TRANSLATION_SVC (CHAPTER 6),  OR THE RESET SIGNAL TO  THE PU FSM
│                 AND TO ASSOCIATED LU FSM'S AND DACTPU TO UPM_TRANSLATION_SVC
│
│      NOTE:      PROCESSING OF THIS REQUEST RESUMES IN CS.ACTPU_RSP (PAGE 7-54) OR IN
│                 CS.DACTPU_RSP  (PAGE   7-56)  WHEN   PU.SVC_MGR.CSC_MGR  RETURNS   A
│                 RESPONSE.
│
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                 CS.CONTACTED_PROC                 PAGE 7-77
│                 CS.DEACTIVATION_CLEANUP           PAGE 7-119
│                 CS.PROCSTAT_PROC                  PAGE 7-89
│                 CS.PU_T2_IPL_ABORT                PAGE 7-93
│                 SSCP.SVC_MGR.CS.RCV               PAGE 7-50
│                 SSCP.SVC_MGR.CS.SEND              PAGE 7-48
│
│      REFERS TO THE FOLLOWING PROCEDURE(S):
│                 FSM_PU_ACT_DOM_RES                PAGE 7-128
│                 RESOURCE_ACTIVE_CHECK             PAGE 7-116
│                 UPM_SAVE_TARGET_NA                PAGE 7-122
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL ALS_NA BIT(48);
DCL ASSOC_PTR POINTER;
DCL PU_NA BIT(48);
DCL TARGET_NA BIT(48);

TARGET_NA = DSAF||DEF;
DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);                  /* APPENDIX B          */

IF DRCB.RESOURCE_CATEGORY = ALS THEN
  DRCB_PTR = FIND_SUBORDINATE_DOM_RES(TARGET_NA);            /* APPENDIX B          */

IF (DRCB_PTR = NULL |
     DRCB.RESOURCE_CATEGORY ¬= (SUBAREA_PU | PERIPHERAL_PU)) THEN
  SEND SEND_CHECK(X'0806') TO UPM_TRANSLATION_SVC;           /* RESOURCE UNKNOWN    */

ELSE
  DO;
  . PU_NA = DRCB.NETWORK_ADDRESS;
  .
  . IF DRCB.RESOURCE_CATEGORY = PERIPHERAL_PU THEN
  .   DO;
  .   . ALS_NA = PU_NA;
  .   . IF RESOURCE_ACTIVE_CHECK(ALS_NA,ALS) = NG THEN       /* PAGE 7-116          */
  .   .   RETURN;
  .   END;
  .
  . SELECT ANYORDER(RQ_CODE);
```

```
                                                                                /*
                        ┌──────────────────────────────────────┐
                        │                ACTPU                 │
                        └──────────────────────────────────────┘
                                                                                */
   . . WHEN(ACTPU)
   . .   DO;
   . .   . IF FSM_PU_ACT_DOM_RES = RESET THEN                  /* PAGE 7-128    */
   . .   .   DO;
   . .   .   . CALL FSM_PU_ACT_DOM_RES;                       /* PAGE 7-128    */
   . .   .   . CALL UPM_SAVE_TARGET_NA(TARGET_NA);            /* PAGE 7-122    */
   . .   .   . SEND MU TO PU.SVC_MGR.CSC_MGR.SEND;            /* CHAPTER 13    */
   . .   .   END;
   . .   .
   . .   . ELSE
   . .   .   SEND SEND_CHECK(X'0815') TO UPM_TRANSLATION_SVC; /* FUNCTION ACTIVE */
   . .   END;
                                                                                /*
                        ┌──────────────────────────────────────┐
                        │               DACTPU                 │
                        └──────────────────────────────────────┘
                                                                                */
   . . WHEN(DACTPU)
   . .   DO;
   . .   . IF FSM_PU_ACT_DOM_RES ¬= RESET THEN                 /* PAGE 7-128    */
   . .   .   DO;
   . .   .   . CALL FSM_PU_ACT_DOM_RES;                       /* PAGE 7-128    */
   . .   .   . CALL UPM_SAVE_TARGET_NA(TARGET_NA);            /* PAGE 7-122    */
   . .   .   . SEND MU TO PU.SVC_MGR.CSC_MGR.SEND;            /* CHAPTER 13    */
   . .   .   END;
   . .   .
   . .   . ELSE
   . .   .   SEND SEND_CHECK(X'0816') TO UPM_TRANSLATION_SVC; /* FUNCTION INACTIVE */
   . .   END;
   . END;
   END;

END CS.PU_PROC;
```

```
CS.ACTPU_RSP: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  IF THE  PU FSM IS  NOT IN THE  PEND_ACTIVE OR PEND_RESET  STATE, THE │
│                 ACTPU RESPONSE IS SENT TO UPM_LOG.   OTHERWISE, THE RESPONSE IS SENT │
│                 TO THE FSM AND TO UPM_TRANSLATION_SVC.                   │
│                                                                          │
│                 IF  THE PU  FSM IS  IN THE  PEND_ACTIVE  STATE AND  THE RESPONSE  IS │
│                 POSITIVE, THE  PU'S SAVE_MU_FOR_RETRY_LIST IS  CHECKED TO SEE  IF IT │
│                 CONTAINS ANY  ELEMENTS; IF SO, ALL  ARE REMOVED AND SENT  TO CS.SEND │
│                 (PAGE 7-48).                                             │
│                                                                          │
│                 IF  THE  RSP(ACTPU) IS  FROM  A  PU_T2  AND THE  RESPONSE  INDICATES │
│                 IPL_REQUIRED,  CS.INITIATE_IPL_PROC   (PAGE  7-91)   IS  CALLED   TO │
│                 DETERMINE WHETHER THE PU_T2  NODE IS TO BE LOADED BY  THE SSCP OR BY │
│                 THE SUBAREA PU ADJACENT TO THE PU_T2.                    │
│                                                                          │
│                 WHEN THE TARGET RESOURCE IS A PERIPHERAL PU WHOSE ASSOCIATED LINK IS │
│                 SWITCHED,  FURTHER PROCESSING  IS  PERFORMED.   IF THE  RESPONSE  IS │
│                 POSITIVE, AN RNAA  REQUEST IS GENERATED TO  OBTAIN NETWORK ADDRESSES │
│                 FOR  ALL OF  THE  LU'S SUBORDINATE  TO THE  PERIPHERAL  PU.  IF  THE │
│                 RESPONSE IS NEGATIVE, DISCONTACT AND ABCONN REQUESTS ARE GENERATED.  │
│                                                                          │
│                 WHEN THE TARGET RESOURCE IS A SUBAREA PU, THIS PROCEDURE GENERATES A │
│                 SDT REQUEST, WHICH IS SENT TO TC.SC.SEND.                │
│                                                                          │
│      INPUT:     POSITIVE OR  NEGATIVE RESPONSE TO ACTPU  FROM PU.SVC_MGR.CSC_MGR.RCV │
│                 (CHAPTER   13)   AND   A   COPY   OF   THE   TARGET   ADDRESS   FROM │
│                 UPM_RETRIEVE_TARGET_NA (PAGE 7-122)                      │
│                                                                          │
│      OUTPUT:    ±RSP(ACTPU) TO  UPM_TRANSLATION_SVC (CHAPTER 6)  AND TO THE  PU FSM, │
│                 AND, IF THE PU IS A PERIPHERAL PU WHOSE ASSOCIATED LINK IS SWITCHED, │
│                 RNAA OR BOTH DISCONTACT  AND ABCONN; IF THE PU IS  A SUBAREA PU, SDT │
│                 TO TC.SC.SEND (CHAPTER 4); OR ±RSP(ACTPU) TO UPM_LOG (APPENDIX B)    │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 SSCP.SVC_MGR.CS.RCV                PAGE 7-50             │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 CONTACT_DISCONTACT_SEND_CHECK      PAGE 7-118            │
│                 CS.CONN_PROC                       PAGE 7-68             │
│                 CS.DISCONTACT_PROC                 PAGE 7-74             │
│                 CS.INITIATE_IPL_PROC               PAGE 7-91             │
│                 CS.RNAA_PROC                       PAGE 7-94             │
│                 FSM_PU_ACT_DOM_RES                 PAGE 7-128            │
│                 UPM_RETRIEVE_TARGET_NA             PAGE 7-122            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL TARGET_NA BIT(48);
DCL LIST_PTR POINTER;
DCL PERIPHERAL_PU_NA BIT(48);
DCL SAVE_MU_PTR POINTER;
DCL P POINTER;
DCL RESPONSE_TYPE BIT(1);
DCL ADJ_PU_LOAD_CAP BIT(1);

TARGET_NA = UPM_RETRIEVE_TARGET_NA;                    /* PAGE 7-122      */
DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);            /* APPENDIX B      */

IF DRCB.RESOURCE_CATEGORY = ALS THEN
  DRCB_PTR = FIND_SUBORDINATE_DOM_RES(TARGET_NA);      /* APPENDIX B      */

IF (DRCB_PTR = NULL |
    DRCB.RESOURCE_CATEGORY ¬= (SUBAREA_PU | PERIPHERAL_PU)) THEN
  DO;
  . CALL UPM_LOG;                                      /* APPENDIX B      */
  . DISCARD MU;
  END;
```

```
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │ POSITIVE OR NEGATIVE RESPONSE TO ACTPU │
                    └──────────────────────────────────────┘
                                                                        */
ELSE
  DO;
  . IF FSM_PU_ACT_DOM_RES ¬= (PEND_ACTIVE | PEND_RESET) THEN   /* PAGE 7-128      */
  .   DO;
  .   . CALL UPM_LOG;                                         /* APPENDIX B      */
  .   . DISCARD MU;
  .   END;
  .
  . ELSE
  .   DO;
  .   . IF FSM_PU_ACT_DOM_RES = PEND_ACTIVE THEN
  .   .   DO;
  .   .   . IF RTI = POSITIVE THEN
  .   .   .   DO WHILE( ¬EMPTY(DRCB.SAVE_MU_FOR_RETRY_LIST));
  .   .   .   . LIST_PTR = FIRST_ENTRY(DRCB.SAVE_MU_FOR_RETRY_LIST);
  .   .   .   . REMOVE LIST_PTR->MU FROM DRCB.SAVE_MU_FOR_RETRY_LIST;
  .   .   .   . SEND LIST_PTR->MU TO SSCP.SVC_MGR.CS.SEND;  /* PAGE 7-48       */
  .   .   .   END;
  .   .   .
  .   .   . SAVE_MU_PTR = MU_PTR;
  .   .   . RESPONSE_TYPE = RTI;
  .   .   . IF DRCB.RESOURCE_CATEGORY = PERIPHERAL_PU THEN
  .   .   .   DO;
  .   .   .   . PERIPHERAL_PU_NA = DRCB.NETWORK_ADDRESS;
  .   .   .   . DRCB_PTR = FIND_LINK_FOR_DOM_RES(DRCB.NETWORK_ADDRESS);
  .   .   .   .                                               /* APPENDIX B      */
  .   .   .   . IF DRCB.SWITCHED_LINK = SWITCHED THEN
  .   .   .   .
  .   .   .   .   IF RESPONSE_TYPE = POSITIVE THEN
  .   .   .   .     DO;
  .   .   .   .     . MU_PTR = UPM_CREATE_RQ('RNAA');          /* APPENDIX B      */
  .   .   .   .     . RNAA_RQ.ASSIGNMENT_TYPE = RNAA_BF_LU;
  .   .   .   .     . I = 0;
  .   .   .   .     . SCAN DRCB_LIST PTR(DRCB_PTR);
  .   .   .   .     . . IF (DRCB.RESOURCE_CATEGORY = PERIPHERAL_LU &
  .   .   .   .     . .       (DRCB.ASSOCIATED_RES_PTR->DRCB.NETWORK_ADDRESS =
  .   .   .   .     . .       PERIPHERAL_PU_NA)) THEN
  .   .   .   .     . .   DO;
  .   .   .   .     . .   . RNAA_RQ.SUBFIELD(I,8:15) = DRCB.BF_LOCAL_ID;
  .   .   .   .     . .   . I = I + 1;
  .   .   .   .     . .   END;
  .   .   .   .     . SCANEND;
  .   .   .   .     . RNAA_RQ.ENTRY_CNT = I;
  .   .   .   .     . DSAP = OSAP;
  .   .   .   .     . CALL CS.RNAA_PROC;                     /* PAGE 7-94       */
  .   .   .   .     END;
  .   .   .   .
  .   .   .   .   ELSE                                        /* RESPONSE IS NEGATIVE */
  .   .   .   .     DO;
  .   .   .   .     . IF CONTACT_DISCONTACT_SEND_CHECK = OK THEN
  .   .   .   .     .                                         /* PAGE 7-118      */
  .   .   .   .     .   DO;
  .   .   .   .     .   . MU_PTR = UPM_CREATE_RQ('DISCONTACT');  /* APPENDIX B    */
  .   .   .   .     .   . DSAP = OSAP;
  .   .   .   .     .   . CALL CS.DISCONTACT_PROC;           /* PAGE 7-74       */
  .   .   .   .     .   . MU_PTR = UPM_CREATE_RQ('ABCONN');  /* APPENDIX B      */
  .   .   .   .     .   . DSAP = OSAP;
  .   .   .   .     .   . CALL CS.CONN_PROC;                 /* PAGE 7-68       */
  .   .   .   .     .   END;
  .   .   .   .     END;
  .   .   .   .
  .   .   .   . DRCB_PTR = FIND_DOMAIN_RESOURCE(PERIPHERAL_PU_NA);
  .   .   .   .                                               /* APPENDIX B      */
  .   .   .   . IF DRCB.PERIPHERAL_PU_TYPE = PU_T2 &
  .   .   .   .      ACTPU_RSP.TYPE_ACTIVATION = IPL_REQUIRED THEN
  .   .   .   .   DO;
  .   .   .   .   . P = ADDR(ACTPU_FMT2_RSP.CONTROL_VECTORS);
  .   .   .   .   . ADJ_PU_LOAD_CAP = P->CONTROL_VECTOR_TYPE_07.ADJ_PU_LOAD_CAPABILITY;
  .   .   .   .   . CALL CS.INITIATE_IPL_PROC(TARGET_NA,ADJ_PU_LOAD_CAP);
  .   .   .   .   .                                           /* PAGE 7-91       */
  .   .   .   .   END;
  .   .   .   END;
  .   .   .
  .   .   . ELSE
  .   .   .   DO;
  .   .   .   . MU_PTR = UPM_CREATE_RQ('SDT');               /* APPENDIX B      */
  .   .   .   . SEND MU TO TC.SC.SEND;                       /* CHAPTER 4       */
  .   .   .   END;
  .   .   .
  .   .   . MU_PTR = SAVE_MU_PTR;
  .   .   END;
  .   .
  .   . CALL FSM_PU_ACT_DOM_RES;                            /* PAGE 7-128      */
  .   . SEND MU TO UPM_TRANSLATION_SVC;                      /* CHAPTER 6       */
  .   END;
  END;

END CS.ACTPU_RSP;
```

```
CS.DACTPU_RSP: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  IF THE PU  FSM IS NOT IN  THE PEND_RESET OR RESET  STATE, THE DACTPU │
│                 RESPONSE IS SENT TO UPM_LOG.  OTHERWISE, THE RESPONSE IS SENT TO THE │
│                 FSM AND TO UPM_TRANSLATION_SVC.                          │
│                                                                          │
│                 IN ADDITION, IF THE  PU IS A SUBAREA PU, MULTIPLE  INOP REQUESTS ARE │
│                 GENERATED TO RESET  ALL LINKS AND ADJACENT LINK  STATIONS WITHIN THE │
│                 PU'S SUBAREA.                                            │
│                                                                          │
│                 IF  THE PU  IS  A PERIPHERAL  PU  AND THE  PU'S  ASSOCIATED LINK  IS │
│                 SWITCHED, THE NETWORK ADDRESS FIELD OF THE DOMAIN RESOURCE ENTRY FOR │
│                 THE PU IS SET TO 0 AND  THE ASSOCIATED RESOURCE POINTER FIELD IS SET │
│                 TO A NULL VALUE.  DISCONTACT AND ABCONN REQUESTS ARE THEN GENERATED. │
│                 IF THE PU'S ASSOCIATED LINK IS  NONSWITCHED, A DISCONTACT REQUEST IS │
│                 GENERATED.   IF  THE  SEND_CONTACT_IMMEDIATELY  BIT  OF  THE  DOMAIN │
│                 RESOURCE  ENTRY  FOR  THE  PU  IS ON,  A  CONTACT  REQUEST  IS  ALSO │
│                 GENERATED.     (IF   THE    BIT    IS   ON,    IT    WAS   SET    IN │
│                 CS.REQCONT_REQDISCONT_PROC, FOUND ON PAGE 7-114.)        │
│                                                                          │
│      INPUT:     POSITIVE OR NEGATIVE RESPONSE  TO DACTPU FROM PU.SVC_MGR.CSC_MGR.RCV │
│                 (CHAPTER   13)   AND   A   COPY   OF   THE   TARGET   ADDRESS   FROM │
│                 UPM_RETRIEVE_TARGET_NA (PAGE 7-122)                      │
│                                                                          │
│      OUTPUT:    ±RSP(DACTPU) TO UPM_TRANSLATION_SVC  (CHAPTER 6) AND TO  THE PU FSM, │
│                 AND ONE  OR MORE  OF THE FOLLOWING:  INOP, DISCONTACT,  ABCONN, AND │
│                 CONTACTED  (SEE  FUNCTION  DESCRIPTION ABOVE);  OR  ±RSP(DACTPU)  TO │
│                 UPM_LOG (APPENDIX B).   IF THE TARGET OF THE DACTPU  IS A PERIPHERAL │
│                 PU,  THEN THE  NETWORK  ADDRESS FIELD  AND  THE ASSOCIATED  RESOURCE │
│                 POINTER FIELD  OF THE  PU DOMAIN RESOURCE  ENTRY ARE  SET TO  A NULL │
│                 VALUE.                                                   │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 SSCP.SVC_MGR.CS.RCV                PAGE 7-50             │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 CONTACT_DISCONTACT_SEND_CHECK      PAGE 7-118            │
│                 CS.CONN_PROC                       PAGE 7-68             │
│                 CS.CONTACT_PROC                    PAGE 7-72             │
│                 CS.DISCONTACT_PROC                 PAGE 7-74             │
│                 CS.INOP_PROC                       PAGE 7-110            │
│                 FSM_PU_ACT_DOM_RES                 PAGE 7-128            │
│                 UPM_RETRIEVE_TARGET_NA             PAGE 7-122            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL TARGET_NA BIT(48);
  DCL PU_PTR POINTER;

  TARGET_NA = UPM_RETRIEVE_TARGET_NA;                    /* PAGE 7-122     */
  DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);            /* APPENDIX B     */

  IF DRCB.RESOURCE_CATEGORY = ALS THEN
    DRCB_PTR = FIND_SUBORDINATE_DOM_RES(TARGET_NA);      /* APPENDIX B     */

  IF (DRCB_PTR = NULL |
      DRCB.RESOURCE_CATEGORY ¬= (SUBAREA_PU | PERIPHERAL_PU)) THEN
    DO;
    . CALL UPM_LOG;                                      /* APPENDIX B     */
    . DISCARD MU;
    END;
```

```
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │  POSITIVE OR NEGATIVE RESPONSE TO DACTPU  │
                    └──────────────────────────────────────┘
                                                                        */
ELSE
  DO;
  . IF FSM_PU_ACT_DOM_RES ¬= (PEND_RESET | RESET) THEN          /* PAGE 7-128   */
  .   DO;
  .   . CALL UPM_LOG;                                          /* APPENDIX B   */
  .   . DISCARD MU;
  .   END;
  .
  . ELSE
  .   DO;
  .   . PU_PTR = DRCB_PTR;
  .   . CALL FSM_PU_ACT_DOM_RES;                               /* PAGE 7-128   */
  .   . SEND MU TO UPM_TRANSLATION_SVC;                        /* CHAPTER 6    */
  .   .
  .   . IF DRCB.RESOURCE_CATEGORY = SUBAREA_PU THEN
  .   .   SCAN DRCB_LIST PTR(DRCB_PTR);
  .   .   . IF DRCB.RESOURCE_CATEGORY = LINK &
  .   .   .       DRCB.NETWORK_ADDRESS(0:31) = TARGET_NA(0:31) THEN
  .   .   .   DO;
  .   .   .   . MU_PTR = UPM_CREATE_RQ('INOP');                /* APPENDIX B   */
  .   .   .   . INOP_RQ.INOP_LINK_OR_ALS_ADDRESS =
  .   .   .   .       DRCB.NETWORK_ADDRESS(32:47);
  .   .   .   . CALL CS.INOP_PROC;                            /* PAGE 7-110   */
  .   .   .   END;
  .   .   SCANEND;
  .   .
  .   . ELSE
  .   .   DO;
  .   .   . DRCB_PTR = FIND_LINK_FOR_DOM_RES(TARGET_NA);      /* APPENDIX B   */
  .   .   .
  .   .   . IF CONTACT_DISCONTACT_SEND_CHECK = OK THEN        /* PAGE 7-118   */
  .   .   .
  .   .   .   IF DRCB.SWITCHED_LINK = SWITCHED THEN
  .   .   .     DO;
  .   .   .     . PU_PTR->DRCB.ASSOCIATED_RES_PTR = NULL;
  .   .   .     . PU_PTR->DRCB.NETWORK_ADDRESS = 0;
  .   .   .     . MU_PTR = UPM_CREATE_RQ('DISCONTACT');       /* APPENDIX B   */
  .   .   .     . DSAF = OSAF;
  .   .   .     . CALL CS.DISCONTACT_PROC;                    /* PAGE 7-74    */
  .   .   .     . MU_PTR = UPM_CREATE_RQ('ABCONN');           /* APPENDIX B   */
  .   .   .     . DSAF = OSAF;
  .   .   .     . CALL CS.CONN_PROC;                          /* PAGE 7-68    */
  .   .   .     END;
  .   .   .
  .   .   .   ELSE
  .   .   .     DO;
  .   .   .     . MU_PTR = UPM_CREATE_RQ('DISCONTACT');       /* APPENDIX B   */
  .   .   .     . DSAF = OSAF;
  .   .   .     . CALL CS.DISCONTACT_PROC;                    /* PAGE 7-74    */
  .   .   .     . IF DRCB.SEND_CONTACT_IMMEDIATELY = YES THEN
  .   .   .     .   DO;
  .   .   .     .   . MU_PTR = UPM_CREATE_RQ('CONTACT');      /* APPENDIX B   */
  .   .   .     .   . DSAF = OSAF;
  .   .   .     .   . CALL CS.CONTACT_PROC;                   /* PAGE 7-72    */
  .   .   .     .   . DRCB.SEND_CONTACT_IMMEDIATELY = NO;
  .   .   .     .   END;
  .   .   .     END;
  .   .   END;
  .   END;
  END;

END CS.DACTPU_RSP;
```

```
CS.LU_PROC: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE  HANDLES THE ACTIVATION  AND DEACTIVATION  OF SUBAREA │
│                 AND PERIPHERAL LU'S.                                     │
│                                                                          │
│                 SOME  SUBAREA   LU'S  SUPPORT   PARALLEL  SESSIONS.   AN  LU   WITH │
│                 PARALLEL-SESSION CAPABILITY HAS ONE SECONDARY LU NETWORK ADDRESS AND │
│                 MULTIPLE  PRIMARY LU  NETWORK  ADDRESSES  ASSOCIATED WITH  IT.  THE │
│                 PRIMARY NETWORK ADDRESSES ARE ASSIGNED  VIA THE RNAA REQUEST.  WHILE │
│                 ALL LU'S  THAT DO NOT SUPPORT  PARALLEL SESSIONS ARE SENT  ACTLU, AN │
│                 ACTLU REQUEST IS SENT ONLY TO THE SECONDARY LU ADDRESS FOR LU'S THAT │
│                 DO SUPPORT PARALLEL SESSIONS; THE PRIMARY LU ADDRESSES BECOME ACTIVE │
│                 WHEN THE SECONDARY LU ADDRESS DOES.                      │
│                                                                          │
│                 THE  TARGET LU'S  ASSOCIATED  RESOURCE (EITHER  A  SUBAREA  PU OR  A │
│                 PERIPHERAL PU) IS CHECKED TO SEE IF IT  IS ACTIVE.  IF THE PU FSM IS │
│                 NOT ACTIVE, THE PROCEDURE  RESOURCE_ACTIVE_CHECK, WHICH PERFORMS THE │
│                 CHECKING,    INSERTS    THE    ACTLU    REQUEST    INTO    THE    PU'S │
│                 SAVE_MU_FOR_RETRY_LIST.  IF THE PU FSM  IS ACTIVE, PROCESSING OF THE │
│                 REQUEST CONTINUES IMMEDIATELY.                           │
│                                                                          │
│                 WHEN ACTLU  IS THE  INPUT AND THE  LU FSM  IS RESET,  THIS PROCEDURE │
│                 SENDS    THE    ACTLU    REQUEST    TO    THE    LU    FSM    AND    TO │
│                 PU.SVC_MGR.CSC_MGR.SEND.   IF  THE  LU  FSM  IS  NOT  ACTIVE,  THIS │
│                 PROCEDURE    GENERATES    A    SEND_CHECK,    WHICH    IS    SENT    TO │
│                 UPM_TRANSLATION_SVC.                                     │
│                                                                          │
│                 WHEN DACTLU  IS THE INPUT,  THE LU FSM  IS CHECKED  TO SEE IF  IT IS │
│                 ACTIVE OR  PEND_ACTIVE OR PEND_RESET.   IF SO, THIS  PROCEDURE SENDS │
│                 THE DACTLU REQUEST TO THE LU FSM AND TO PU.SVC_MGR.CSC_MGR.SEND.  IF │
│                 THE  LU  FSM  IS  NOT ACTIVE  OR  PEND_ACTIVE  OR  PEND_RESET,  THIS │
│                 PROCEDURE    GENERATES    A    SEND_CHECK,    WHICH    IS    SENT    TO │
│                 UPM_TRANSLATION_SVC.                                     │
│                                                                          │
│      INPUT:     ACTLU OR DACTLU FROM UPM_TRANSLATION_SVC  (CHAPTER 6); OR ACTLU FROM │
│                 CS.PERIPHERAL_LU_ADD     (PAGE     7-97);     OR     DACTLU     FROM │
│                 SSCP.SVC_MGR.SS.SEND  (CHAPTER  8) OR  FROM  CS.DEACTIVATION_CLEANUP │
│                 (PAGE 7-119)                                             │
│                                                                          │
│      OUTPUT:    ACTLU OR DACTLU  TO PU.SVC_MGR.CSC_MGR.SEND (CHAPTER 13)  AND TO THE │
│                 LU FSM AND A COPY OF  THE TARGET ADDRESS TO UPM_SAVE_TARGET_NA (PAGE │
│                 7-122),  OR  A   SEND_CHECK  WITH  AN  APPROPRIATE   ERROR  CODE  TO │
│                 UPM_TRANSLATION_SVC (CHAPTER 6),  OR THE RESET SIGNAL TO  THE LU FSM │
│                 AND DACTLU TO UPM_TRANSLATION_SVC                        │
│                                                                          │
│      NOTE:      PROCESSING OF  THIS REQUEST  RESUMES IN  CS.LU_RSP (PAGE  7-60) WHEN │
│                 PU.SVC_MGR.CSC_MGR RETURNS A RESPONSE.                   │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 CS.DEACTIVATION_CLEANUP          PAGE 7-119              │
│                 CS.PERIPHERAL_LU_ADD             PAGE 7-97               │
│                 SSCP.SVC_MGR.CS.RCV              PAGE 7-50               │
│                 SSCP.SVC_MGR.CS.SEND             PAGE 7-48               │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 FSM_LU_ACT_DOM_RES               PAGE 7-128              │
│                 RESOURCE_ACTIVE_CHECK            PAGE 7-116              │
│                 UPM_SAVE_TARGET_NA               PAGE 7-122              │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL PU_NA BIT(48);
  DCL TARGET_NA BIT(48);
  DCL P POINTER;

  TARGET_NA = DSAF||DEF;
  DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);                  /* APPENDIX B          */

  IF DRCB.RESOURCE_CATEGORY ¬= (SUBAREA_LU | PERIPHERAL_LU) THEN
    SEND SEND_CHECK(X'0806') TO UPM_TRANSLATION_SVC;           /* RESOURCE UNKNOWN    */

  ELSE
    DO;
    . P = DRCB.ASSOCIATED_RES_PTR;
    . PU_NA = P->DRCB.NETWORK_ADDRESS;
    .
    . IF RESOURCE_ACTIVE_CHECK(PU_NA,PU) = OK THEN             /* PAGE 7-116          */
    .
    .   SELECT ANYORDER(RQ_CODE);
```

```
                                                                                /*
                        ┌──────────────────────────────────────────┐
                        │                 ACTLU                    │
                        └──────────────────────────────────────────┘
                                                                                */
    .   . WHEN(ACTLU)
    .   .   DO;
    .   .   . IF FSM_LU_ACT_DOM_RES = RESET THEN                  /* PAGE 7-128   */
    .   .   .   DO;
    .   .   .   . CALL FSM_LU_ACT_DOM_RES;                       /* PAGE 7-128   */
    .   .   .   . CALL UPM_SAVE_TARGET_NA(TARGET_NA);            /* PAGE 7-122   */
    .   .   .   . SEND MU TO PU.SVC_MGR.CSC_MGR.SEND;            /* CHAPTER 13   */
    .   .   .   END;
    .   .   .
    .   .   . ELSE
    .   .   .   SEND SEND_CHECK(X'0815') TO UPM_TRANSLATION_SVC; /* FUNCTION ACTIVE */
    .   .   END;
                                                                                /*
                        ┌──────────────────────────────────────────┐
                        │                 DACTLU                   │
                        └──────────────────────────────────────────┘
                                                                                */
    .   . WHEN(DACTLU)
    .   .   DO;
    .   .   . IF FSM_LU_ACT_DOM_RES = (PEND_ACTIVE | ACTIVE | PEND_RESET) THEN
    .   .   .                                                     /* PAGE 7-128   */
    .   .   .   DO;
    .   .   .   . CALL FSM_LU_ACT_DOM_RES;                       /* PAGE 7-128   */
    .   .   .   . CALL UPM_SAVE_TARGET_NA(TARGET_NA);            /* PAGE 7-122   */
    .   .   .   . SEND MU TO PU.SVC_MGR.CSC_MGR.SEND;            /* CHAPTER 13   */
    .   .   .   END;
    .   .   .
    .   .   . ELSE
    .   .   .   SEND SEND_CHECK(X'0816') TO UPM_TRANSLATION_SVC; /* FUNCTION INACTIVE */
    .   .   END;
    .   END;
    END;

END CS.LU_PROC;
```

```
CS.LU_RSP: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  WHEN RSP(ACTLU) IS THE INPUT AND THE LU FSM IS NOT IN THE │
│                 PEND_ACTIVE OR PEND_RESET STATE, THE RESPONSE IS SENT TO UPM_LOG.
│                 OTHERWISE, THE RESPONSE IS SENT TO THE FSM AND TO        │
│                 UPM_TRANSLATION_SVC, AND A COPY OF THE RESPONSE IS SENT TO
│                 SSCP.SVC_MGR.SS.RCV. FURTHERMORE, IF THE RESPONSE IS POSITIVE AND
│                 THE LU FSM IS IN THE PEND_ACTIVE STATE, THE LU'S         │
│                 SAVE_MU_FOR_RETRY_LIST IS CHECKED TO SEE IF IT CONTAINS ANY
│                 ELEMENTS; IF SO, ALL ARE REMOVED AND SENT TO CS.SEND (PAGE 7-48).
│                                                                          │
│                 WHEN RSP(DACTLU) IS THE INPUT AND THE LU FSM IS NOT IN THE
│                 PEND_RESET OR RESET STATE THE RESPONSE IS SENT TO UPM_LOG.
│                 OTHERWISE, THE RESPONSE IS SENT TO THE FSM AND TO        │
│                 UPM_TRANSLATION_SVC. IF THE TARGET RESOURCE IS A PERIPHERAL LU
│                 WHOSE ASSOCIATED LINK IS SWITCHED AND THE RESPONSE IS POSITIVE, AN
│                 FNA REQUEST IS GENERATED TO FREE THE NETWORK ADDRESS CURRENTLY BEING
│                 USED FOR THE PERIPHERAL LU.                              │
│                                                                          │
│      INPUT:     POSTIVE OR NEGATIVE RESPONSE TO ACTLU OR DACTLU FROM     │
│                 PU.SVC_MGR.CSC_MGR.RCV (CHAPTER 13) AND A COPY OF THE TARGET ADDRESS
│                 FROM UPM_RETRIEVE_TARGET_NA (PAGE 7-122)                 │
│                                                                          │
│      OUTPUT:    ±RSP(ACTLU) TO UPM_TRANSLATION_SVC (CHAPTER 6), TO THE LU FSM, AND
│                 TO SSCP.SVC_MGR.SS.RCV (CHAPTER 8); OR ±RSP(DACTLU) TO   │
│                 UPM_TRANSLATION_SVC AND TO THE LU FSM AND FNA TO CS.FNA_PROC (PAGE
│                 7-99); OR ±RSP(ACTLU|DACTLU) TO UPM_LOG (APPENDIX B)     │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 SSCP.SVC_MGR.CS.RCV              PAGE 7-50               │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 CS.FNA_PROC                      PAGE 7-99               │
│                 FSM_LU_ACT_DOM_RES               PAGE 7-128              │
│                 UPM_RETRIEVE_TARGET_NA           PAGE 7-122              │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL TARGET_NA BIT(48);
DCL LIST_PTR POINTER;
DCL RESPONSE_TYPE BIT(1);

TARGET_NA = UPM_RETRIEVE_TARGET_NA;                     /* PAGE 7-122      */
DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);             /* APPENDIX B      */

IF DRCB.RESOURCE_CATEGORY ¬= (SUBAREA_LU | PERIPHERAL_LU) THEN
  DO;
  . CALL UPM_LOG;                                       /* APPENDIX B      */
  . DISCARD MU;
  END;

ELSE

  SELECT ANYORDER(RQ_CODE);
                                                                        /*
                ┌──────────────────────────────────────┐
                │  POSITIVE OR NEGATIVE RESPONSE TO ACTLU  │
                └──────────────────────────────────────┘
                                                                        */
  . WHEN(ACTLU)
  .   DO;
  .   . IF FSM_LU_ACT_DOM_RES ¬= (PEND_ACTIVE | PEND_RESET) THEN
  .   .                                                 /* PAGE 7-128      */
  .   .   DO;
  .   .   . CALL UPM_LOG;                               /* APPENDIX B      */
  .   .   . DISCARD MU;
  .   .   END;
  .   .
  .   . ELSE
  .   .   DO;
  .   .   . IF (FSM_LU_ACT_DOM_RES = PEND_ACTIVE & RTI = POSITIVE) THEN
  .   .   .   DO WHILE( ¬EMPTY(DRCB.SAVE_MU_FOR_RETRY_LIST));
  .   .   .   . LIST_PTR = FIRST_ENTRY(DRCB.SAVE_MU_FOR_RETRY_LIST);
  .   .   .   . REMOVE LIST_PTR->MU FROM DRCB.SAVE_MU_FOR_RETRY_LIST;
  .   .   .   . SEND LIST_PTR->MU TO SSCP.SVC_MGR.CS.SEND; /* PAGE 7-48      */
  .   .   .   END;
  .   .   . CALL FSM_LU_ACT_DOM_RES;                    /* PAGE 7-128      */
  .   .   . RESPONSE_TYPE = RTI;
  .   .   . SEND MU TO UPM_TRANSLATION_SVC;             /* CHAPTER 6       */
  .   .   . MU_PTR = UPM_CREATE_RSP('ACTLU');           /* APPENDIX A      */
  .   .   . RTI = RESPONSE_TYPE;
  .   .   . SEND MU TO SSCP.SVC_MGR.SS.RCV;             /* CHAPTER 8       */
  .   .   END;
  .   END;
```

```
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │  POSITIVE OR NEGATIVE RESPONSE TO DACTLU  │
                    └──────────────────────────────────────┘
                                                                        */
    . WHEN(DACTLU)
    .   DO;
    .   . IF FSM_LU_ACT_DOM_RES ¬= (PEND_RESET | RESET) THEN
    .   .                                                  /* PAGE 7-128   */
    .   .   DO;
    .   .   . CALL UPM_LOG;                               /* APPENDIX B   */
    .   .   . DISCARD MU;
    .   .   END;
    .   .
    .   . ELSE
    .   .   DO;
    .   .   . CALL FSM_LU_ACT_DOM_RES;                    /* PAGE 7-128   */
    .   .   . RESPONSE_TYPE = RTI;
    .   .   . SEND MU TO UPM_TRANSLATION_SVC;             /* CHAPTER 6    */
    .   .   . IF RESPONSE_TYPE = POSITIVE &
    .   .   .      DRCB.RESOURCE_CATEGORY = PERIPHERAL_LU THEN
    .   .   .   DO;
    .   .   .   . DRCB_PTR = FIND_LINK_FOR_DOM_RES(TARGET_NA);  /* APPENDIX B */
    .   .   .   . IF DRCB.SWITCHED_LINK = SWITCHED THEN
    .   .   .   .   DO;
    .   .   .   .   . MU_PTR = UPM_CREATE_RQ('FNA');      /* APPENDIX B   */
    .   .   .   .   . FNA_RQ.SUBFIELD(0) = TARGET_NA(32:47);
    .   .   .   .   . DSAF = OSAF;
    .   .   .   .   . CALL CS.FNA_PROC;                   /* PAGE 7-99    */
    .   .   .   .   END;
    .   .   .   END;
    .   .   END;
    .   END;
    END;

END CS.LU_RSP;
```

```
CS.LINK_PROC: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  THIS PROCEDURE HANDLES THE ACTIVATION AND DEACTIVATION OF LINKS.
│
│                THE TARGET LINK'S ASSOCIATED  SUBAREA PU IS CHECKED TO SEE  IF IT IS
│                ACTIVE.    IF   THE   PU   FSM   IS   NOT   ACTIVE,   THE   PROCEDURE
│                RESOURCE_ACTIVE_CHECK,  WHICH  PERFORMS THE  CHECKING,  INSERTS  THE
│                ACTLINK REQUEST INTO THE PU'S SAVE_MU_FOR_RETRY_LIST.  IF THE PU FSM
│                IS ACTIVE, PROCESSING OF THE REQUEST CONTINUES IMMEDIATELY.
│
│                WHEN ACTLINK IS THE INPUT AND THE  LINK FSM IS RESET, THIS PROCEDURE
│                SENDS THE ACTLINK REQUEST  TO THE LINK FSM AND TO  SNS.SEND.  IF THE
│                LINK FSM IS NOT RESET, THIS  PROCEDURE GENERATES A SEND_CHECK, WHICH
│                IS SENT TO UPM_TRANSLATION_SVC.
│
│                WHEN  DACTLINK  IS  THE  INPUT  AND   THE  LINK  FSM  IS  ACTIVE  OR
│                PEND_ACTIVE, THIS PROCEDURE  SENDS THE DACTLINK REQUEST  TO THE LINK
│                FSM AND TO SNS.SEND.  IF THE LINK  FSM IS NOT ACTIVE, THIS PROCEDURE
│                GENERATES A SEND_CHECK, WHICH IS SENT TO UPM_TRANSLATION_SVC.
│
│     INPUT:     ACTLINK OR DACTLINK FROM UPM_TRANSLATION_SVC (CHAPTER 6)
│
│     OUTPUT:    ACTLINK OR DACTLINK TO SNS.SEND (CHAPTER 6)  AND TO THE LINK FSM AND
│                A COPY OF THE TARGET ADDRESS  TO UPM_SAVE_TARGET_NA (PAGE 7-122), OR
│                A SEND_CHECK WITH  AN APPROPRIATE ERROR CODE  TO UPM_TRANSLATION_SVC
│                (CHAPTER 6)
│
│     NOTE:      PROCESSING OF THIS  REQUEST RESUMES IN CS.LINK_RSP  (PAGE 7-67) WHEN
│                SNS RETURNS A RESPONSE.
│
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                SSCP.SVC_MGR.CS.SEND                PAGE 7-48
│
│     REFERS TO THE FOLLOWING PROCEDURE(S):
│                CS.DACTLINK_SEND_CHECKS             PAGE 7-64
│                FSM_LINK_ACT_DOM_RES                PAGE 7-129
│                RESOURCE_ACTIVE_CHECK               PAGE 7-116
│                UPM_SAVE_TARGET_NA                  PAGE 7-122
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL TARGET_NA BIT(48);
DCL LINK_NA BIT(48);
DCL PU_NA BIT(48);
DCL P POINTER;


TARGET_NA = DSAF||(NSC_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
                                                        /* APPENDIX A      */

DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);             /* APPENDIX B      */

IF DRCB.RESOURCE_CATEGORY ¬= LINK THEN
  SEND SEND_CHECK(X'0806') TO UPM_TRANSLATION_SVC;      /* RESOURCE UNKNOWN */

ELSE
  DO;
  . LINK_NA = DRCB.NETWORK_ADDRESS;
  . P = FIND_PU_FOR_DOM_RES(LINK_NA);                   /* APPENDIX B      */
  . PU_NA = P->DRCB.NETWORK_ADDRESS;
  .
  . IF RESOURCE_ACTIVE_CHECK(PU_NA,PU) = OK THEN        /* PAGE 7-116      */
  .
  .   SELECT ANYORDER(NS_RQ_CODE);
                                                        /*
                ┌──────────────────────────────────────┐
                │               ACTLINK                │
                └──────────────────────────────────────┘
                                                        */
  .   . WHEN(ACTLINK)
  .   .   DO;
  .   .   . IF FSM_LINK_ACT_DOM_RES = RESET THEN       /* PAGE 7-129      */
  .   .   .   DO;
  .   .   .   . CALL FSM_LINK_ACT_DOM_RES;             /* PAGE 7-129      */
  .   .   .   . CALL UPM_SAVE_TARGET_NA(TARGET_NA);    /* PAGE 7-122      */
  .   .   .   . SEND MU TO SNS.SEND;                   /* CHAPTER 6       */
  .   .   .   END;
  .   .   .
  .   .   . ELSE
  .   .   .   SEND SEND_CHECK(X'0815') TO UPM_TRANSLATION_SVC; /* FUNCTION ACTIVE */
  .   .   END;
```

```
                                                                                /*
                          ┌──────────────────────────────────────────┐
                          │                DACTLINK                  │
                          └──────────────────────────────────────────┘
                                                                                */
   .    . WHEN(DACTLINK)
   .    .   DO;
   .    .   . IF CS.DACTLINK_SEND_CHECKS(LINK_NA) = OK THEN      /* PAGE 7-64      */
   .    .   .   DO;
   .    .   .   . IF FSM_LINK_ACT_DOM_RES = (ACTIVE | PEND_ACTIVE) THEN
   .    .   .   .                                                /* PAGE 7-129     */
   .    .   .   .     DO;
   .    .   .   .     . CALL FSM_LINK_ACT_DOM_RES;              /* PAGE 7-129     */
   .    .   .   .     . CALL UPM_SAVE_TARGET_NA(TARGET_NA);     /* PAGE 7-122     */
   .    .   .   .     . SEND MU TO SNS.SEND;                    /* CHAPTER 6      */
   .    .   .   .     END;
   .    .   .   .
   .    .   .   . ELSE
   .    .   .   .   SEND SEND_CHECK(X'0816') TO UPM_TRANSLATION_SVC;
   .    .   .   .                                                /* FUNCTION INACTIVE */
   .    .   .   END;
   .    .   .
   .    .   . ELSE
   .    .   .   SEND SEND_CHECK(X'081A') TO UPM_TRANSLATION_SVC; /* REQUEST SEQUENCE ERROR */
   .    .   END;
   .    END;
   END;

END CS.LINK_PROC;
```

```
CS.DACTLINK_SEND_CHECKS: PROCEDURE(RES_NA) RETURNS(BIT(1));

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE  PERFORMS STATE SEND CHECKS  ON A GROUP OF  FSM'S FOR │
│                 EVERY ADJACENT LINK STATION ASSOCIATED WITH A GIVEN LINK.  │
│                                                                          │
│      INPUT:     THE NETWORK ADDRESS OF THE LINK                          │
│                                                                          │
│      OUTPUT:    OK IF ALL FSM'S ARE IN THE RESET STATE; NG IF NOT         │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 CS.LINK_PROC                    PAGE 7-62                │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 FSM_ALS_CONNECTED_DOM_RES       PAGE 7-133               │
│                 FSM_ALS_CONTACT_DOM_RES         PAGE 7-130               │
│                 FSM_ALS_DUMP_DOM_RES            PAGE 7-131               │
│                 FSM_ALS_IPL_DOM_RES             PAGE 7-131               │
│                 FSM_ALS_RPO_DOM_RES             PAGE 7-132               │
│                 FSM_LINK_CONNIN_DOM_RES         PAGE 7-129               │
│                 FSM_LINK_CONNOUT_DOM_RES        PAGE 7-130               │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL RES_NA BIT(48);
  DCL CHECK BIT(1);
  DCL P POINTER;
  DCL SAVE_DRCB_PTR POINTER;

  SAVE_DRCB_PTR = DRCB_PTR;
  CHECK = OK;

  SCAN DRCB_LIST PTR(P) WHILE(CHECK = OK);
  .
  . IF P->DRCB.RESOURCE_CATEGORY = ALS THEN
  .
  .   IF P->DRCB.ASSOCIATED_RES_PTR->DRCB.NETWORK_ADDRESS = RES_NA THEN
  .
  .     SELECT ANYORDER;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │            PRIMARY SWITCHED ALS          │
                  └──────────────────────────────────────────┘
                                                                        */
  .     . WHEN(P->DRCB.SWITCHED_LINK = SWITCHED
  .     .       & P->DRCB.LINK_DLC_ROLE = PRIMARY)
  .     .   DO;
  .     .   . IF FSM_LINK_CONNOUT_DOM_RES ¬= RESET |          /* PAGE 7-130      */
  .     .   .      FSM_LINK_CONNIN_DOM_RES ¬= RESET THEN      /* PAGE 7-129      */
  .     .   .   CHECK = NG;
  .     .   .
  .     .   . DRCB_PTR = P;                                   /* POINTER TO ALS  */
  .     .   . IF FSM_ALS_CONNECTED_DOM_RES ¬= RESET |         /* PAGE 7-133      */
  .     .   .      FSM_ALS_CONTACT_DOM_RES ¬= RESET THEN      /* PAGE 7-130      */
  .     .   .   CHECK = NG;
  .     .   END;
  .     .
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │          SECONDARY SWITCHED ALS          │
                  └──────────────────────────────────────────┘
                                                                        */
  .     . WHEN(P->DRCB.SWITCHED_LINK = SWITCHED
  .     .       & P->DRCB.LINK_DLC_ROLE = SECONDARY)
  .     .   DO;
  .     .   . IF FSM_LINK_CONNOUT_DOM_RES ¬= RESET |          /* PAGE 7-130      */
  .     .   .      FSM_LINK_CONNIN_DOM_RES ¬= RESET THEN      /* PAGE 7-129      */
  .     .   .   CHECK = NG;
  .     .   .
  .     .   . DRCB_PTR = P;                                   /* POINTER TO ALS  */
  .     .   . IF FSM_ALS_CONNECTED_DOM_RES ¬= RESET |         /* PAGE 7-133      */
  .     .   .      FSM_ALS_CONTACT_DOM_RES ¬= RESET |         /* PAGE 7-130      */
  .     .   .      FSM_ALS_IPL_DOM_RES ¬= RESET |             /* PAGE 7-131      */
  .     .   .      FSM_ALS_DUMP_DOM_RES ¬= RESET |            /* PAGE 7-131      */
  .     .   .      FSM_ALS_RPO_DOM_RES ¬= RESET THEN          /* PAGE 7-132      */
  .     .   .   CHECK = NG;
  .     .   END;
```

```
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │        PRIMARY NONSWITCHED ALS       │
                    └──────────────────────────────────────┘
                                                                        */
      . WHEN(P->DRCB.SWITCHED_LINK ¬= SWITCHED
      .      & P->DRCB.LINK_DLC_ROLE = PRIMARY)
      .   DO;
      .   . DRCB_PTR = P;                                  /* POINTER TO ALS      */
      .   . IF FSM_ALS_CONTACT_DOM_RES ¬= RESET THEN       /* PAGE 7-130          */
      .   .   CHECK = NG;
      .   END;
      .
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │      SECONDARY NONSWITCHED ALS       │
                    └──────────────────────────────────────┘
                                                                        */
      . WHEN(P->DRCB.SWITCHED_LINK ¬= SWITCHED
      .      & P->DRCB.LINK_DLC_ROLE = SECONDARY)
      .   DO;
      .   . DRCB_PTR = P;                                  /* POINTER TO ALS      */
      .   . IF FSM_ALS_CONTACT_DOM_RES ¬= RESET |          /* PAGE 7-130          */
      .   .      FSM_ALS_IPL_DOM_RES ¬= RESET |            /* PAGE 7-131          */
      .   .      FSM_ALS_DUMP_DOM_RES ¬= RESET |           /* PAGE 7-131          */
      .   .      FSM_ALS_RPO_DOM_RES ¬= RESET THEN         /* PAGE 7-132          */
      .   .   CHECK = NG;
      .   END;
      . END;
      .
  SCANEND;

  DRCB_PTR = SAVE_DRCB_PTR;

  RETURN(CHECK);

END CS.DACTLINK_SEND_CHECKS;
```

This page intentionally left blank

```
CS.LINK_RSP: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │     FUNCTION:  WHEN  RSP(ACTLINK) IS  THE INPUT  AND THE  LINK  FSM IS  NOT IN  THE │
 │                PEND_ACTIVE  OR  PEND_RESET  STATE,  THE RSP  IS  SENT  TO  UPM_LOG. │
 │                OTHERWISE, THE  RSP IS SENT TO  THE FSM AND  TO UPM_TRANSLATION_SVC. │
 │                FURTHERMORE, IF THE RESPONSE IS POSITIVE AND  THE LINK FSM IS IN THE │
 │                PEND_ACTIVE STATE,  THE LINK'S SAVE_MU_FOR_RETRY_LIST IS  CHECKED TO │
 │                SEE IF IT CONTAINS ANY ELEMENTS; IF  SO, ALL ARE REMOVED AND SENT TO │
 │                CS.SEND (PAGE 7-48).                                  │
 │                                                                      │
 │                WHEN  RSP(DACTLINK)  IS  THE  INPUT  AND THE  LINK  FSM  IS  IN  THE │
 │                PEND_RESET   STATE,  THE   RSP   IS  SENT   TO   THE   FSM  AND   TO │
 │                UPM_TRANSLATION_SVC.  OTHERWISE, THE RSP IS SENT TO UPM_LOG.          │
 │                                                                      │
 │     INPUT:     POSITIVE OR  NEGATIVE RESPONSE TO  ACTLINK OR DACTLINK  FROM SNS.RCV │
 │                (CHAPTER   6)   AND   A   COPY   OF   THE   TARGET   ADDRESS   FROM │
 │                UPM_RETRIEVE_TARGET_NA (PAGE 7-122)                   │
 │                                                                      │
 │     OUTPUT:    ±RSP(ACTLINK|DACTLINK) TO UPM_TRANSLATION_SVC (CHAPTER 6) AND TO THE │
 │                LINK FSM, OR ±RSP(ACTLINK|DACTLINK) TO UPM_LOG (APPENDIX B)           │
 │                                                                      │
 │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
 │                SSCP.SVC_MGR.CS.RCV                PAGE 7-50          │
 │                                                                      │
 │     REFERS TO THE FOLLOWING PROCEDURE(S):                            │
 │                FSM_LINK_ACT_DOM_RES               PAGE 7-129         │
 │                UPM_RETRIEVE_TARGET_NA             PAGE 7-122         │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL TARGET_NA BIT(48);
  DCL LIST_PTR POINTER;

  TARGET_NA = UPM_RETRIEVE_TARGET_NA;                    /* PAGE 7-122   */
  DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);            /* APPENDIX B   */

  IF DRCB.RESOURCE_CATEGORY ¬= LINK THEN
    DO;
    . CALL UPM_LOG;                                      /* APPENDIX B   */
    . DISCARD MU;
    END;

  ELSE

    SELECT ANYORDER(NS_RQ_CODE);
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │    POSITIVE OR NEGATIVE RESPONSE TO ACTLINK    │
                  └──────────────────────────────────────┘
                                                                        */
    . WHEN(ACTLINK)
    .   DO;
    .   . IF FSM_LINK_ACT_DOM_RES ¬= (PEND_ACTIVE | PEND_RESET) THEN
    .   .                                                /* PAGE 7-129   */
    .   .   DO;
    .   .   . CALL UPM_LOG;                              /* APPENDIX B   */
    .   .   . DISCARD MU;
    .   .   END;
    .   .
    .   . ELSE
    .   .   DO;
    .   .   . IF (FSM_LINK_ACT_DOM_RES = PEND_ACTIVE & RTI = POSITIVE) THEN
    .   .   .   DO WHILE( ¬EMPTY(DRCB.SAVE_MU_FOR_RETRY_LIST));
    .   .   .   . LIST_PTR = FIRST_ENTRY(DRCB.SAVE_MU_FOR_RETRY_LIST);
    .   .   .   . REMOVE LIST_PTR->MU FROM DRCB.SAVE_MU_FOR_RETRY_LIST;
    .   .   .   . SEND LIST_PTR->MU TO SSCP.SVC_MGR.CS.SEND;  /* PAGE 7-48   */
    .   .   .   END;
    .   .   . CALL FSM_LINK_ACT_DOM_RES;                 /* PAGE 7-129   */
    .   .   . SEND MU TO UPM_TRANSLATION_SVC;            /* CHAPTER 6    */
    .   .   END;
    .   END;
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │    POSITIVE OR NEGATIVE RESPONSE TO DACTLINK   │
                  └──────────────────────────────────────┘
                                                                        */
    . WHEN(DACTLINK)
    .   DO;
    .   . IF FSM_LINK_ACT_DOM_RES ¬= PEND_RESET THEN     /* PAGE 7-129   */
    .   .   DO;
    .   .   . CALL UPM_LOG;                              /* APPENDIX B   */
    .   .   . DISCARD MU;
    .   .   END;
    .   .
    .   . ELSE
    .   .   DO;
    .   .   . CALL FSM_LINK_ACT_DOM_RES;                 /* PAGE 7-129   */
    .   .   . SEND MU TO UPM_TRANSLATION_SVC;            /* CHAPTER 6    */
    .   .   END;
    .   END;
    END;

END CS.LINK_RSP;
```

```
CS.CONN_PROC: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE HANDLES THE CONNECTION  AND DISCONNECTION OF SWITCHED │
│                 LINKS.                                                   │
│                                                                          │
│                 WHEN A REQUEST IS RECEIVED, THE TARGET  LINK IS CHECKED TO SEE IF IT │
│                 HAS BEEN ACTIVATED (I.E., SENT ACTLINK).  IF NOT, THEN THE PROCEDURE │
│                 RESOURCE_ACTIVE_CHECK,  WHICH  PERFORMS THE  CHECKING,  INSERTS  THE │
│                 REQUEST INTO THE  LINK'S SAVE_MU_FOR_RETRY_LIST. IF THE  LINK FSM IS │
│                 ACTIVE, PROCESSING OF THE REQUEST CONTINUES IMMEDIATELY.  │
│                                                                          │
│                 WHEN  ACTCONNIN,  DACTCONNIN,  OR ABCONNOUT  IS  RECEIVED,  AND  THE │
│                 PERTINENT FSM IS IN AN APPROPRIATE STATE, THE REQUEST IS SENT TO THE │
│                 FSM  AND  TO  SNS.SEND.   OTHERWISE,  THIS  PROCEDURE  GENERATES  A │
│                 SEND_CHECK, WHICH IS SENT TO UPM_TRANSLATION_SVC.  │
│                                                                          │
│                 WHEN CONNOUT IS RECEIVED AND THE  APPROPRIATE LINK AND ADJACENT LINK │
│                 STATION FSM'S ARE  RESET, THEN CONNOUT IS SENT TO  THE ADJACENT LINK │
│                 STATION FSM AND  TO SNS.SEND.  IF THE  CONNOUT IS FOR A  MANUAL DIAL │
│                 OUT OPERATION, THE TELEPHONE NUMBER OF THE LINK TO BE DIALED IS SENT │
│                 TO THE OPERATOR VIA UPM_MANUAL_DIAL (PAGE  7-126).  IF THE FSM'S ARE │
│                 NOT RESET, THIS  PROCEDURE GENERATES A SEND_CHECK, WHICH  IS SENT TO │
│                 UPM_TRANSLATION_SVC.  │
│                                                                          │
│                 WHEN   ABCONN  IS   RECEIVED,  THE   ABCONN  REQUEST   IS  SENT   TO │
│                 FSM_ALS_CONNECTED_DOM_RES AND TO SNS.SEND.  │
│                                                                          │
│      INPUT:     ACTCONNIN,     DACTCONNIN,     CONNOUT,    OR     ABCONNOUT     FROM │
│                 UPM_TRANSLATION_SVC (CHAPTER 6); OR  ABCONN FROM UPM_TRANSLATION_SVC │
│                 (CHAPTER 6), FROM CS.ACTPU_RSP (PAGE 7-54), FROM CS.DACTPU_RSP (PAGE │
│                 7-56),     FROM    CS.INOP_PROC     (PAGE     7-110),    OR     FROM │
│                 CS.REQCONT_REQDISCONT_PROC (PAGE 7-114)  │
│                                                                          │
│      OUTPUT:    A VALID REQUEST TO SNS.SEND (CHAPTER 6) AND TO THE FSM AND A COPY OF │
│                 THE  TARGET  ADDRESS  TO  UPM_SAVE_TARGET_NA   (PAGE  7-122),  OR  A │
│                 SEND_CHECK  WITH AN  APPROPRIATE ERROR  CODE TO  UPM_TRANSLATION_SVC │
│                 (CHAPTER 6)  │
│                                                                          │
│      NOTE:      PROCESSING OF THIS  REQUEST RESUMES IN CS.CONN_RSP  (PAGE 7-70) WHEN │
│                 SNS RETURNS A RESPONSE.  │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 CS.ACTPU_RSP                        PAGE 7-54            │
│                 CS.DACTPU_RSP                       PAGE 7-56            │
│                 CS.INOP_PROC                        PAGE 7-110           │
│                 CS.REQCONT_REQDISCONT_PROC          PAGE 7-114           │
│                 SSCP.SVC_MGR.CS.SEND                PAGE 7-48            │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 FSM_ALS_CONNECTED_DOM_RES           PAGE 7-133           │
│                 FSM_LINK_CONNIN_DOM_RES             PAGE 7-129           │
│                 FSM_LINK_CONNOUT_DOM_RES            PAGE 7-130           │
│                 RESOURCE_ACTIVE_CHECK               PAGE 7-116           │
│                 UPM_MANUAL_DIAL                     PAGE 7-126           │
│                 UPM_SAVE_TARGET_NA                  PAGE 7-122           │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL TARGET_NA BIT(48);
DCL LINK_NA BIT(48);
DCL SAVE_DRCB_PTR POINTER;

TARGET_NA = DSAF||(NSC_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
                                                        /* APPENDIX A      */
DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);             /* APPENDIX B      */

IF DRCB.RESOURCE_CATEGORY ¬= LINK THEN
  SEND SEND_CHECK(X'0806') TO UPM_TRANSLATION_SVC;      /* RESOURCE UNKNOWN */

ELSE
  IF DRCB.SWITCHED_LINK ¬= SWITCHED THEN
    SEND SEND_CHECK(X'080C') TO UPM_TRANSLATION_SVC;    /* PROCEDURE NOT SUPPORTED */

  ELSE
    DO;
    . LINK_NA = DRCB.NETWORK_ADDRESS;
    . IF RESOURCE_ACTIVE_CHECK(LINK_NA,LINK) = OK THEN  /* PAGE 7-116      */
    .
    .   SELECT ANYORDER(NS_RQ_CODE);
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │                ACTCONNIN                 │
                  └──────────────────────────────────────────┘
                                                                        */
    .   . WHEN(ACTCONNIN)
    .   .   DO;
    .   .   . IF FSM_LINK_CONNIN_DOM_RES ¬= RESET THEN  /* PAGE 7-129      */
    .   .   .   SEND SEND_CHECK(X'0815') TO UPM_TRANSLATION_SVC;
    .   .   .                                           /* FUNCTION ACTIVE */
    .   .   . ELSE
    .   .   .   DO;
    .   .   .   . CALL FSM_LINK_CONNIN_DOM_RES;         /* PAGE 7-129      */
    .   .   .   . CALL UPM_SAVE_TARGET_NA(TARGET_NA);   /* PAGE 7-122      */
    .   .   .   . SEND MU TO SNS.SEND;                  /* CHAPTER 6       */
    .   .   .   END;
    .   .   END;
```

```
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │                DACTCONNIN                │
                    └──────────────────────────────────────────┘
                                                                        */
      . WHEN(DACTCONNIN)
      .   DO;
      .   . IF FSM_LINK_CONNIN_DOM_RES ¬= ACTIVE THEN       /* PAGE 7-129    */
      .   .   SEND SEND_CHECK(X'0816') TO UPM_TRANSLATION_SVC;
      .   .                                                 /* FUNCTION INACTIVE */
      .   . ELSE
      .   .   DO;
      .   .   . CALL FSM_LINK_CONNIN_DOM_RES;              /* PAGE 7-129    */
      .   .   . CALL UPM_SAVE_TARGET_NA(TARGET_NA);        /* PAGE 7-122    */
      .   .   . SEND MU TO SNS.SEND;                       /* CHAPTER 6     */
      .   .   END;
      .   END;
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │                 CONNOUT                  │
                    └──────────────────────────────────────────┘
                                                                        */
      . WHEN(CONNOUT)
      .   DO;
      .   . IF FSM_LINK_CONNOUT_DOM_RES ¬= RESET THEN       /* PAGE 7-130    */
      .   .   SEND SEND_CHECK(X'0815') TO UPM_TRANSLATION_SVC;
      .   .                                                 /* FUNCTION ACTIVE */
      .   . ELSE
      .   .   DO;
      .   .   . SAVE_DRCB_PTR = DRCB_PTR;
      .   .   . DRCB_PTR = FIND_SUBORDINATE_DOM_RES(LINK_NA);
      .   .   .                                             /* APPENDIX B    */
      .   .   . IF FSM_ALS_CONNECTED_DOM_RES ¬= RESET THEN  /* PAGE 7-133    */
      .   .   .   SEND SEND_CHECK(X'0801') TO UPM_TRANSLATION_SVC;
      .   .   .                                             /* RESOURCE NOT AVAILABLE */
      .   .   . ELSE
      .   .   .   DO;
      .   .   .   . DRCB_PTR = SAVE_DRCB_PTR;
      .   .   .   . CALL FSM_LINK_CONNOUT_DOM_RES;         /* PAGE 7-130    */
      .   .   .   . CALL UPM_SAVE_TARGET_NA(TARGET_NA);    /* PAGE 7-122    */
      .   .   .   . SEND MU TO SNS.SEND;                   /* CHAPTER 6     */
      .   .   .   . IF CONNOUT_RQ.CONNECT_OUT_TYPE = MANUAL THEN
      .   .   .   .   CALL UPM_MANUAL_DIAL;               /* PAGE 7-126    */
      .   .   .   END;
      .   .   END;
      .   END;
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │               ABCONNOUT                  │
                    └──────────────────────────────────────────┘
                                                                        */
      . WHEN(ABCONNOUT)
      .   DO;
      .   . IF FSM_LINK_CONNOUT_DOM_RES ¬= ACTIVE THEN       /* PAGE 7-130    */
      .   .   SEND SEND_CHECK(X'0816') TO UPM_TRANSLATION_SVC;
      .   .                                                 /* FUNCTION INACTIVE */
      .   . ELSE
      .   .   DO;
      .   .   . CALL FSM_LINK_CONNOUT_DOM_RES;              /* PAGE 7-130    */
      .   .   . CALL UPM_SAVE_TARGET_NA(TARGET_NA);         /* PAGE 7-122    */
      .   .   . SEND MU TO SNS.SEND;                        /* CHAPTER 6     */
      .   .   END;
      .   END;
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │                 ABCONN                   │
                    └──────────────────────────────────────────┘
                                                                        */
      . WHEN(ABCONN)
      .   DO;
      .   . DRCB_PTR = FIND_ALS_FOR_DOM_RES(LINK_NA);       /* APPENDIX B    */
      .   . CALL FSM_ALS_CONNECTED_DOM_RES;                 /* PAGE 7-133    */
      .   . CALL UPM_SAVE_TARGET_NA(TARGET_NA);             /* PAGE 7-122    */
      .   . SEND MU TO SNS.SEND;                            /* CHAPTER 6     */
      .   END;
      . END;
    END;

END CS.CONN_PROC;
```

```
CS.CONN_RSP: PROCEDURE;
                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE CHECKS THE PERTINENT FSM TO SEE IF IT IS IN THE  │
│                 APPROPRIATE STATE TO RECEIVE THE INPUT RESPONSE.  IF SO, THE  │
│                 RESPONSE IS SENT TO THE FSM AND TO UPM_TRANSLATION_SVC.  WHEN  │
│                 +RSP(ABCONN) IS THE INPUT, THE ALS SUBTREE RESET PROCEDURE IS ALSO  │
│                 CALLED.                                                  │
│                                                                          │
│                 IF THE FSM IS NOT IN THE APPROPRIATE STATE TO RECEIVE THE INPUT, THE  │
│                 INPUT IS SENT TO UPM_LOG.                                │
│                                                                          │
│      INPUT:     ±RSP(ACTCONNIN|DACTCONNIN|CONNOUT|ABCONNOUT|ABCONN)  FROM  SNS.RCV  │
│                 (CHAPTER 6) AND A COPY OF THE TARGET ADDRESS FROM  │
│                 UPM_RETRIEVE_TARGET_NA (PAGE 7-122)                      │
│                                                                          │
│      OUTPUT:    RESPONSES THAT WERE RECEIVED AS INPUT ARE SENT TO  │
│                 UPM_TRANSLATION_SVC (CHAPTER 6) AND TO THE APPROPRIATE FSM, IF  │
│                 VALID; OTHERWISE, THE RESPONSES ARE SENT TO UPM_LOG (APPENDIX B).  │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 SSCP.SVC_MGR.CS.RCV              PAGE 7-50               │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 CS.ALS_SUBTREE_RESET             PAGE 7-113              │
│                 FSM_ALS_CONNECTED_DOM_RES        PAGE 7-133              │
│                 FSM_LINK_CONNIN_DOM_RES          PAGE 7-129              │
│                 FSM_LINK_CONNOUT_DOM_RES         PAGE 7-130              │
│                 UPM_RETRIEVE_TARGET_NA           PAGE 7-122              │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL TARGET_NA BIT(48);
DCL LINK_NA BIT(48);
DCL ALS_NA BIT(48);

TARGET_NA = UPM_RETRIEVE_TARGET_NA;                     /* PAGE 7-122      */
DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);             /* APPENDIX B      */

IF DRCB.RESOURCE_CATEGORY ¬= LINK THEN
  DO;
  . CALL UPM_LOG;                                       /* APPENDIX B      */
  . DISCARD MU;
  END;

ELSE

  SELECT ANYORDER(NS_RQ_CODE);
                                                                        /*
                ┌──────────────────────────────────────────┐
                │  POSITIVE OR NEGATIVE RESPONSE TO ACTCONNIN  │
                └──────────────────────────────────────────┘
                                                                        */
  . WHEN(ACTCONNIN)
  .   DO;
  .   . IF FSM_LINK_CONNIN_DOM_RES ¬= PEND_ACTIVE THEN  /* PAGE 7-129      */
  .   .   DO;
  .   .   . CALL UPM_LOG;                               /* APPENDIX B      */
  .   .   . DISCARD MU;
  .   .   END;
  .   .
  .   . ELSE
  .   .   DO;
  .   .   . CALL FSM_LINK_CONNIN_DOM_RES;               /* PAGE 7-129      */
  .   .   . SEND MU TO UPM_TRANSLATION_SVC;             /* CHAPTER 6       */
  .   .   END;
  .   END;
                                                                        /*
                ┌──────────────────────────────────────────┐
                │  POSITIVE OR NEGATIVE RESPONSE TO DACTCONNIN  │
                └──────────────────────────────────────────┘
                                                                        */
  . WHEN(DACTCONNIN)
  .   DO;
  .   . IF FSM_LINK_CONNIN_DOM_RES ¬= PEND_RESET THEN  /* PAGE 7-129      */
  .   .   DO;
  .   .   . CALL UPM_LOG;                               /* APPENDIX B      */
  .   .   . DISCARD MU;
  .   .   END;
  .   .
  .   . ELSE
  .   .   DO;
  .   .   . CALL FSM_LINK_CONNIN_DOM_RES;               /* PAGE 7-129      */
  .   .   . SEND MU TO UPM_TRANSLATION_SVC;             /* CHAPTER 6       */
  .   .   END;
  .   END;
```

```
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │  POSITIVE OR NEGATIVE RESPONSE TO CONNOUT  │
                    └──────────────────────────────────────┘
                                                                        */
   . WHEN(CONNOUT)
   .   DO;
   .   . IF FSM_LINK_CONNOUT_DOM_RES ¬= PEND_ACTIVE THEN      /* PAGE 7-130      */
   .   .   DO;
   .   .   . CALL UPM_LOG;                                   /* APPENDIX B      */
   .   .   . DISCARD MU;
   .   .   END;
   .   .
   .   . ELSE
   .   .   DO;
   .   .   . CALL FSM_LINK_CONNOUT_DOM_RES;                  /* PAGE 7-130      */
   .   .   . SEND MU TO UPM_TRANSLATION_SVC;                 /* CHAPTER 6       */
   .   .   END;
   .   END;
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │  POSITIVE OR NEGATIVE RESPONSE TO ABCONNOUT  │
                    └──────────────────────────────────────┘
                                                                        */
   . WHEN(ABCONNOUT)
   .   DO;
   .   . IF FSM_LINK_CONNOUT_DOM_RES ¬= PEND_RESET THEN      /* PAGE 7-130      */
   .   .   DO;
   .   .   . CALL UPM_LOG;                                   /* APPENDIX B      */
   .   .   . DISCARD MU;
   .   .   END;
   .   .
   .   . ELSE
   .   .   DO;
   .   .   . CALL FSM_LINK_CONNOUT_DOM_RES;                  /* PAGE 7-130      */
   .   .   . SEND MU TO UPM_TRANSLATION_SVC;                 /* CHAPTER 6       */
   .   .   END;
   .   END;
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │  POSITIVE OR NEGATIVE RESPONSE TO ABCONN  │
                    └──────────────────────────────────────┘
                                                                        */
   . WHEN(ABCONN)
   .   DO;
   .   . LINK_NA = DRCB.NETWORK_ADDRESS;
   .   . DRCB_PTR = FIND_ALS_FOR_DOM_RES(LINK_NA);           /* APPENDIX B      */
   .   . ALS_NA = DRCB.NETWORK_ADDRESS;
   .   .
   .   . IF FSM_ALS_CONNECTED_DOM_RES ¬= (RESET | PEND_RESET) THEN
   .   .                                                     /* PAGE 7-133      */
   .   .   DO;
   .   .   . CALL UPM_LOG;                                   /* APPENDIX B      */
   .   .   . DISCARD MU;
   .   .   END;
   .   .
   .   . ELSE
   .   .   DO;
   .   .   . CALL FSM_ALS_CONNECTED_DOM_RES;                 /* PAGE 7-133      */
   .   .   . IF RTI = POSITIVE THEN                          /* RESPONSE IS POSITIVE */
   .   .   .   CALL CS.ALS_SUBTREE_RESET(AIS_NA);            /* PAGE 7-113      */
   .   .   . SEND MU TO UPM_TRANSLATION_SVC;                 /* CHAPTER 6       */
   .   .   END;
   .   END;
   END;

END CS.CONN_RSP;
```

```
CS.CONTACT_PROC: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE HANDLES THE CONTACTING OF REMOTE NODES.   │
│                                                                          │
│                 IF  THE  LINK  IS  SWITCHED,  THE  CONTACT  WAS GENERATED  BY │
│                 CS.REQCONT_REQDISCONT_PROC  (PAGE 7-114)  AS  A  RESULT  OF  HAVING │
│                 RECEIVED A REQCONT REQUEST.   CS.REQCONT_REQDISCONT_PROC CREATES THE │
│                 CONTACT AND CALLS THIS PROCEDURE.                        │
│                                                                          │
│                 WHEN  CONTACT  IS  RECEIVED,  THE  TARGET  ADJACENT  LINK  STATION'S │
│                 ASSOCIATED LINK  IS CHECKED TO SEE  IF IT HAS BEEN  ACTIVATED (I.E., │
│                 SENT ACTLINK).   IF THE LINK FSM  IS NOT ACTIVE, THEN  THE PROCEDURE │
│                 RESOURCE_ACTIVE_CHECK,  WHICH  PERFORMS THE  CHECKING,  INSERTS  THE │
│                 CONTACT REQUEST INTO THE LINK'S SAVE_MU_FOR_RETRY_LIST.  │
│                                                                          │
│                 IF THE LINK  FSM IS ACTIVE, THE FSM'S CORRESPONDING  TO THE ADJACENT │
│                 LINK STATION WHOSE  ADDRESS IS CONTAINED IN THE  CONTACT REQUEST ARE │
│                 CHECKED TO SEE IF  THEY ARE IN A SUITABLE STATE.   IF THEY ARE, THIS │
│                 PROCEDURE SENDS THE  CONTACT REQUEST TO SNS.SEND AND  TO THE CONTACT │
│                 FSM.  OTHERWISE,  THIS PROCEDURE  GENERATES A  SEND_CHECK, WHICH  IS │
│                 SENT TO UPM_TRANSLATION_SVC.                             │
│                                                                          │
│      INPUT:     CONTACT   FROM    UPM_TRANSLATION_SVC   (CHAPTER    6),   OR    FROM │
│                 CS.REQCONT_REQDISCONT_PROC (PAGE 7-114), OR FROM CS.DACTPU_RSP (PAGE │
│                 7-56)                                                    │
│                                                                          │
│      OUTPUT:    CONTACT TO  SNS.SEND (CHAPTER 6)  AND TO THE FSM  AND A COPY  OF THE │
│                 TARGET ADDRESS TO  UPM_SAVE_TARGET_NA (PAGE 7-122), OR  A SEND_CHECK │
│                 WITH AN APPROPRIATE ERROR CODE TO UPM_TRANSLATION_SVC (CHAPTER 6)  │
│                                                                          │
│      NOTE:      PROCESSING  OF  THIS REQUEST  RESUMES  IN  CS.CONTACT_DISCONTACT_RSP │
│                 (PAGE 7-76) WHEN SNS RETURNS A RESPONSE.                 │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 CS.DACTPU_RSP                    PAGE 7-56               │
│                 CS.REQCONT_REQDISCONT_PROC       PAGE 7-114              │
│                 SSCP.SVC_MGR.CS.SEND             PAGE 7-48               │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 FSM_ALS_CONTACT_DOM_RES          PAGE 7-130              │
│                 FSM_ALS_DUMP_DOM_RES             PAGE 7-131              │
│                 FSM_ALS_IPL_DOM_RES              PAGE 7-131              │
│                 FSM_ALS_RPO_DOM_RES              PAGE 7-132              │
│                 RESOURCE_ACTIVE_CHECK            PAGE 7-116              │
│                 UPM_SAVE_TARGET_NA               PAGE 7-122              │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL TARGET_NA BIT(48);
DCL ALS_NA BIT(48);
DCL LINK_NA BIT(48);
DCL LINK_PTR POINTER;

TARGET_NA = DSAF||(NSC_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
                                                        /* APPENDIX A      */
DRCB_PTR = FIND_ALS_FOR_DOM_RES(TARGET_NA);             /* APPENDIX B      */

IF DRCB.RESOURCE_CATEGORY ¬= ALS THEN
  SEND SEND_CHECK(X'0806') TO UPM_TRANSLATION_SVC;      /* RESOURCE UNKNOWN */

ELSE
  DO;
  . ALS_NA = DRCB.NETWORK_ADDRESS;
  . LINK_PTR = FIND_LINK_FOR_DOM_RES(ALS_NA);           /* APPENDIX B      */
  . LINK_NA = LINK_PTR->DRCB.NETWORK_ADDRESS;
  .
  . IF FSM_ALS_CONTACT_DOM_RES ¬= RESET THEN            /* PAGE 7-130      */
  .   SEND SEND_CHECK(X'0815') TO UPM_TRANSLATION_SVC;  /* FUNCTION ACTIVE */
  .
  . ELSE
  .   IF RESOURCE_ACTIVE_CHECK(LINK_NA,LINK) = OK THEN  /* PAGE 7-116      */
  .
  .     SELECT ANYORDER(DRCB.LINK_DLC_ROLE);
```

```
                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │ CONTACT A PRIMARY ADJACENT LINK STATION. │
                    └──────────────────────────────────────────┘
                                                                                */
.     . WHEN(PRIMARY)
.     .   DO;
.     .   . CALL FSM_ALS_CONTACT_DOM_RES;                    /* PAGE 7-130      */
.     .   . CALL UPM_SAVE_TARGET_NA(TARGET_NA);              /* PAGE 7-122      */
.     .   . SEND MU TO SWS.SEND;                             /* CHAPTER 6       */
.     .   END;

                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │ CONTACT A SECONDARY ADJACENT LINK STATION.│
                    └──────────────────────────────────────────┘
                                                                                */
.     . WHEN(SECONDARY)
.     .   DO;
.     .   . IF FSM_ALS_IPL_DOM_RES = RESET &                 /* PAGE 7-131      */
.     .   .      FSM_ALS_DUMP_DOM_RES = RESET &              /* PAGE 7-131      */
.     .   .      FSM_ALS_RPO_DOM_RES = RESET THEN            /* PAGE 7-132      */
.     .   .   DO;
.     .   .   . CALL FSM_ALS_CONTACT_DOM_RES;                /* PAGE 7-130      */
.     .   .   . CALL UPM_SAVE_TARGET_NA(TARGET_NA);          /* PAGE 7-122      */
.     .   .   . SEND MU TO SWS.SEND;                         /* CHAPTER 6       */
.     .   .   END;
.     .   .
.     .   . ELSE
.     .   .   SEND SEND_CHECK(X'0818') TO UPM_TRANSLATION_SVC;
.     .   .                                                  /* LINK PROC IN PROGRESS */
.     .   END;
.     . END;
. END;

END CS.CONTACT_PROC;
```

```
CS.DISCONTACT_PROC: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE HANDLES THE DISCONTACTING OF REMOTE NODES.  │
│                                                                          │
│                 WHEN  DISCONTACT IS  RECEIVED, THE  TARGET  ADJACENT LINK  STATION'S │
│                 ASSOCIATED LINK  IS CHECKED TO SEE  IF IT HAS BEEN  ACTIVATED (I.E., │
│                 SENT ACTLINK).   IF THE LINK FSM  IS NOT ACTIVE, THEN  THE PROCEDURE │
│                 RESOURCE_ACTIVE_CHECK,  WHICH  PERFORMS THE  CHECKING,  INSERTS  THE │
│                 DISCONTACT REQUEST INTO THE LINK'S SAVE_MU_FOR_RETRY_LIST.  │
│                                                                          │
│                 IF  THE LINK  FSM IS  ACTIVE, THEN  THE FSM'S  CORRESPONDING TO  THE │
│                 ADJACENT LINK STATION  WHOSE ADDRESS IS CONTAINED  IN THE DISCONTACT │
│                 REQUEST ARE CHECKED TO SEE IF THEY ARE IN A SUITABLE STATE.  IF THEY │
│                 ARE, THIS PROCEDURE SENDS THE DISCONTACT  REQUEST TO SNS.SEND AND TO │
│                 THE  DISCONTACT   FSM.   OTHERWISE,   THIS  PROCEDURE   GENERATES  A │
│                 SEND_CHECK, WHICH IS SENT TO UPM_TRANSLATION_SVC.  │
│                                                                          │
│      INPUT:     DISCONTACT    FROM    UPM_TRANSLATION_SVC    (CHAPTER    6),    FROM │
│                 CS.REQCONT_REQDISCONT_PROC  (PAGE  7-114), FROM  CS.ACTPU_RSP  (PAGE │
│                 7-54), OR FROM CS.DACTPU_RSP (PAGE 7-56)  │
│                                                                          │
│      OUTPUT:    DISCONTACT TO SNS.SEND (CHAPTER 6) AND TO  THE FSM AND A COPY OF THE │
│                 TARGET ADDRESS TO  UPM_SAVE_TARGET_NA (PAGE 7-122), OR  A SEND_CHECK │
│                 WITH AN APPROPRIATE ERROR CODE TO UPM_TRANSLATION_SVC (CHAPTER 6)  │
│                                                                          │
│      NOTE:      PROCESSING  OF  THIS REQUEST  RESUMES  IN  CS.CONTACT_DISCONTACT_RSP │
│                 (PAGE 7-76) WHEN SNS RETURNS A RESPONSE.  │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):  │
│                 CS.ACTPU_RSP                        PAGE 7-54  │
│                 CS.DACTPU_RSP                       PAGE 7-56  │
│                 CS.REQCONT_REQDISCONT_PROC          PAGE 7-114  │
│                 SSCP.SVC_MGR.CS.SEND                PAGE 7-48  │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):  │
│                 FSM_ALS_CONTACT_DOM_RES             PAGE 7-130  │
│                 FSM_ALS_DUMP_DOM_RES                PAGE 7-131  │
│                 FSM_ALS_IPL_DOM_RES                 PAGE 7-131  │
│                 FSM_ALS_RPO_DOM_RES                 PAGE 7-132  │
│                 RESOURCE_ACTIVE_CHECK               PAGE 7-116  │
│                 UPM_SAVE_TARGET_NA                  PAGE 7-122  │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL TARGET_NA BIT(48);
DCL ALS_NA BIT(48);
DCL LINK_NA BIT(48);
DCL LINK_PTR POINTER;

TARGET_NA = DSAF||(NSC_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
                                                        /* APPENDIX A      */
DRCB_PTP = FIND_ALS_FOR_DOM_RES(TARGET_NA);             /* APPENDIX B      */

IF DRCB.RESOURCE_CATEGORY ¬= ALS THEN
  SEND SEND_CHECK(X'0806') TO UPM_TRANSLATION_SVC;      /* RESOURCE UNKNOWN */

ELSE
  DO;
  . ALS_NA = DRCB.NETWORK_ADDRESS;
  . LINK_PTR = FIND_LINK_FOR_DOM_RES(ALS_NA);           /* APPENDIX B      */
  . LINK_NA = LINK_PTR->DRCB.NETWORK_ADDRESS;
  .
  . IF FSM_ALS_CONTACT_DOM_RES ¬= (RESET | PEND_ACTIVE_CONTACTED | ACTIVE) THEN
  .                                                     /* PAGE 7-130      */
  .   SEND SEND_CHECK(X'0809') TO UPM_TRANSLATION_SVC;  /* MODE INCONSISTENCY. */
  .
  . ELSE
  .   IF RESOURCE_ACTIVE_CHECK(LINK_NA,LINK) = OK THEN  /* PAGE 7-116      */
  .
  .     SELECT ANYORDER(DRCB.LINK_DLC_ROLE);
```

```
                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │  DISCONTACT A PRIMARY ADJACENT LINK STATION.  │
                    └──────────────────────────────────────────┘
                                                                                */
   .     . WHEN(PRIMARY)
   .     .   DO;
   .     .     . CALL FSM_ALS_CONTACT_DOM_RES;                  /* PAGE 7-130   */
   .     .     . CALL UPM_SAVE_TARGET_NA(TARGET_NA);            /* PAGE 7-122   */
   .     .     . SEND MU TO SNS.SEND;                           /* CHAPTER 6    */
   .     .     END;

                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │  DISCONTACT A SECONDARY ADJACENT LINK.STATION.  │
                    └──────────────────────────────────────────┘
                                                                                */
   .     . WHEN(SECONDARY)
   .     .   DO;
   .     .     . IF FSM_ALS_IPL_DOM_RES = RESET &               /* PAGE 7-131   */
   .     .     .      FSM_ALS_DUMP_DOM_RES = RESET &            /* PAGE 7-131   */
   .     .     .      FSM_ALS_RPO_DOM_RES = RESET THEN          /* PAGE 7-132   */
   .     .     .       DO;
   .     .     .       . CALL FSM_ALS_CONTACT_DOM_RES;          /* PAGE 7-130   */
   .     .     .       . CALL UPM_SAVE_TARGET_NA(TARGET_NA);    /* PAGE 7-122   */
   .     .     .       . SEND MU TO SNS.SEND;                   /* CHAPTER 6    */
   .     .     .       END;
   .     .     .
   .     .     . ELSE
   .     .     .       SEND SEND_CHECK(X'0818') TO UPM_TRANSLATION_SVC;
   .     .     .                                                /* LINK PROC IN PROGRESS */
   .     .     END;
   .     .   END;
   .     END;

END CS.DISCONTACT_PROC;
```

```
CS.CONTACT_DISCONTACT_RSP: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE CHECKS THE PERTINENT FSM TO SEE IF IT IS IN THE  │
│                 APPROPRIATE STATE TO RECEIVE THE INPUT RESPONSE.  IF SO, THE   │
│                 RESPONSE IS SENT TO THE FSM AND TO UPM_TRANSLATION_SVC.        │
│                                                                          │
│                 IF THE FSM IS NOT IN THE APPROPRIATE STATE TO RECEIVE THE INPUT, THE │
│                 INPUT IS SENT TO UPM_LOG.                                │
│                                                                          │
│                 IN ADDITION, IF RSP(DISCONTACT) IS THE INPUT, THIS PROCEDURE CALLS │
│                 CS.DEACTIVATION_CLEANUP (PAGE 7-119).  CS.DEACTIVATION_CLEANUP │
│                 GENERATES DACTPU(CLEANUP) OR DACTLU(CLEANUP) FOR EACH PERIPHERAL │
│                 PU|LU ASSOCIATED WITH THE TARGET ALS.                    │
│                                                                          │
│      INPUT:     POSITIVE OR NEGATIVE RESPONSE TO CONTACT OR DISCONTACT FROM SNS.RCV │
│                 (CHAPTER 6) AND A COPY OF THE TARGET ADDRESS FROM        │
│                 UPM_RETRIEVE_TARGET_NA (PAGE 7-122)                      │
│                                                                          │
│      OUTPUT:    ±RSP(CONTACT|DISCONTACT) TO UPM_TRANSLATION_SVC (CHAPTER 6) AND TO │
│                 THE APPROPRIATE FSM, OR ±RSP(CONTACT|DISCONTACT) TO UPM_LOG │
│                 (APPENDIX B)                                             │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 SSCP.SVC_MGR.CS.RCV              PAGE 7-50               │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 CS.DEACTIVATION_CLEANUP          PAGE 7-119              │
│                 FSM_ALS_CONTACT_DOM_RES          PAGE 7-130              │
│                 UPM_RETRIEVE_TARGET_NA           PAGE 7-122              │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL TARGET_NA BIT(48);
  DCL LINK_NA BIT(48);
  DCL LIST_PTR POINTER;

  TARGET_NA = UPM_RETRIEVE_TARGET_NA;                   /* PAGE 7-122      */
  DRCB_PTR = FIND_ALS_FOR_DOM_RES(TARGET_NA);           /* APPENDIX B      */

  IF DRCB.RESOURCE_CATEGORY ¬= ALS THEN
    DO;
    . CALL UPM_LOG;                                     /* APPENDIX B      */
    . DISCARD MU;
    END;

  ELSE

    SELECT ANYORDER(NS_RQ_CODE);

                                                                        /*
                  ┌──────────────────────────────────────┐
                  │  POSITIVE OR NEGATIVE RESPONSE TO CONTACT  │
                  └──────────────────────────────────────┘
                                                                        */
    . WHEN(CONTACT)
    .   DO;
    .   . IF FSM_ALS_CONTACT_DOM_RES ¬= PEND_ACTIVE_RSP THEN   /* PAGE 7-130  */
    .   .   DO;
    .   .   . CALL UPM_LOG;                             /* APPENDIX B      */
    .   .   . DISCARD MU;
    .   .   END;
    .   . ELSE
    .   .   DO;
    .   .   . CALL FSM_ALS_CONTACT_DOM_RES;             /* PAGE 7-130      */
    .   .   . SEND MU TO UPM_TRANSLATION_SVC;           /* CHAPTER 6       */
    .   .   END;
    .   END;
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │  POSITIVE OR NEGATIVE RESPONSE TO DISCONTACT  │
                  └──────────────────────────────────────┘
                                                                        */
    . WHEN(DISCONTACT)
    .   DO;
    .   . IF FSM_ALS_CONTACT_DOM_RES ¬= (RESET | PEND_RESET_CONTACTED | PEND_RESET_RSP) THEN
    .   .                                               /* PAGE 7-130      */
    .   .   DO;
    .   .   . CALL UPM_LOG;                             /* APPENDIX B      */
    .   .   . DISCARD MU;
    .   .   END;
    .   . ELSE
    .   .   DO;
    .   .   . CALL FSM_ALS_CONTACT_DOM_RES;             /* PAGE 7-130      */
    .   .   . LINK_NA = DRCB.ASSOCIATED_RES_PTR->DRCB.NETWORK_ADDRESS;
    .   .   . CALL CS.DEACTIVATION_CLEANUP(LINK_NA);    /* PAGE 7-119      */
    .   .   . SEND MU TO UPM_TRANSLATION_SVC;           /* CHAPTER 6       */
    .   .   END;
    .   END;
    END;

END CS.CONTACT_DISCONTACT_RSP;
```

```
CS.CONTACTED_PROC: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  THIS PROCEDURE CHECKS THE PERTINENT FSM TO SEE IF IT IS IN THE │
   │                APPROPRIATE STATE TO RECEIVE THE CONTACTED REQUEST.  IF SO, THE │
   │                REQUEST IS SENT TO THE FSM AND TO UPM_TRANSLATION_SVC AND A POSITIVE │
   │                RESPONSE TO CONTACTED IS GENERATED AND SENT TO SNS.SEND.  (IF THE │
   │                LINK IS SWITCHED, AN ACTPU REQUEST IS ALSO GENERATED.)  IN ADDITION, │
   │                WHEN CONTACTED(LOADED) IS THE INPUT AND THE FSM IS IN THE ACTIVE │
   │                STATE, THE LINK'S SAVE_MU_FOR_RETRY_LIST IS CHECKED TO SEE IF IT │
   │                CONTAINS ANY ELEMENTS; IF SO, ALL ARE REMOVED AND SENT TO CS.SEND │
   │                (PAGE 7-48).                                              │
   │                                                                          │
   │                IF THE FSM IS NOT IN THE APPROPRIATE STATE TO RECEIVE THE INPUT, THE │
   │                INPUT IS SENT TO UPM_LOG AND -RSP(CONTACTED) IS SENT TO SNS.SEND. │
   │                                                                          │
   │     INPUT:     CONTACTED FROM SNS.RCV (CHAPTER 6)                        │
   │                                                                          │
   │     OUTPUT:    CONTACTED TO UPM_TRANSLATION_SVC (CHAPTER 6) AND TO THE APPROPRIATE │
   │                FSM, +RSP(CONTACTED) TO SNS.SEND (CHAPTER 6), AND, IF THE LINK IS │
   │                SWITCHED, AN ACTPU TO CS.PU_PROC (PAGE 7-52); OR CONTACTED TO │
   │                UPM_LOG (APPENDIX B) AND -RSP(CONTACTED) TO SNS.SEND      │
   │                                                                          │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
   │                SSCP.SVC_MGR.CS.RCV              PAGE 7-50                │
   │                                                                          │
   │     REFERS TO THE FOLLOWING PROCEDURE(S):                                │
   │                CS.PU_PROC                       PAGE 7-52                │
   │                FSM_ALS_CONTACT_DOM_RES          PAGE 7-130               │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL TARGET_NA BIT(48);
  DCL LIST_PTR POINTER;

  TARGET_NA = OSAF||(NSC_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
                                                                  /* APPENDIX A    */
  DRCB_PTR = FIND_ALS_FOR_DOM_RES(TARGET_NA);                     /* APPENDIX B    */

  IF DRCB.RESOURCE_CATEGORY ¬= ALS THEN
    DO;
    . CALL UPM_LOG;                                               /* APPENDIX B    */
    . CALL CHANGE_MU_TO_NEG_RSP(0806);     /* APPENDIX B, RESOURCE UNKNOWN       */
    . SEND MU TO SNS.SEND;                                        /* CHAPTER 6     */
    END;

  ELSE
    DO;
    . IF FSM_ALS_CONTACT_DOM_RES ¬= (PEND_ACTIVE_CONTACTED | PEND_RESET_CONTACTED) THEN
    .                                                             /* PAGE 7-130    */
    .   DO;
    .   . CALL UPM_LOG;                                           /* APPENDIX B    */
    .   . CALL CHANGE_MU_TO_NEG_RSP(0809);  /* APPENDIX B, MODE INCONSISTENCY     */
    .   . SEND MU TO SNS.SEND;                                    /* CHAPTER 6     */
    .   END;
    .
    . ELSE
    .   DO;
    .   . CALL FSM_ALS_CONTACT_DOM_RES;                          /* PAGE 7-130    */
    .   .
    .   . IF FSM_ALS_CONTACT_DOM_RES = ACTIVE &                  /* PAGE 7-130    */
    .   .      CONTACTED_RQ.STATUS = LOADED THEN
    .   .   DO WHILE( ¬EMPTY(DRCB.SAVE_MU_FOR_RETRY_LIST));
    .   .   . LIST_PTR = FIRST_ENTRY(DRCB.SAVE_MU_FOR_RETRY_LIST);
    .   .   . REMOVE LIST_PTR->MU FROM DRCB.SAVE_MU_FOR_RETRY_LIST;
    .   .   . SEND LIST_PTR->MU TO SSCP.SVC_MGR.CS.SEND;         /* PAGE 7-48     */
    .   .   END;
    .   .
    .   . SEND MU TO UPM_TRANSLATION_SVC;                        /* CHAPTER 6     */
    .   . MU_PTR = UPM_CREATE_RSP('CONTACTED');                  /* APPENDIX B    */
    .   . RTI = POSITIVE;                                        /* APPENDIX B    */
    .   . SEND MU TO SNS.SEND;                                   /* CHAPTER 6     */
    .   .
    .   . IF DRCB.SWITCHED_LINK = SWITCHED THEN
    .   .   DO;
    .   .   . MU_PTR = UPM_CREATE_RQ('ACTPU');                   /* APPENDIX B    */
    .   .   . ACTPU_RQ.SSCP_ID = NCB.SSCP_ID;
    .   .   . DSAF = OSAF;
    .   .   . CALL CS.PU_PROC;                                   /* PAGE 7-52     */
    .   .   END;
    .   END;
    END;

END CS.CONTACTED_PROC;
```

```
CS.LOAD_PROC: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE HANDLES THE LOADING OF REMOTE PU_T4 NODES. REQUESTS │
│                 THAT HAVE TARGETS THAT ARE NOT SECONDARY ALS'S ARE REJECTED.      │
│                                                                          │
│                 WHEN IPLINIT IS THE INPUT, THE TARGET ALS'S ASSOCIATED LINK IS   │
│                 CHECKED TO SEE IF IT IS ACTIVE. IF THE LINK FSM IS NOT ACTIVE, THE │
│                 PROCEDURE RESOURCE_ACTIVE_CHECK, WHICH PERFORMS THE CHECKING,    │
│                 INSERTS THE IPLINIT REQUEST INTO THE LINK'S SAVE_MU_FOR_RETRY_LIST. │
│                 IF THE LINK FSM IS ACTIVE, THE FSM'S FOR DUMP, IPL, AND RPO ARE  │
│                 CHECKED TO SEE IF THESE PROCEDURES ARE IN INTERRUPTIBLE STATES. IF │
│                 NOT, THE REQUEST IS REJECTED. IF THE PROCEDURES ARE ALL          │
│                 INTERRUPTIBLE, THEN THE REQUEST IS SENT TO SNS.SEND, AND THE IPL FSM │
│                 IS UPDATED TO INDICATE THAT AN INITIAL PROGRAM LOAD IS BEGINNING. │
│                                                                          │
│                 IF IPLTEXT OR IPLFINAL IS THE INPUT AND THE IPL FSM DOES NOT │
│                 INDICATE IPL IN PROGRESS, THE REQUEST IS REJECTED; OTHERWISE, THE │
│                 REQUEST IS SENT TO THE FSM AND TO SNS.SEND.              │
│                                                                          │
│      INPUT:     IPLINIT, IPLTEXT, OR IPLFINAL FROM UPM_TRANSLATION_SVC (CHAPTER 6) │
│                                                                          │
│      OUTPUT:    IPLINIT, IPLTEXT, OR IPLFINAL TO SNS.SEND (CHAPTER 6) AND TO THE FSM │
│                 AND A COPY OF THE TARGET ADDRESS TO UPM_SAVE_TARGET_NA (PAGE 7-122), │
│                 OR A SEND_CHECK WITH AN APPROPRIATE ERROR CODE TO        │
│                 UPM_TRANSLATION_SVC (CHAPTER 6)                          │
│                                                                          │
│      NOTE:      PROCESSING OF THIS REQUEST RESUMES IN CS.LOAD_DUMP_RPO_RSP (PAGE │
│                 7-84) WHEN SNS RETURNS A RESPONSE.                       │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 SSCP.SVC_MGR.CS.SEND              PAGE 7-48              │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 FSM_ALS_CONNECTED_DOM_RES         PAGE 7-133             │
│                 FSM_ALS_IPL_DOM_RES               PAGE 7-131             │
│                 RESOURCE_ACTIVE_CHECK             PAGE 7-116             │
│                 SEC_ALS_SUBTREE_INTERRUPT         PAGE 7-120             │
│                 UPM_SAVE_TARGET_NA                PAGE 7-122             │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL TARGET_NA BIT(48);
  DCL ALS_NA BIT(48);
  DCL LINK_NA BIT(48);

  TARGET_NA = DSAF||(NSC_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
                                                        /* APPENDIX A      */
  DRCB_PTR = FIND_ALS_FOR_DOM_RES(TARGET_NA);           /* APPENDIX B      */

  IF DRCB.RESOURCE_CATEGORY ¬= ALS THEN
    SEND SEND_CHECK(X'0806') TO UPM_TRANSLATION_SVC;    /* RESOURCE UNKNOWN */

  ELSE
    DO;
    . ALS_NA = DRCB.NETWORK_ADDRESS;
    .
    . IF DRCB.LINK_DLC_ROLE ¬= SECONDARY THEN
    .   SEND SEND_CHECK(X'0849') TO UPM_TRANSLATION_SVC;  /* INVALID REQUESTED PROC */
    .
    . ELSE
    .   IF DRCB.SWITCHED_LINK = SWITCHED &
    .        FSM_ALS_CONNECTED_DOM_RES ¬= ACTIVE THEN     /* PAGE 7-133      */
    .     SEND SEND_CHECK(X'0801') TO UPM_TRANSLATION_SVC; /* RESOURCE NOT AVAILABLE */
    .
    .   ELSE
    .
    .     SELECT ANYORDER(NS_RQ_CODE);
```

```
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │                IPLINIT                   │
                    └──────────────────────────────────────────┘
                                                                        */
.     . WHEN(IPLINIT)
.     .   DO;
.     .   . LINK_NA = DRCB.ASSOCIATED_RES_PTR->DRCB.NETWORK_ADDRESS;
.     .   . IF RESOURCE_ACTIVE_CHECK(LINK_NA,LINK) = OK THEN /* PAGE 7-116  */
.     .   .   IF SEC_ALS_SUBTREE_INTERRUPT(ALS_NA) = OK THEN /* PAGE 7-120  */
.     .   .     DO;
.     .   .     . CALL FSM_ALS_IPL_DOM_RES;                 /* PAGE 7-131  */
.     .   .     . CALL UPM_SAVE_TARGET_NA(TARGET_NA);       /* PAGE 7-122  */
.     .   .     . SEND MU TO SNS.SEND;                      /* CHAPTER 6   */
.     .   .     END;
.     .   .
.     .   .   ELSE
.     .   .     SEND SEND_CHECK(X'0818') TO UPM_TRANSLATION_SVC;
.     .   .                                                 /* LINK PROC IN PROGRESS */
.     .   END;
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │           IPLTEXT OR IPLFINAL            │
                    └──────────────────────────────────────────┘
                                                                        */
.     . WHEN(IPLTEXT,IPLFINAL)
.     .   DO;
.     .   . IF FSM_ALS_IPL_DOM_RES = INIPL THEN             /* PAGE 7-131  */
.     .   .   DO;
.     .   .   . CALL FSM_ALS_IPL_DOM_RES;                   /* PAGE 7-131  */
.     .   .   . CALL UPM_SAVE_TARGET_NA(TARGET_NA);         /* PAGE 7-122  */
.     .   .   . SEND MU TO SNS.SEND;                        /* CHAPTER 6   */
.     .   .   END;
.     .   .
.     .   . ELSE
.     .   .   SEND SEND_CHECK(X'081A') TO UPM_TRANSLATION_SVC;
.     .   .                                                 /* RQ SEQUENCE ERROR */
.     .   END;
.     END;
    END;

END CS.LOAD_PROC;
```

```
CS.DUMP_PROC: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE HANDLES  THE DUMPING OF REMOTE  NODES.  REQUESTS THAT │
│                 HAVE TARGETS THAT ARE NOT SECONDARY ALS'S ARE REJECTED.   │
│                                                                          │
│                 WHEN DUMPINIT  IS THE  INPUT, THE  TARGET ALS'S  ASSOCIATED LINK  IS │
│                 CHECKED TO SEE IF IT IS ACTIVE.  IF  THE LINK FSM IS NOT ACTIVE, THE │
│                 PROCEDURE  RESOURCE_ACTIVE_CHECK,   WHICH  PERFORMS   THE  CHECKING, │
│                 INSERTS THE DUMPINIT REQUEST INTO THE LINK'S SAVE_MU_FOR_RETRY_LIST. │
│                 IF THE  LINK FSM  IS ACTIVE, THE  FSM'S FOR DUMP,  IPL, AND  RPO ARE │
│                 CHECKED TO SEE IF THESE PROCEDURES  ARE IN INTERRUPTIBLE STATES.  IF │
│                 NOT,  THE   REQUEST  IS  REJECTED.    IF  THE  PROCEDURES   ARE  ALL │
│                 INTERRUPTIBLE, THE REQUEST IS SENT TO  SNS.SEND, AND THE DUMP FSM IS │
│                 UPDATED TO INDICATE THAT A DUMP IS BEGINNING.            │
│                                                                          │
│                 IF DUMPTEXT  OR DUMPFINAL  IS THE INPUT  AND THE  DUMP FSM  DOES NOT │
│                 INDICATE DUMP IN PROGRESS, THEN  THE REQUEST IS REJECTED; OTHERWISE, │
│                 THE REQUEST IS SENT TO THE FSM AND TO SNS.SEND.          │
│                                                                          │
│      INPUT:     DUMPINIT, DUMPTEXT,  OR DUMPFINAL FROM  UPM_TRANSLATION_SVC (CHAPTER │
│                 6)                                                       │
│                                                                          │
│      OUTPUT:    DUMPINIT, DUMPTEXT, OR DUMPFINAL TO SNS.SEND  (CHAPTER 6) AND TO THE │
│                 FSM AND  A COPY  OF THE TARGET  ADDRESS TO  UPM_SAVE_TARGET_NA (PAGE │
│                 7-122),  OR  A   SEND_CHECK  WITH  AN  APPROPRIATE   ERROR  CODE  TO │
│                 UPM_TRANSLATION_SVC (CHAPTER 6)                          │
│                                                                          │
│      NOTE:      PROCESSING  OF THIS  REQUEST RESUMES  IN CS.LOAD_DUMP_RPO_RSP  (PAGE │
│                 7-84) WHEN SNS RETURNS A RESPONSE.                       │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 SSCP.SVC_MGR.CS.SEND              PAGE 7-48              │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 FSM_ALS_CONNECTED_DOM_RES         PAGE 7-133             │
│                 FSM_ALS_DUMP_DOM_RES              PAGE 7-131             │
│                 RESOURCE_ACTIVE_CHECK             PAGE 7-116             │
│                 SEC_ALS_SUBTREE_INTERRUPT         PAGE 7-120             │
│                 UPM_SAVE_TARGET_NA                PAGE 7-122             │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL TARGET_NA BIT(48);
  DCL ALS_NA BIT(48);
  DCL LINK_NA BIT(48);

  TARGET_NA = DSAF||(NSC_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
                                                    /* APPENDIX A          */
  DRCB_PTR = FIND_ALS_FOR_DOM_RES(TARGET_NA);       /* APPENDIX B          */

  IF DRCB.RESOURCE_CATEGORY ¬= ALS THEN
    SEND SEND_CHECK(X'0806') TO UPM_TRANSLATION_SVC;  /* RESOURCE UNKNOWN  */

  ELSE
    DO;
    . ALS_NA = DRCB.NETWORK_ADDRESS;
    .
    . IF DRCB.LINK_DLC_ROLE ¬= SECONDARY THEN
    .   SEND SEND_CHECK(X'0849') TO UPM_TRANSLATION_SVC;  /* INVALID REQUESTED PROC */
    .
    . ELSE
    .   IF DRCB.SWITCHED_LINK = SWITCHED &
    .        FSM_ALS_CONNECTED_DOM_RES ¬= ACTIVE THEN     /* PAGE 7-133     */
    .     SEND SEND_CHECK(X'0801') TO UPM_TRANSLATION_SVC;  /* RESOURCE NOT AVAILABLE */
    .
    .   ELSE
    .
    .     SELECT ANYORDER(NS_RQ_CODE);
```

```
                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │                DUMPINIT                  │
                    └──────────────────────────────────────────┘
                                                                                */
      . WHEN(DUMPINIT)
      .   DO;
      .   . LINK_NA = DRCB.ASSOCIATED_RES_PTR->DRCB.NETWORK_ADDRESS;
      .   . IF RESOURCE_ACTIVE_CHECK(LINK_NA,LINK) = OK THEN /* PAGE 7-116      */
      .   .   IF SEC_ALS_SUBTREE_INTERRUPT(ALS_NA) = OK THEN /* PAGE 7-120      */
      .   .     DO;
      .   .     . CALL FSM_ALS_DUMP_DOM_RES;                /* PAGE 7-131      */
      .   .     . CALL UPM_SAVE_TARGET_NA(TARGET_NA);       /* PAGE 7-122      */
      .   .     . SEND MU TO SNS.SEND;                      /* CHAPTER 6       */
      .   .     END;
      .   .
      .   .   ELSE
      .   .     SEND SEND_CHECK(X'0818') TO UPM_TRANSLATION_SVC;
      .   .                                                 /* LINK PROC IN PROGRESS */
      .   . END;
                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │          DUMPTEXT OR DUMPFINAL           │
                    └──────────────────────────────────────────┘
                                                                                */
      . WHEN(DUMPTEXT,DUMPFINAL)
      .   DO;
      .   . IF FSM_ALS_DUMP_DOM_RES = INDUMP THEN           /* PAGE 7-131      */
      .   .   DO;
      .   .   . CALL FSM_ALS_DUMP_DOM_RES;                  /* PAGE 7-131      */
      .   .   . CALL UPM_SAVE_TARGET_NA(TARGET_NA);         /* PAGE 7-122      */
      .   .   . SEND MU TO SNS.SEND;                        /* CHAPTER 6       */
      .   .   END;
      .   .
      .   . ELSE
      .   .   SEND SEND_CHECK(X'081A') TO UPM_TRANSLATION_SVC;
      .   .                                                 /* RQ SEQUENCE ERROR */
      .   . END;
      . END;
    END;

END CS.DUMP_PROC;
```

This page
intentionally
left blank

```
CS.RPO_PROC: PROCEDURE;

/*
┌──────────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE HANDLES  THE POWERING-OFF OF REMOTE  NODES.  REQUESTS │
│                 THAT HAVE TARGETS THAT ARE NOT SECONDARY ALS'S ARE REJECTED.  │
│                                                                              │
│                 UPON RECEIPT OF AN RPO REQUEST,  THE TARGET ALS'S ASSOCIATED LINK IS │
│                 CHECKED TO SEE IF IT IS ACTIVE.  IF  THE LINK FSM IS NOT ACTIVE, THE │
│                 PROCEDURE  RESOURCE_ACTIVE_CHECK,   WHICH  PERFORMS   THE  CHECKING, │
│                 INSERTS THE RPO REQUEST INTO THE LINK'S SAVE_MU_FOR_RETRY_LIST. │
│                                                                              │
│                 IF THE LINK FSM IS ACTIVE, THE FSM'S FOR CONTACT, IPL, DUMP, AND RPO │
│                 ARE CHECKED.  IF  ALL ARE RESET, THE RPO  IS SENT TO THE  FSM AND TO │
│                 UPM_TRANSLATION_SVC.  IF ANY OF THE CHECKED  FSM'S IS NOT RESET, THE │
│                 RPO IS REJECTED.                                             │
│                                                                              │
│      INPUT:     RPO FROM UPM_TRANSLATION_SVC (CHAPTER 6)                     │
│                                                                              │
│      OUTPUT:    RPO TO SNS.SEND (CHAPTER 6) AND TO THE  FSM AND A COPY OF THE TARGET │
│                 ADDRESS TO UPM_SAVE_TARGET_NA (PAGE 7-122),  OR A SEND_CHECK WITH AN │
│                 APPROPRIATE ERROR CODE TO UPM_TRANSLATION_SVC (CHAPTER 6)    │
│                                                                              │
│      NOTE:      PROCESSING  OF THIS  REQUEST RESUMES  IN CS.LOAD_DUMP_RPO_RSP  (PAGE │
│                 7-84) WHEN SNS RETURNS A RESPONSE.                           │
│                                                                              │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
│                 SSCP.SVC_MGR.CS.SEND                PAGE 7-48                │
│                                                                              │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
│                 FSM_ALS_CONNECTED_DOM_RES           PAGE 7-133               │
│                 FSM_ALS_RPO_DOM_RES                 PAGE 7-132               │
│                 RESOURCE_ACTIVE_CHECK               PAGE 7-116               │
│                 SEC_ALS_SUBTREE_CHECK               PAGE 7-121               │
│                 UPM_SAVE_TARGET_NA                  PAGE 7-122               │
└──────────────────────────────────────────────────────────────────────────────┘
*/

DCL TARGET_NA BIT(48);
DCL ALS_NA BIT(48);
DCL LINK_NA BIT(48);

TARGET_NA = DSAF||(NSC_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
                                                          /* APPENDIX A          */
DRCB_PTR = FIND_ALS_FOR_DOM_RES(TARGET_NA);               /* APPENDIX B          */

IF DRCB.RESOURCE_CATEGORY ¬= ALS THEN
  SEND SEND_CHECK(X'0806') TO UPM_TRANSLATION_SVC;        /* RESOURCE UNKNOWN    */

ELSE
  DO;
  . ALS_NA = DRCB.NETWORK_ADDRESS;
  .
  . IF DRCB.LINK_DLC_ROLE ¬= SECONDARY THEN
  .   SEND SEND_CHECK(X'0849') TO UPM_TRANSLATION_SVC;   /* INVALID REQUESTED PROC */
  .
  . ELSE
  .   IF DRCB.SWITCHED_LINK = SWITCHED &
  .        FSM_ALS_CONNECTED_DOM_RES ¬= ACTIVE THEN      /* PAGE 7-133          */
  .     SEND SEND_CHECK(X'0801') TO UPM_TRANSLATION_SVC; /* RESOURCE NOT AVAILABLE */
  .
  .   ELSE
  .     DO;
  .     . LINK_NA = DRCB.ASSOCIATED_RES_PTR->DRCB.NETWORK_ADDRESS;
  .     .
  .     . IF RESOURCE_ACTIVE_CHECK(LINK_NA,LINK) = OK THEN  /* PAGE 7-116        */
  .     .   IF SEC_ALS_SUBTREE_CHECK(ALS_NA) = NG THEN      /* PAGE 7-121        */
  .     .     SEND SEND_CHECK(X'0834') TO UPM_TRANSLATION_SVC; /* RPO NOT INITIATED */
  .     .
  .     .   ELSE
  .     .     DO;
  .     .     . CALL UPM_SAVE_TARGET_NA(TARGET_NA);       /* PAGE 7-122        */
  .     .     . CALL FSM_ALS_RPO_DOM_RES;                 /* PAGE 7-132        */
  .     .     . SEND MU TO SNS.SEND;                      /* CHAPTER 6         */
  .     .     END;
  .     END;
  END;

END CS.RPO_PROC;
```

```
CS.LOAD_DUMP_RPO_RSP: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE CHECKS THE PERTINENT FSM TO SEE IF IT IS IN THE  │
│                 APPROPRIATE STATE TO RECEIVE THE INPUT RESPONSE.  IF SO, THE   │
│                 RESPONSE IS SENT TO THE FSM AND TO UPM_TRANSLATION_SVC.        │
│                                                                          │
│                 IF THE FSM IS NOT IN THE APPROPRIATE STATE TO RECEIVE THE INPUT, THE │
│                 INPUT IS SENT TO UPM_LOG.                                │
│                                                                          │
│      INPUT:     ±RSP(IPLINIT|IPLTEXT|IPLFINAL) OR ±RSP(DUMPINIT|DUMPTEXT|DUMPFINAL) │
│                 OR ±RSP(RPO), FROM SNS.RCV (CHAPTER 6) AND A COPY OF THE TARGET │
│                 ADDRESS FROM UPM_RETRIEVE_TARGET_NA (PAGE 7-122)         │
│                                                                          │
│      OUTPUT:    THE INPUT RESPONSE, IF VALID, IS SENT TO UPM_TRANSLATION_SVC │
│                 (CHAPTER 6) AND TO THE APPROPRIATE FSM; OTHERWISE, THE RESPONSE IS │
│                 SENT TO UPM_LOG (APPENDIX B).                            │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 SSCP.SVC_MGR.CS.RCV              PAGE 7-50               │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 FSM_ALS_DUMP_DOM_RES             PAGE 7-131              │
│                 FSM_ALS_IPL_DOM_RES              PAGE 7-131              │
│                 FSM_ALS_RPO_DOM_RES              PAGE 7-132              │
│                 UPM_RETRIEVE_TARGET_NA           PAGE 7-122              │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL TARGET_NA BIT(48);
  DCL LINK_NA BIT(48);

  TARGET_NA = UPM_RETRIEVE_TARGET_NA;                    /* PAGE 7-122    */
  DRCB_PTR = FIND_ALS_FOR_DOM_RES(TARGET_NA);            /* APPENDIX B    */

  IF DRCB.RESOURCE_CATEGORY ¬= ALS THEN
    DO;
    . CALL UPM_LOG;                                      /* APPENDIX B    */
    . DISCARD MU;
    END;

  ELSE

    SELECT ANYORDER(NS_RQ_CODE);
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │    POSITIVE OR NEGATIVE RESPONSE TO IPLINIT    │
                  └──────────────────────────────────────────┘
                                                                        */
    . WHEN(IPLINIT)
    .   DO;
    .   . IF FSM_ALS_IPL_DOM_RES = PEND_INIPL THEN       /* PAGE 7-131    */
    .   .   DO;
    .   .   . CALL FSM_ALS_IPL_DOM_RES;                  /* PAGE 7-131    */
    .   .   . SEND MU TO UPM_TRANSLATION_SVC;            /* CHAPTER 6     */
    .   .   END;
    .   . ELSE
    .   .   DO;
    .   .   . CALL UPM_LOG;                              /* APPENDIX B    */
    .   .   . DISCARD MU;
    .   .   END;
    .   END;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │    POSITIVE OR NEGATIVE RESPONSE TO IPLTEXT    │
                  └──────────────────────────────────────────┘
                                                                        */
    . WHEN(IPLTEXT)
    .   DO;
    .   . IF FSM_ALS_IPL_DOM_RES = (PEND_INIPL | PEND_INIPL_TEXT) THEN
    .   .                                                /* PAGE 7-131    */
    .   .   DO;
    .   .   . CALL FSM_ALS_IPL_DOM_RES;                  /* PAGE 7-131    */
    .   .   . SEND MU TO UPM_TRANSLATION_SVC;            /* CHAPTER 6     */
    .   .   END;
    .   . ELSE
    .   .   DO;
    .   .   . CALL UPM_LOG;                              /* APPENDIX B    */
    .   .   . DISCARD MU;
    .   .   END;
    .   END;
```

```
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │  POSITIVE OR NEGATIVE RESPONSE TO IPLFINAL  │
                    └──────────────────────────────────────────┘
                                                                        */
    . WHEN(IPLFINAL)
    .   DO;
    .   . IF FSM_ALS_IPL_DOM_RES = (PEND_INIPL | PEND_RESET) THEN
    .   .                                                   /* PAGE 7-131   */
    .   .   DO;
    .   .   . CALL FSM_ALS_IPL_DOM_RES;                     /* PAGE 7-131   */
    .   .   . SEND MU TO UPM_TRANSLATION_SVC;               /* CHAPTER 6    */
    .   .   END;
    .   . ELSE
    .   .   DO;
    .   .   . CALL UPM_LOG;                                 /* APPENDIX B   */
    .   .   . DISCARD MU;
    .   .   END;
    .   END;
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │  POSITIVE OR NEGATIVE RESPONSE TO DUMPINIT  │
                    └──────────────────────────────────────────┘
                                                                        */
    . WHEN(DUMPINIT)
    .   DO;
    .   . IF FSM_ALS_DUMP_DOM_RES = PEND_INDUMP THEN        /* PAGE 7-131   */
    .   .   DO;
    .   .   . CALL FSM_ALS_DUMP_DOM_RES;                    /* PAGE 7-131   */
    .   .   . SEND MU TO UPM_TRANSLATION_SVC;               /* CHAPTER 6    */
    .   .   END;
    .   . ELSE
    .   .   DO;
    .   .   . CALL UPM_LOG;                                 /* APPENDIX B   */
    .   .   . DISCARD MU;
    .   .   END;
    .   END;
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │  POSITIVE OR NEGATIVE RESPONSE TO DUMPTEXT  │
                    └──────────────────────────────────────────┘
                                                                        */
    . WHEN(DUMPTEXT)
    .   DO;
    .   . IF FSM_ALS_DUMP_DOM_RES = (PEND_INDUMP | PEND_INDUMP_TEXT) THEN
    .   .                                                   /* PAGE 7-131   */
    .   .   DO;
    .   .   . CALL FSM_ALS_DUMP_DOM_RES;                    /* PAGE 7-131   */
    .   .   . SEND MU TO UPM_TRANSLATION_SVC;               /* CHAPTER 6    */
    .   .   END;
    .   . ELSE
    .   .   DO;
    .   .   . CALL UPM_LOG;                                 /* APPENDIX B   */
    .   .   . DISCARD MU;
    .   .   END;
    .   END;
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │  POSITIVE OR NEGATIVE RESPONSE TO DUMPFINAL  │
                    └──────────────────────────────────────────┘
                                                                        */
    . WHEN(DUMPFINAL)
    .   DO;
    .   . IF FSM_ALS_DUMP_DOM_RES = (PEND_INDUMP | PEND_RESET) THEN
    .   .                                                   /* PAGE 7-131   */
    .   .   DO;
    .   .   . CALL FSM_ALS_DUMP_DOM_RES;                    /* PAGE 7-131   */
    .   .   . SEND MU TO UPM_TRANSLATION_SVC;               /* CHAPTER 6    */
    .   .   END;
    .   . ELSE
    .   .   DO;
    .   .   . CALL UPM_LOG;                                 /* APPENDIX B   */
    .   .   . DISCARD MU;
    .   .   END;
    .   END;
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │      POSITIVE OR NEGATIVE RESPONSE TO RPO      │
                    └──────────────────────────────────────────┘
                                                                        */
    . WHEN(RPO)
    .   DO;
    .   . IF FSM_ALS_RPO_DOM_RES = PEND THEN                /* PAGE 7-132   */
    .   .   DO;
    .   .   . CALL FSM_ALS_RPO_DOM_RES;                     /* PAGE 7-132   */
    .   .   . SEND MU TO UPM_TRANSLATION_SVC;               /* CHAPTER 6    */
    .   .   END;
    .   . ELSE
    .   .   DO;
    .   .   . CALL UPM_LOG;                                 /* APPENDIX B   */
    .   .   . DISCARD MU;
    .   .   END;
    .   END;
    . END;

END CS.LOAD_DUMP_RPO_RSP;
```

```
CS.LDREQD_PROC: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE PROCESSES LDREQD REQUESTS.            │
  │                                                                      │
  │                 IF THE PU REQUESTING THE LOAD IS NOT A PU_T2, A NEGATIVE RESPONSE IS
  │                 SENT TO  THE REQUESTING  PU.  IF  THE PU  REQUESTING THE  LOAD IS  A
  │                 PU_T2, FSM_PU_ACT_DOM_RES IS  CHECKED TO SEE IF IT IS  IN THE ACTIVE
  │                 STATE.  IF  THE FSM IS  IN THE ACTIVE STATE,  THE LDREQD IS  SENT TO
  │                 CS.INITIATE_IPL_PROC (PAGE 7-91).   IF THE FSM IS NOT  IN THE ACTIVE
  │                 STATE, A NEGATIVE RESPONSE IS SENT TO THE REQUESTING PU.
  │                                                                      │
  │      INPUT:     LDREQD FROM SNS.RCV (CHAPTER 6)                      │
  │                                                                      │
  │      OUTPUT:    -RSP(LDREQD)    TO   SNS.SEND    (CHAPTER   6),    OR   LDREQD    TO
  │                 CS.INITIATE_IPL_PROC (PAGE 7-91)
  │                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
  │                 SSCP.SVC_MGR.CS.RCV                PAGE 7-50         │
  │                                                                      │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
  │                 CS.INITIATE_IPL_PROC               PAGE 7-91         │
  │                 FSM_PU_ACT_DOM_RES                 PAGE 7-128        │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL TARGET_NA BIT(48);

  TARGET_NA = OSAF||OEF;
  DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);                    /* APPENDIX B          */

  IF DRCB.RESOURCE_CATEGORY = ALS THEN
    DRCB_PTR = FIND_SUBORDINATE_DOM_RES(TARGET_NA);

  IF DRCB_PTR = NULL THEN
    DO;
    . CALL UPM_LOG;                                              /* APPENDIX B          */
    . CALL CHANGE_MU_TO_NEG_RSP(X'0806');  /* APPENDIX B, RESOURCE UNKNOWN              */
    . SEND MU TO SNS.SEND;                                       /* CHAPTER 6           */
    END;

  ELSE
    IF DRCB.RESOURCE_CATEGORY = PERIPHERAL_PU &
         DRCB.PERIPHERAL_PU_TYPE = PU_T2 THEN

      IF FSM_PU_ACT_DOM_RES ¬= ACTIVE THEN                       /* PAGE 7-128          */
        DO;
        . CALL UPM_LOG;                                          /* APPENDIX B          */
        . CALL CHANGE_MU_TO_NEG_RSP(X'0809');  /* MODE INCONSISTENCY, APPENDIX B        */
        . SEND MU TO SNS.SEND;                                   /* CHAPTER 6           */
        END;

      ELSE
        DO;
        . CALL CHANGE_MU_TO_POS_RSP;                             /* APPENDIX B          */
        . SEND MU TO SNS.SEND;                                   /* CHAPTER 6           */
        . CALL CS.INITIATE_IPL_PROC(TARGET_NA,LDREQD_RQ.ADJ_PU_LOAD_CAPABILITY);
        .                                                        /* PAGE 7-91           */
        END;

    ELSE
      DO;
      . CALL UPM_LOG;                                            /* APPENDIX B          */
      . CALL CHANGE_MU_TO_NEG_RSP(X'0849');  /* INVALID REQUESTED RESOURCE, APPENDIX B    */
      . SEND MU TO SNS.SEND;                                     /* CHAPTER 6           */
      END;

END CS.LDREQD_PROC;
```

```
CS.INITPROC_PROC: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE  HANDLES THE INITIATION  OF AN NC_IPL  PROCEDURE THAT
 │                 WILL LOAD A PERIPHERAL PU.
 │
 │                 THE TARGET PERIPHERAL PU'S ASSOCIATED ALS IS CHECKED TO SEE IF IT IS
 │                 ACTIVE.    IF   THE  ALS   FSM   IS   NOT  ACTIVE,   THE   PROCEDURE
 │                 RESOURCE_ACTIVE_CHECK,  WHICH  PERFORMS THE  CHECKING,  INSERTS  THE
 │                 INITPROC REQUEST INTO THE ALS'S SAVE_MU_FOR_RETRY_LIST.
 │
 │                 IF  THE ALS  FSM  IS ACTIVE  AND  THE INITPROC  FSM  IS RESET,  THIS
 │                 PROCEDURE SENDS THE INITPROC REQUEST TO SNS.SEND AND TO THE FSM.  IF
 │                 THE FSM IS NOT RESET, THIS PROCEDURE GENERATES A SEND_CHECK WHICH IS
 │                 SENT TO UPM_TRANSLATION_SVC.
 │
 │      INPUT:     INITPROC FROM CS.INITIATE_IPL_PROC (PAGE 7-91)
 │
 │      OUTPUT:    INITPROC TO SNS.SEND (CHAPTER 6) AND TO  THE INITPROC FSM AND A COPY
 │                 OF  THE TARGET  ADDRESS  TO UPM_SAVE_TARGET_NA  (PAGE  7-122), OR  A
 │                 SEND_CHECK  WITH AN  APPROPRIATE ERROR  CODE TO  UPM_TRANSLATION_SVC
 │                 (CHAPTER 6)
 │
 │      NOTE:      PROCESSING OF  THIS REQUEST RESUMES  IN CS.INITPROC_RSP  (PAGE 7-88)
 │                 WHEN SNS RETURNS A RESPONSE
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 CS.INITIATE_IPL_PROC                PAGE 7-91
 │                 SSCP.SVC_MGR.CS.SEND                PAGE 7-48
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 FSM_PROC_DOM_RES                    PAGE 7-132
 │                 RESOURCE_ACTIVE_CHECK               PAGE 7-116
 │                 UPM_SAVE_TARGET_NA                  PAGE 7-122
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL TARGET_NA BIT(48);
  DCL PERIPHERAL_PU_NA BIT(48);
  DCL ALS_NA BIT(48);
  DCL ALS_PTR POINTER;

  TARGET_NA = DSAF||(NSC_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
                                                          /* APPENDIX A      */
  DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);             /* APPENDIX B      */

  IF DRCB.RESOURCE_CATEGORY = ALS THEN
    DRCB_PTR = FIND_SUBORDINATE_DOM_RES(DRCB.NETWORK_ADDRESS);  /* APPENDIX B  */

  IF (DRCB_PTR = NULL |
       DRCB.RESOURCE_CATEGORY ¬= PERIPHERAL_PU |
       DRCB.PERIPHERAL_PU_TYPE ¬= PU_T2) THEN
    SEND SEND_CHECK(X'0806') TO UPM_TRANSLATION_SVC;      /* RESOURCE UNKNOWN */

  ELSE
    IF INITPROC_RQ.PROCEDURE_TYPE ¬= LOAD THEN
      SEND SEND_CHECK(X'080C') TO UPM_TRANSLATION_SVC;    /* PROC NOT SUPPORTED */

    ELSE
      DO;
      . PERIPHERAL_PU_NA = DRCB.NETWORK_ADDRESS;
      . ALS_PTR = FIND_ALS_FOR_DOM_RES(PERIPHERAL_PU_NA);  /* APPENDIX B      */
      . ALS_NA = ALS_PTR->DRCB.NETWORK_ADDRESS;
      .
      . IF RESOURCE_ACTIVE_CHECK(ALS_NA,ALS) = OK THEN     /* PAGE 7-116      */
      .   IF FSM_PROC_DOM_RES = RESET THEN                 /* PAGE 7-132      */
      .     DO;
      .     . CALL UPM_SAVE_TARGET_NA(TARGET_NA);          /* PAGE 7-122      */
      .     . CALL FSM_PROC_DOM_RES;                       /* PAGE 7-132      */
      .     . SEND MU TO SNS.SEND;                         /* CHAPTER 6       */
      .     END;
      .
      .   ELSE
      .     SEND SEND_CHECK(X'0815') TO UPM_TRANSLATION_SVC;  /* FUNCTION ACTIVE */
      END;

END CS.INITPROC_PROC;
```

```
CS.INITPROC_RSP: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS  PROCEDURE CHECKS  THE INITPROC  FSM TO  SEE  IF IT  IS IN  THE │
│                 APPROPRIATE  STATE TO  RECEIVE THE  INITPROC RESPONSE.   IF SO,  THE │
│                 RESPONSE IS SENT TO THE FSM AND TO UPM_TRANSLATION_SVC.  IF NOT, THE │
│                 INPUT IS SENT TO UPM_LOG.  IF  THE RESPONSE TO INITPROC IS NEGATIVE, │
│                 THE SSCP ATTEMPTS TO INITIATE AN  SSCP-PU_T2 LOAD OPERATION.  IF THE │
│                 SSCP   CANNOT   LOAD   THE  PU_T2   NODE,   THIS   PROCEDURE   CALLS │
│                 CS.PU_T2_IPL_ABORT (PAGE 7-93).                          │
│                                                                          │
│      INPUT:     POSITIVE OR NEGATIVE  RESPONSE TO INITPROC FROM  SNS.RCV (CHAPTER 6) │
│                 AND A COPY  OF THE TARGET ADDRESS  FROM UPM_RETRIEVE_TARGET_NA (PAGE │
│                 7-122)                                                   │
│                                                                          │
│      OUTPUT:    ±RSP(INITPROC)  TO  UPM_TRANSLATION_SVC  (CHAPTER  6)  AND  TO  THE │
│                 APPROPRIATE FSM, OR ±RSP(INITPROC) TO  UPM_LOG (APPENDIX B); NETWORK │
│                 ADDRESS OF THE PU_T2 TO CS.INITIATE_IPL_PROC  (PAGE 7-91) OR A SENSE │
│                 CODE TO CS.PU_T2_IPL_ABORT (PAGE 7-93)                   │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 SSCP.SVC_MGR.CS.RCV              PAGE 7-50               │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 CS.INITIATE_IPL_PROC             PAGE 7-91               │
│                 CS.PU_T2_IPL_ABORT               PAGE 7-93               │
│                 FSM_PROC_DOM_RES                 PAGE 7-132              │
│                 UPM_CAN_SSCP_IPL_PU_T2           PAGE 7-126              │
│                 UPM_RETRIEVE_TARGET_NA           PAGE 7-122              │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL TARGET_NA BIT(48);

  TARGET_NA = UPM_RETRIEVE_TARGET_NA;                          /* PAGE 7-122          */
  DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);                  /* APPENDIX B          */

  IF DRCB.RESOURCE_CATEGORY = ALS THEN
    DRCB_PTR = FIND_SUBORDINATE_DOM_RES(DRCB.NETWORK_ADDRESS);  /* APPENDIX B          */

  IF (DRCB_PTR = NULL |
      DRCB.RESOURCE_CATEGORY ¬= PERIPHERAL_PU |
      DRCB.PERIPHERAL_PU_TYPE ¬= PU_T2) THEN
    DO;
    . CALL UPM_LOG;                                            /* APPENDIX B          */
    . DISCARD MU;
    END;

  ELSE
    DO;
    . IF FSM_PROC_DOM_RES = PEND_ACTIVE THEN                   /* PAGE 7-132          */
    .   DO;
    .   . CALL FSM_PROC_DOM_RES;                               /* PAGE 7-132          */
    .   . IF RTI = NEGATIVE THEN
    .   .
    .   .   IF UPM_CAN_SSCP_IPL_PU_T2 = YES THEN               /* PAGE 7-126          */
    .   .     CALL CS.INITIATE_IPL_PROC(TARGET_NA, ¬CAPABLE);  /* PAGE 7-91           */
    .   .   ELSE
    .   .     CALL CS.PU_T2_IPL_ABORT(X'084B');
    .   .                          /* PAGE 7-93, REQUESTED RESOURCE NOT AVAILABLE */
    .   . SEND MU TO UPM_TRANSLATION_SVC;                      /* CHAPTER 6           */
    .   END;
    .
    . ELSE
    .   DO;
    .   . CALL UPM_LOG;                                        /* APPENDIX B          */
    .   . DISCARD MU;
    .   END;
    END;

END CS.INITPROC_RSP;
```

```
CS.PROCSTAT_PROC: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  THIS PROCEDURE CHECKS THE PROCSTAT FSM TO SEE IF IT IS IN │
│                APPROPRIATE STATE TO RECEIVE THE PROCSTAT REQUEST.  IF NOT,│
│                INPUT IS SENT TO UPM_LOG.  IF SO, THE REQUEST IS SENT TO THE FSM AND
│                TO UPM_TRANSLATION_SVC.  IN ADDITION, THE PROCEDURE STATUS OF THE
│                PROCSTAT IS INSPECTED.  THE FOLLOWING VALUES OF PROCEDURE STATUS ARE
│                POSSIBLE:                                                 │
│                • PROCEDURE FAILURE                                       │
│                  IF LOAD WAS REQUESTED VIA THE ACTPU RESPONSE AND PROCEDURE FAILURE
│                  IS INDICATED, THE SSCP ATTEMPTS TO LOAD THE PU_T2 NODE.  IF THE
│                  SSCP CANNOT PERFORM THE LOAD, A DACTPU IS SENT TO CS.PU_PROC (PAGE
│                  7-52).  IF LOAD WAS REQUESTED VIA LDREQD, IT IS THE PU_T2'S
│                  RESPONSIBILITY TO REQUEST ANOTHER LOAD OR TO REQUEST DISCONTACT.
│                • IPL SUCCESSFUL                                          │
│                  IF THE IPL WAS SUCCESSFUL, FSM_PU_ACT_DOM_RES IS UPDATED.
│                                                                          │
│     INPUT:     PROCSTAT FROM SNS.RCV (CHAPTER 6)                         │
│                                                                          │
│     OUTPUT:    PROCSTAT TO UPM_TRANSLATION_SVC (CHAPTER 6) AND TO THE APPROPRIATE
│                FSM, AND, IF APPROPRIATE, DACTPU TO CS.PU_PROC (PAGE 7-52); OR
│                PROCSTAT TO UPM_LOG (APPENDIX B) AND -RSP(PROCSTAT) TO SNS.SEND
│                (CHAPTER 6)                                               │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                SSCP.SVC_MGR.CS.RCV                PAGE 7-50              │
│                                                                          │
│     REFERS TO THE FOLLOWING PROCEDURE(S):                                │
│                CS.INITIATE_IPL_PROC               PAGE 7-91              │
│                CS.PU_PROC                         PAGE 7-52              │
│                FSM_PROC_DOM_RES                   PAGE 7-132             │
│                FSM_PU_ACT_DOM_RES                 PAGE 7-128             │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL TARGET_NA BIT(48);
  DCL SAVE_MU_PTR POINTER;

  SAVE_MU_PTR = MU_PTR;
  TARGET_NA = OSAF||(NSC_RQ.TARGET_ADDRESS & NCE.NODE_ELEMENT_MASK);
                                                        /* APPENDIX A      */
  DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);           /* APPENDIX B      */

  IF DRCB.RESOURCE_CATEGORY = ALS THEN
    DRCB_PTR = FIND_SUBORDINATE_DOM_RES(DRCB.NETWORK_ADDRESS);  /* APPENDIX B */

  IF (DRCB_PTR = NULL |
      DRCB.RESOURCE_CATEGORY ¬= PERIPHERAL_PU |
      DRCB.PERIPHERAL_PU_TYPE ¬= PU_T2 |
      FSM_PROC_DOM_RES ¬= ACTIVE |
      FSM_PU_ACT_DOM_RES ¬= (ACTIVE | PEND_IPL)) THEN
    DO;
    . CALL UPM_LOG;                                     /* APPENDIX B      */
    . CALL CHANGE_MU_TO_NEG_RSP(0806);     /* APPENDIX B, RESOURCE UNKNOWN */
    . SEND MU TO SNS.SEND;                              /* CHAPTER 6       */
    END;

  ELSE

    SELECT ANYORDER(PROCSTAT_RQ.PROCEDURE_STATUS);
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │ PROCEDURE FAILURE:  SSCP WILL TRY TO IPL IF │
                  │ LOAD WAS REQUESTED VIA RESPONSE TO ACTPU.  │
                  └──────────────────────────────────────────┘
                                                                        */
    . WHEN(PROCEDURE_FAILURE)
    .   DO;
    .   . IF FSM_PU_ACT_DOM_RES = PEND_IPL THEN         /* PAGE 7-128      */
    .   .   CALL CS.INITIATE_IPL_PROC(TARGET_NA, ¬CAPABLE);  /* PAGE 7-91     */
    .   .
    .   . ELSE
    .   .   DO;
    .   .   . MU_PTR = UPM_CREATE_RQ('DACTPU');         /* APPENDIX B      */
    .   .   . CALL CS.PU_PROC;                          /* PAGE 7-52       */
    .   .   END;
    .   END;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │              IPL SUCCESSFUL              │
                  └──────────────────────────────────────────┘
                                                                        */
    . WHEN(IPL_SUCCESSFUL)
    .   CALL FSM_PU_ACT_DOM_RES;                        /* PAGE 7-128      */
    .
    END;

  MU_PTR = SAVE_MU_PTR;

  CALL FSM_PROC_DOM_RES;                                /* PAGE 7-132      */
  SEND MU TO UPM_TRANSLATION_SVC;                       /* CHAPTER 6       */

END CS.PROCSTAT_PROC;
```

This page
intentionally
left blank

```
CS.INITIATE_IPL_PROC: PROCEDURE(PU_T2_NA,ADJ_PU_LOAD_CAP);

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │     FUNCTION:  THIS PROCEDURE  DETERMINES WHETHER  THE TARGET PU_T2  NODE IS  TO BE  │
 │                LOADED BY THE SSCP OR BY THE SUBAREA PU ADJACENT TO THE PU_T2.       │
 │                                                                      │
 │                THE TYPE  OF LOAD  OPERATION IS  DETERMINED BY  THE CONTENTS  OF THE │
 │                ADJACENT PU LOAD CAPABILITY BIT (LOCATED  IN CONTROL VECTOR X'07' OF │
 │                THE  ACTPU  RESPONSE,  OR  IN  LDREQD).    IF  THE  BIT  IS  SET  TO │
 │                "NOT_CAPABLE"  (INDICATING THE  SUBAREA  PU  ADJACENT  TO  THE  PU_T2 │
 │                CANNOT    LOAD    THE    PU_T2    NODE),    THIS    PROCEDURE    CALLS │
 │                UPM_CAN_SSCP_IPL_PU_T2    (PAGE   7-126).     UPM_CAN_SSCP_IPL_PU_T2 │
 │                DETERMINES WHETHER  THE SSCP CAN LOAD  THE PU_T2 NODE.  IF  THE SSCP │
 │                CAN  LOAD  THE  PU_T2  NODE, THIS  PROCEDURE  SENDS  NS_IPL_INIT  TO │
 │                SNS.SEND (CHAPTER 6).  IF THE SSCP  CANNOT LOAD THE PU_T2 NODE, THIS │
 │                PROCEDURE CALLS CS.PU_T2_IPL_ABORT (PAGE 7-93).       │
 │                                                                      │
 │                IF  THE  ADJACENT  PU  LOAD  CAPABILITY  BIT  IS  SET  TO  "CAPABLE" │
 │                (INDICATING THE SUBAREA PU CAN LOAD  THE PU_T2 NODE), THIS PROCEDURE │
 │                SENDS INITPROC TO CS.INITPROC_PROC (PAGE 7-87).       │
 │                                                                      │
 │     INPUT:     THE NETWORK ADDRESS OF THE PU_T2 AND THE ADJACENT PU LOAD CAPABILITY │
 │                BIT ARE PASSED FROM CS.LDREQD_PROC  (PAGE 7-86) OR FROM CS.ACTPU_RSP │
 │                (PAGE  7-54)   OR  FROM   CS.PROCSTAT_PROC  (PAGE  7-89)  OR  FROM │
 │                CS.INITPROC_RSP (PAGE 7-88).                          │
 │                                                                      │
 │     OUTPUT:    NS_IPL_INIT TO SNS.SEND (CHAPTER 6)  OR INITPROC TO CS.INITPROC_PROC │
 │                (PAGE 7-87)  OR AN APPROPRIATE  SENSE CODE TO  PU_T2_IPL_ABORT (PAGE │
 │                7-93) OR -RSP TO SNS.SEND (CHAPTER 6)                 │
 │                                                                      │
 │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
 │                CS.ACTPU_RSP                    PAGE 7-54             │
 │                CS.INITPROC_RSP                 PAGE 7-88             │
 │                CS.LDREQD_PROC                  PAGE 7-86             │
 │                CS.PROCSTAT_PROC                PAGE 7-89             │
 │                                                                      │
 │     REFERS TO THE FOLLOWING PROCEDURE(S):                            │
 │                CS.INITPROC_PROC                PAGE 7-87             │
 │                CS.PU_T2_IPL_ABORT              PAGE 7-93             │
 │                FSM_PU_T2_IPL_DOM_RES           PAGE 7-133            │
 │                UPM_CAN_SSCP_IPL_PU_T2          PAGE 7-126            │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL PU_T2_NA BIT(48);
  DCL ADJ_PU_LOAD_CAP BIT(1);
  DCL SAVE_MU_PTR PTR;

  SAVE_MU_PTR = MU_PTR;

  IF FSM_PU_T2_IPL_DOM_RES ¬= RESET THEN                    /* PAGE 7-133    */
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'0809');                   /* APPENDIX B    */
    . SEND MU TO SNS.SEND;                                  /* CHAPTER 6     */
    END;

  ELSE
    IF ADJ_PU_LOAD_CAP = CAPABLE THEN
      DO;
      . MU_PTR = UPM_CREATE_RQ('INITPROC');                 /* APPENDIX B    */
      . CALL CS.INITPROC_PROC;                              /* PAGE 7-87     */
      END;

    ELSE                                  /* ADJ PU CANNOT LOAD THE PU_T2    */
      IF UPM_CAN_SSCP_IPL_PU_T2 = YES THEN                  /* PAGE 7-126    */
        DO;
        . MU_PTR = UPM_CREATE_RQ('NS_IPL_INIT');            /* APPENDIX B    */
        . CALL FSM_PU_T2_IPL_DOM_RES;                       /* PAGE 7-133    */
        . SEND MU TO SNS.SEND;                              /* CHAPTER 6     */
        END;

      ELSE
        CALL CS.PU_T2_IPL_ABORT(X'084B');  /* REQUESTED RESOURCE NOT AVAILABLE */

  MU_PTR = SAVE_MU_PTR;

END CS.INITIATE_IPL_PROC;
```

```
CS.PU_T2_LOAD_RSP: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE HANDLES THE LOADING OF PU_T2 NODES.       │
│                                                                          │
│                 THE NS_IPL_INIT  REQUEST IS GENERATED BY  CS.INITIATE_IPL_PROC (PAGE │
│                 7-91).  UPON  RECEIPT OF THE  RESPONSE TO NS_IPL_INIT,  THIS ROUTINE │
│                 CALLS UPM_BUILD_TEXT_OR_FINAL (PAGE 7-127).  UPM_BUILD_TEXT_OR_FINAL │
│                 RETURNS A POINTER TO THE FIRST NS_IPL_TEXT REQUEST.  UPON RECEIPT OF │
│                 THE RSP(NS_IPL_TEXT),  UPM_BUILD_TEXT_OR_FINAL IS  CALLED TO  CREATE │
│                 THE  NEXT  NS_IPL_TEXT  REQUEST.   WHEN THE  LOAD  MODULE  HAS  BEEN │
│                 COMPLETELY   TRANSFERRED,   UPM_BUILD_TEXT_OR_FINAL    CREATES   THE │
│                 NS_IPL_FINAL   REQUEST.   NS_IPL_FINAL   INCLUDES  THE   ENTRY-POINT │
│                 LOCATION FOR THE  PU_T2 NODE TO BEGIN EXECUTION OF  THE LOAD MODULE. │
│                 IF AN ERROR IS DETECTED, THIS  PROCEDURE PASSES AN APPROPRIATE SENSE │
│                 CODE  TO  CS.PU_T2_IPL_ABORT (PAGE  7-93).   CS.PU_T2_IPL_ABORT,  IN │
│                 TURN, SENDS NS_IPL_ABORT WITH THE SENSE CODE TO THE PU_T2.           │
│                                                                          │
│      INPUT:     ±RSP(NS_IPL_INIT |  NS_IPL_TEXT | NS_IPL_FINAL |  NS_IPL_ABORT) FROM │
│                 SNS.RCV (CHAPTER 6)                                      │
│                                                                          │
│      OUTPUT:    NS_IPL_TEXT  |   NS_IPL_FINAL  TO  SNS.SEND   (CHAPTER  6),   OR  AN │
│                 APPROPRIATE SENSE CODE TO CS.PU_T2_IPL_ABORT  (PAGE 7-93)            │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 SSCP.SVC_MGR.CS.RCV                PAGE 7-50             │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 CS.PU_T2_IPL_ABORT                 PAGE 7-93             │
│                 FSM_PU_ACT_DOM_RES                 PAGE 7-128            │
│                 FSM_PU_T2_IPL_DOM_RES              PAGE 7-133            │
│                 UPM_BUILD_TEXT_OR_FINAL            PAGE 7-127            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
SELECT ANYORDER(NS_RQ_CODE);
                                                                        /*
                ┌──────────────────────────────────────────────┐
                │  POSITIVE OR NEGATIVE RESPONSE TO NS_IPL_INIT  │
                └──────────────────────────────────────────────┘
                                                                        */
. WHEN(NS_IPL_INIT)
.   DO;
.   . IF FSM_PU_T2_IPL_DOM_RES ¬= TEXT THEN                  /* PAGE 7-133      */
.   .   CALL CS.PU_T2_IPL_ABORT(X'0809');  /* MODE INCONSISTENCY, PAGE 7-93      */
.   .
.   . ELSE
.   .   IF RTI = NEGATIVE THEN
.   .     CALL CS.PU_T2_IPL_ABORT(SNC);                      /* PAGE 7-93       */
.   .
.   .   ELSE
.   .     DO;
.   .     . MU_PTR = UPM_BUILD_TEXT_OR_FINAL;                /* PAGE 7-127      */
.   .     . CALL FSM_PU_T2_IPL_DOM_RES;                      /* PAGE 7-133      */
.   .     . SEND MU TO SNS.SEND;                             /* CHAPTER 6       */
.   .     END;
.   END;
                                                                        /*
                ┌──────────────────────────────────────────────┐
                │  POSITIVE OR NEGATIVE RESPONSE TO NS_IPL_TEXT  │
                └──────────────────────────────────────────────┘
                                                                        */
. WHEN(NS_IPL_TEXT)
.   DO;
.   . IF FSM_PU_T2_IPL_DOM_RES ¬= TEXT THEN                  /* PAGE 7-133      */
.   .   CALL CS.PU_T2_IPL_ABORT(X'0809');  /* MODE INCONSISTENCY, PAGE 7-93      */
.   .
.   . ELSE
.   .   IF RTI = NEGATIVE THEN
.   .     CALL CS.PU_T2_IPL_ABORT(SNC);                      /* PAGE 7-93       */
.   .
.   .   ELSE
.   .     DO;
.   .     . MU_PTR = UPM_BUILD_TEXT_OR_FINAL;                /* PAGE 7-127      */
.   .     . CALL FSM_PU_T2_IPL_DOM_RES;                      /* PAGE 7-133      */
.   .     . SEND MU TO SNS.SEND;                             /* CHAPTER 6       */
.   .     END;
.   END;
```

```
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │ POSITIVE OR NEGATIVE REPONSE TO NS_IPL_FINAL │
                    └──────────────────────────────────────────┘
                                                                        */
  . WHEN(NS_IPL_FINAL)
  .   DO;
  .   . IF FSM_PU_T2_IPL_DOM_RES ¬= TEXT THEN
  .   .   CALL CS.PU_T2_IPL_ABORT(X'0809');  /* MODE INCONSISTENCY, PAGE 7-93  */
  .   .
  .   . ELSE
  .   .   IF RTI = NEGATIVE THEN                                  /* PAGE 7-133 */
  .   .     CALL CS.PU_T2_IPL_ABORT(SNC);                         /* PAGE 7-93  */
  .   .
  .   .   ELSE
  .   .     DO;
  .   .     . CALL FSM_PU_T2_IPL_DOM_RES;                         /* PAGE 7-133 */
  .   .     . CALL FSM_PU_ACT_DOM_RES;                            /* PAGE 7-128 */
  .   .     . SEND MU TO UPM_TRANSLATION_SVC;                     /* CHAPTER 6  */
  .   .     END;
  .   END;
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │ POSITIVE OR NEGATIVE RESPONSE TO NS_IPL_ABORT │
                    └──────────────────────────────────────────┘
                                                                        */
  . WHEN(NS_IPL_ABORT)
  .   DO;
  .   . IF RTI = NEGATIVE THEN
  .   .   CALL UPM_LOG;                                           /* APPENDIX B */
  .   . SEND MU TO UPM_TRANSLATION_SVC;                           /* CHAPTER 6  */
  .   END;
  END;

END CS.PU_T2_LOAD_RSP;



CS.PU_T2_IPL_ABORT: PROCEDURE(SENSE);
                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE IS INVOKED WHEN AN  SSCP CANNOT COMPLETE A PU_T2 LOAD │
  │                 OPERATION.  FAILURE TO  COMPLETE A LOAD OPERATION  CAN HAPPEN EITHER │
  │                 WHEN -RSP(NS_IPL_INIT | NS_IPL_TEXT | NS_IPL_FINAL) IS RECEIVED FROM │
  │                 THE PU_T2,  OR IF  THE SSCP LOSES  ACCESS TO  THE LOAD  MODULE.  THE │
  │                 NS_IPL_ABORT CARRIES  THE APPROPRIATE SENSE  DATA TO THE  PU_T2.  IF │
  │                 THE  NS_IPL_ABORT IS  THE RESULT  OF  A NEGATIVE  RESPONSE FROM  THE │
  │                 PU_T2, THE SENSE  CODE OF THE RESPONSE  IS PLACED IN THE  SENSE DATA │
  │                 FIELD OF THE  NS_IPL_ABORT.  IF LOAD WAS REQUESTED  VIA THE RESPONSE │
  │                 TO ACTPU, THIS ROUTINE SENDS A DACTPU TO CS.PU_PROC (PAGE 7-52).  IF │
  │                 LOAD WAS REQUESTED VIA LDREQD, THE PU_T2 MAY REQUEST ANOTHER LOAD OR │
  │                 MAY SEND REQDISCONT.                                     │
  │                                                                          │
  │      INPUT:     THE APPROPRIATE  SENSE CODE  IS PASSED  ALONG WITH  THE CURRENT  MU. │
  │                 -RSP(NS_IPL_INIT  |  NS_IPL_TEXT  |  NS_IPL_FINAL),  -RSP(INITPROC), │
  │                 LDREQD, OR RSP(ACTPU)                                    │
  │                                                                          │
  │      OUTPUT:    NS_IPL_ABORT; DACTPU WHEN APPROPRIATE                     │
  │                                                                          │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
  │                 CS.INITIATE_IPL_PROC              PAGE 7-91              │
  │                 CS.INITPROC_RSP                   PAGE 7-88              │
  │                 CS.PU_T2_LOAD_RSP                 PAGE 7-92              │
  │                                                                          │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
  │                 CS.PU_PROC                        PAGE 7-52              │
  │                 FSM_PU_ACT_DOM_RES                PAGE 7-128             │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL SAVE_MU_PTR PTR;
  DCL SENSE BIT(32);

  SAVE_MU_PTR = MU_PTR;

  MU_PTR = UPM_CREATE_RQ('NS_IPL_ABORT');                          /* APPENDIX B */
  NS_IPL_ABORT_RQ.SENSE_DATA = SENSE;
  SEND MU TO SNS.SEND;                                             /* CHAPTER 6  */

  IF FSM_PU_ACT_DOM_RES ¬= ACTIVE THEN                             /* PAGE 7-128 */
    DO;
    . MU_PTR = UPM_CREATE_RQ('DACTPU');                            /* APPENDIX B */
    . CALL FSM_PU_ACT_DOM_RES;                                     /* PAGE 7-128 */
    . CALL CS.PU_PROC;                                             /* PAGE 7-52  */
    END;

  MU_PTR = SAVE_MU_PTR;

END CS.PU_T2_IPL_ABORT;
```

```
CS.RNAA_PROC: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE HANDLES THE ASSIGNMENT OF NETWORK ADDRESSES.
 │
 │                 THE FIRST  CHECK MADE  IS TO  DETERMINE IF  THE SSCP  HAS SUFFICIENT
 │                 RESOURCES  TO ASSIGN  THE NETWORK  ADDRESSES SPECIFIED  IN THE  RNAA
 │                 REQUEST.   IF NOT,  THIS PROCEDURE  GENERATES A  SEND_CHECK WITH  AN
 │                 APPROPRIATE ERROR  CODE, WHICH IS  SENT TO  UPM_TRANSLATION_SVC.  IF
 │                 THERE  ARE SUFFICIENT  RESOURCES, THE  TARGET RESOURCE  IN THE  RNAA
 │                 REQUEST IS CHECKED TO SEE IF IT IS ACTIVE.  IF IT IS NOT ACTIVE, THE
 │                 PROCEDURE  RESOURCE_ACTIVE_CHECK,   WHICH  PERFORMS   THE  CHECKING,
 │                 INSERTS THE REQUEST IN THE TARGET RESOURCE'S SAVE_MU_FOR_RETRY_LIST.
 │                 IF THE TARGET RESOURCE IS ACTIVE, THE RNAA IS SENT TO SNS.SEND.
 │
 │      INPUT:     RNAA FROM UPM_TRANSLATION_SVC (CHAPTER 6) OR FROM CS.ACTPU_RSP (PAGE
 │                 7-54)
 │
 │      OUTPUT:    RNAA  TO  SNS.SEND (CHAPTER  6),  A  COPY  OF  THE RNAA  REQUEST  TO
 │                 UPM_SAVE_RNAA_REQUEST (PAGE 7-125), AND A COPY OF THE TARGET ADDRESS
 │                 TO  UPM_SAVE_TARGET_NA  (PAGE  7-122);  OR   A  SEND_CHECK  WITH  AN
 │                 APPROPRIATE ERROR CODE TO UPM_TRANSLATION_SVC (CHAPTER 6)
 │
 │      NOTE:      PROCESSING OF THIS  REQUEST RESUMES IN CS.RNAA_RSP  (PAGE 7-95) WHEN
 │                 SNS RETURNS A RESPONSE.
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 CS.ACTPU_RSP                    PAGE 7-54
 │                 SSCP.SVC_MGR.CS.SEND            PAGE 7-48
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 RESOURCE_ACTIVE_CHECK           PAGE 7-116
 │                 UPM_RNAA_RESOURCE_CHECK         PAGE 7-123
 │                 UPM_SAVE_RNAA_RQ                PAGE 7-125
 │                 UPM_SAVE_TARGET_NA              PAGE 7-122
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RC BIT(1);
  DCL RES_NA BIT(48);
  DCL RES_TYPE BIT(4);

  IF UPM_RNAA_RESOURCE_CHECK = NG THEN                  /* PAGE 7-123      */
    SEND SEND_CHECK(X'0812') TO UPM_TRANSLATION_SVC;    /* INSUFFICIENT RESOURCES */

  ELSE
    DO;
    . RES_NA = DSAF||(NSC_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
    .                                                   /* APPENDIX A      */
    . DRCB_PTR = FIND_DOMAIN_RESOURCE(RES_NA);          /* APPENDIX B      */
    . RES_TYPE = DRCB.RESOURCE_CATEGORY;
    .
    . IF RESOURCE_ACTIVE_CHECK(RES_NA,RES_TYPE) = OK THEN  /* PAGE 7-116    */
    .   DO;
    .   .
    .   . CALL UPM_SAVE_RNAA_RQ;                        /* PAGE 7-125      */
    .   .
    .   . CALL UPM_SAVE_TARGET_NA(RES_NA);              /* PAGE 7-122      */
    .   .
    .   . SEND MU TO SNS.SEND;                          /* CHAPTER 6       */
    .   .
    .   END;
    END;

END CS.RNAA_PROC;
```

```
CS.RNAA_RSP: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │     FUNCTION:  WHEN THE INPUT IS A POSITIVE  RESPONSE TO RNAA, THIS  │
 │                PROCEDURE CALLS THE APPROPRIATE PROCEDURE, WHICH      │
 │                ASSIGNS THE REQUESTED NETWORK ADDRESSES AND CREATES   │
 │                AND ADDS DOMAIN RESOURCE ENTRIES TO THE DOMAIN        │
 │                RESOURCE LIST.                                        │
 │                                                                      │
 │     INPUT:     POSITIVE OR NEGATIVE RESPONSE TO RNAA FROM SNS.RCV    │
 │                (CHAPTER 6)                                           │
 │                                                                      │
 │     OUTPUT:    ±RSP(RNAA) TO UPM_TRANSLATION_SVC (CHAPTER 6)         │
 │                                                                      │
 │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
 │                SSCP.SVC_MGR.CS.RCV              PAGE 7-50            │
 │                                                                      │
 │     REFERS TO THE FOLLOWING PROCEDURE(S):                            │
 │                CS.LU_ADD                        PAGE 7-98            │
 │                CS.PERIPHERAL_LU_ADD             PAGE 7-97            │
 │                CS.PERIPHERAL_PU_AND_ALS_ADD     PAGE 7-96            │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF RTI = POSITIVE THEN                              /* RESPONSE IS POSITIVE  */

    SELECT ANYORDER(RNAA_RSP.ASSIGNMENT_TYPE);
    .
    . WHEN(RNAA_BF_PU)
    .   CALL CS.PERIPHERAL_PU_AND_ALS_ADD;            /* PAGE 7-96             */
    .
    . WHEN(RNAA_BF_LU)
    .   CALL CS.PERIPHERAL_LU_ADD;                    /* PAGE 7-97             */
    .
    . WHEN(RNAA_LU)
    .   CALL CS.LU_ADD;                               /* PAGE 7-98             */
    .
    END;

  SEND MU TO UPM_TRANSLATION_SVC;                     /* CHAPTER 6             */

END CS.RNAA_RSP;
```

```
CS.PERIPHERAL_PU_AND_ALS_ADD: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  THIS PROCEDURE  ADDS ENTRIES  TO THE  DOMAIN RESOURCE  LIST FOR  THE │
│                PERIPHERAL PU'S AND ALS'S SPECIFIED IN THE RSP(RNAA).     │
│                                                                          │
│     INPUT:     +RSP(RNAA) FROM  CS.RNAA_RSP (PAGE  7-95) AND A  COPY OF  THE TARGET │
│                ADDRESS FROM UPM_RETRIEVE_TARGET_NA (PAGE 7-122)          │
│                                                                          │
│     OUTPUT:    THE DOMAIN  RESOURCE ENTRIES  ARE CREATED  AND ADDED  TO THE  DOMAIN │
│                RESOURCE LIST.                                            │
│                                                                          │
│     NOTE:      THE NETWORK ADDRESS OF  A PERIPHERAL PU IS IDENTICAL TO  THAT OF ITS │
│                ASSOCIATED ALS.                                           │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                CS.RNAA_RSP                    PAGE 7-95                  │
│                                                                          │
│     REFERS TO THE FOLLOWING PROCEDURE(S):                                │
│                UPM_RETRIEVE_TARGET_NA         PAGE 7-122                 │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RES_NA BIT(48);
  DCL SAVE_PTR PTR;
  DCL TARGET_NA BIT(48);

  TARGET_NA = UPM_RETRIEVE_TARGET_NA;                    /* PAGE 7-122      */

  DO I = 0 TO RNAA_RSP.ENTRY_CNT - 1;
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │       CREATE AN ALS DOMAIN RESOURCE ENTRY       │
                  └──────────────────────────────────────┘
                                                                        */
  . CREATE DRCB PTR(DRCB_PTR);
  .
  . SAVE_PTR = DRCB_PTR;
  . DRCB.RESOURCE_CATEGORY = ALS;
  . RES_NA = OSAF||(RNAA_RSP.SUBFIELD(I) & NCB.NODE_ELEMENT_MASK);
  .                                                      /* APPENDIX A      */
  . DRCB.NETWORK_ADDRESS = RES_NA;
  . DRCB.ASSOCIATED_RES_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);   /* APPENDIX B */
  .
  . IF DRCB.ASSOCIATED_RES_PTR->DRCB.SWITCHED_LINK = SWITCHED THEN
  .   DRCB.SWITCHED_LINK = SWITCHED;
  .
  . ELSE
  .   DRCB.SWITCHED_LINK = NONSWITCHED;
  .
  . INSERT DRCB IN DRCB_LIST;
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │  CREATE A PERIPHERAL PU DOMAIN RESOURCE ENTRY   │
                  └──────────────────────────────────────┘
                                                                        */
  . CREATE DRCB PTR(DRCB_PTR);
  .
  . DRCB.RESOURCE_CATEGORY = PERIPHERAL_PU;
  . DRCB.NETWORK_ADDRESS = RES_NA;
  . DRCB.ASSOCIATED_RES_PTR = SAVE_PTR;
  . DRCB.BF_LOCAL_ID = 0;
  .
  . INSERT DRCB IN DRCB_LIST;
  END;

  RETURN;

END CS.PERIPHERAL_PU_AND_ALS_ADD;
```

```
CS.PERIPHERAL_LU_ADD: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE  ADDS ENTRIES  TO THE  DOMAIN RESOURCE  LIST, IF  THE │
 │                 ENTRIES DO NOT  ALREADY EXIST, FOR THE PERIPHERAL  LU'S SPECIFIED IN │
 │                 THE RSP(RNAA).                                       │
 │                                                                      │
 │                 AN  ENTRY FOR  A PERIPHERAL  LU  WHOSE ASSOCIATED  LINK IS  SWITCHED │
 │                 ALREADY EXISTS IN  THE DOMAIN RESOURCE LIST PRIOR  TO THE PROCESSING │
 │                 OF THIS PROCEDURE;  HOWEVER, THE NETWORK ADDRESS FIELD  OF THE ENTRY │
 │                 HAS  NOT BEEN  INITIALIZED.  THIS PROCEDURE  STORES  IN THE  DOMAIN │
 │                 RESOURCE ENTRY FOR  THE LU THE NETWORK ADDRESS RETURNED  IN THE RNAA │
 │                 RESPONSE, AND GENERATES SETCV AND ACTLU REQUESTS FOR THE LU.         │
 │                                                                      │
 │      INPUT:     *RSP(RNAA) FROM CS.RNAA_RSP (PAGE 7-95), A  COPY OF THE RNAA REQUEST │
 │                 FROM UPM_RETRIEVE_RNAA_RQ  (PAGE 7-126),  AND A  COPY OF  THE TARGET │
 │                 ADDRESS FROM UPM_RETRIEVE_TARGET_NA (PAGE 7-122)     │
 │                                                                      │
 │      OUTPUT:    THE DOMAIN RESOURCE LIST ENTRIES ARE CREATED AND ADDED TO THE DOMAIN │
 │                 RESOURCE LIST.  IF THE PERIPHERAL  LU'S ASSOCIATED LINK IS SWITCHED, │
 │                 SETCV AND ACTLU ARE GENERATED AND  SENT TO THE APPROPRIATE PROCEDURE │
 │                 FOR PROCESSING.                                      │
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 CS.RNAA_RSP                          PAGE 7-95       │
 │                                                                      │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
 │                 CS.LU_PROC                           PAGE 7-58       │
 │                 FSM_LU_ACT_DOM_RES                   PAGE 7-128      │
 │                 UPM_RETRIEVE_RNAA_RQ                 PAGE 7-126      │
 │                 UPM_RETRIEVE_TARGET_NA               PAGE 7-122      │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL TARGET_NA BIT(48);
  DCL 1 RNAA_RQ_COPY,
        2 NS_REQUEST_CODE BIT(24),
        2 TARGET_ADDRESS BIT(16),
        2 ASSIGNMENT_TYPE BIT(8),
        2 ENTRY_CNT BIT(8),
        2 SUBFIELD(40) BIT(16);
  DCL P POINTER;
  DCL SAVE_MU_PTR POINTER;
  DCL LINK_PTR POINTER;

  TARGET_NA = UPM_RETRIEVE_TARGET_NA;                   /* PAGE 7-122    */
  SAVE_MU_PTR = MU_PTR;
  P = ADDR(RNAA_RSP);
  LINK_PTR = FIND_LINK_FOR_DOM_RES(TARGET_NA);          /* APPENDIX B    */
  CALL UPM_RETRIEVE_RNAA_RQ(RNAA_RQ_COPY);              /* PAGE 7-126    */

  DO I = 0 TO P->RNAA_RSP.ENTRY_CNT - 1;
  .
  . FIND DRCB IN DRCB_LIST
  .      WHERE(DRCB.RESOURCE_CATEGORY = PERIPHERAL_LU &
  .      DRCB.BF_LOCAL_ID = RNAA_RQ_COPY.SUBFIELD(I,8:15) &
  .      DRCB.ASSOCIATED_RES_PTR->DRCB.NETWORK_ADDRESS = TARGET_NA);
  .
  . IF DRCB_PTR = NULL THEN
  .   DO;
  .   . CREATE DRCB PTR(DRCB_PTR);
  .   . DRCB.RESOURCE_CATEGORY = PERIPHERAL_LU;
  .   . DRCB.ASSOCIATED_RES_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);  /* APPENDIX B */
  .   . DRCB.BF_LOCAL_ID = RNAA_RQ_COPY.SUBFIELD(I,8:15);
  .   . INSERT DRCB IN DRCB_LIST;
  .   END;
  .
  . DRCB.NETWORK_ADDRESS = OSAF||P->RNAA_RSP.SUBFIELD(I) & NCB.NODE_ELEMENT_MASK;
  .                                                     /* APPENDIX A    */
  . IF LINK_PTR->DRCB.SWITCHED_LINK = SWITCHED &
  .      FSM_LU_ACT_DOM_RES = RESET THEN                /* PAGE 7-128    */
  .   DO;
  .   . MU_PTR = UPM_CREATE_RQ('SETCV');                /* APPENDIX B    */
  .   . SEND MU TO SNS.SEND;                            /* CHAPTER 6     */
  .   . MU_PTR = UPM_CREATE_RQ('ACTLU');                /* APPENDIX B    */
  .   . DSAF = OSAF;
  .   . CALL CS.LU_PROC;                                /* PAGE 7-58     */
  .   END;
  .
  END;

  MU_PTR = SAVE_MU_PTR;

  RETURN;

END CS.PERIPHERAL_LU_ADD;
```

```
CS.LU_ADD: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE ADDS ENTRIES TO THE DOMAIN RESOURCE LIST FOR THE LU'S  │
  │                 SPECIFIED IN THE RSP(RNAA).                                  │
  │                                                                              │
  │      INPUT:     +RSP(RNAA) FROM  CS.RNAA_RSP (PAGE  7-95) AND A  COPY OF  THE TARGET  │
  │                 ADDRESS FROM UPM_RETRIEVE_TARGET_NA (PAGE 7-122)             │
  │                                                                              │
  │      OUTPUT:    THE DOMAIN  RESOURCE ENTRIES  ARE CREATED  AND ADDED  TO THE  DOMAIN  │
  │                 RESOURCE LIST.                                               │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 CS.RNAA_RSP                    PAGE 7-95                     │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 UPM_RETRIEVE_TARGET_NA         PAGE 7-122                    │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL TARGET_NA BIT(48);

  TARGET_NA = UPM_RETRIEVE_TARGET_NA;                    /* PAGE 7-122          */

  CREATE DRCB PTR(DRCB_PTR);

  DRCB.RESOURCE_CATEGORY = SUBAREA_LU;
  DRCB.NETWORK_ADDRESS = OSAP||(RNAA_RSP.SUBFIELD(0) & NCB.NODE_ELEMENT_MASK);
                                                         /* APPENDIX A          */
  DRCB.ASSOCIATED_RES_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);     /* APPENDIX B    */

  INSERT DRCB IN DRCB_LIST;

  RETURN;

END CS.LU_ADD;
```

```
CS.FNA_PROC: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │     FUNCTION:  THIS PROCEDURE HANDLES THE FREEING OF NETWORK ADDRESSES.
 │
 │                IF  THE TARGET  ADDRESS  OF THE  FNA REQUEST  IS  0, THIS  PROCEDURE
 │                DETERMINES THE  TARGET ADDRESS  TO BE  USED BY  EXAMINING THE  FIRST
 │                ADDRESS IN THE LIST OF ADDRESSES TO  BE FREED (SEE FNA IN APPENDIX E
 │                FOR SPECIFIC DETAILS).
 │
 │                NEXT, THE TARGET RESOURCE IN THE FNA REQUEST IS CHECKED TO SEE IF IT
 │                IS  ACTIVE.  IF  THE RESOURCE'S  FSM  IS NOT  ACTIVE, THE  PROCEDURE
 │                RESOURCE_ACTIVE_CHECK,  WHICH  PERFORMS THE  CHECKING,  INSERTS  THE
 │                REQUEST INTO THE TARGET RESOURCE'S SAVE_RU_FOR_RETRY_LIST.
 │
 │                IF THE TARGET  RESOURCE'S FSM IS ACTIVE, THE FNA  REQUEST IS CHECKED
 │                TO SEE IF IT IS VALID.  IF SO, A  COPY OF THE FNA REQUEST AND OF THE
 │                TARGET  ADDRESS ARE  SAVED  BY  UPM'S FOR  USE  WHEN  A RESPONSE  IS
 │                RETURNED.  THE FNA IS THEN SENT TO SNS.SEND.
 │
 │                NO NETWORK ADDRESSES ARE FREED UNTIL A POSITIVE RESPONSE IS RECEIVED
 │                FROM SNS.  THE PROCEDURE CS.FNA_RSP  (PAGE 7-102) HANDLES THE ACTUAL
 │                FREEING OF ADDRESSES.
 │
 │                IF THE  REQUEST IS  INVALID, THEN  A SEND_CHECK  WITH AN  ERROR CODE
 │                DETERMINED BY THE CHECKING PROCEDURE IS SENT TO UPM_TRANSLATION_SVC.
 │
 │     INPUT:     FNA FROM  UPM_TRANSLATION_SVC (CHAPTER  6) OR  FROM CS.LU_RSP  (PAGE
 │                7-60)
 │
 │     OUTPUT:    FNA TO  SNS.SEND (CHAPTER 6),  A COPY  OF THE RQ  TO UPM_SAVE_FNA_RQ
 │                (PAGE 7-125), AND A COPY OF THE TARGET ADDRESS TO UPM_SAVE_TARGET_NA
 │                (PAGE 7-122);  OR A  SEND_CHECK WITH  AN APPROPRIATE  ERROR CODE  TO
 │                UPM_TRANSLATION_SVC (CHAPTER 6)
 │
 │     NOTE:      PROCESSING OF THIS  REQUEST RESUMES IN CS.FNA_RSP  (PAGE 7-102) WHEN
 │                SNS RETURNS A RESPONSE.
 │
 │     REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                CS.LU_RSP                          PAGE 7-60
 │                SSCP.SVC_MGR.CS.SEND               PAGE 7-48
 │
 │     REFERS TO THE FOLLOWING PROCEDURE(S):
 │                CS.FNA_VALIDITY_CHECK              PAGE 7-100
 │                RESOURCE_ACTIVE_CHECK              PAGE 7-116
 │                UPM_SAVE_FNA_RQ                    PAGE 7-125
 │                UPM_SAVE_TARGET_NA                 PAGE 7-122
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL TARGET_NA BIT(48);
  DCL RES_TYPE BIT(4);
  DCL SENSE BIT(16);
  DCL ASSOC_PTR POINTER;
  DCL P POINTER;

  IF FNA_RQ.TARGET_ADDRESS = 0 THEN
    DO;
    . DRCB_PTR = FIND_DOMAIN_RESOURCE(FNA_RQ.SUBFIELD(1));       /* APPENDIX B   */
    .
    . IF DRCB.RESOURCE_CATEGORY = (ALS | PERIPHERAL_PU) THEN
    .   DO;
    .   . P = FIND_LINK_FOR_DOM_RES(DRCB.NETWORK_ADDRESS);       /* APPENDIX B   */
    .   . TARGET_NA = P->DRCB.NETWORK_ADDRESS;
    .   END;
    .
    . ELSE
    .   DO;
    .   . ASSOC_PTR = DRCB.ASSOCIATED_RES_PTR;
    .   . TARGET_NA = ASSOC_PTR->DRCB.NETWORK_ADDRESS;
    .   END;
    END;

  ELSE
    TARGET_NA = DSAF||(FNA_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
                                                                 /* APPENDIX A   */
  DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);                    /* APPENDIX B   */
  RES_TYPE = DRCB.RESOURCE_CATEGORY;

  IF RESOURCE_ACTIVE_CHECK(TARGET_NA,RES_TYPE) = OK THEN
    DO;
    . SENSE = CS.FNA_VALIDITY_CHECK(TARGET_NA);                  /* PAGE 7-100   */
    .
    . IF SENSE = 0 THEN
    .   DO;
    .   . CALL UPM_SAVE_FNA_RQ;                                  /* PAGE 7-125   */
    .   . CALL UPM_SAVE_TARGET_NA(TARGET_NA);                    /* PAGE 7-122   */
    .   . SEND RU TO SNS.SEND;                                   /* CHAPTER 6    */
    .   END;
    .
    . ELSE
    .   SEND SEND_CHECK(SENSE) TO UPM_TRANSLATION_SVC;           /* CHAPTER 6    */
    END;

END CS.FNA_PROC;
```

```
CS.FNA_VALIDITY_CHECK: PROCEDURE(TARGET_RES);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE CHECKS  TO SEE THAT THE CONDITIONS  REQUIRED TO ALLOW │
│                 THE REQUESTED FNA ARE MET.                               │
│                                                                          │
│                 WHEN  SPECIFIC  ALS'S AND  PERIPHERAL  PU'S  ARE  TO BE  FREED,  THE │
│                 ADDRESSES ARE  CHECKED TO SEE THAT  THEY ARE IN THE  DOMAIN RESOURCE │
│                 LIST, THAT  THEY ARE ASSOCIATED WITH  THE TARGET RESOURCE,  AND THAT │
│                 THE SEC_ALS_SUBTREE IS RESET.                            │
│                                                                          │
│                 WHEN ALL ALS'S AND PERIPHERAL PU'S ASSOCIATED WITH A PARTICULAR LINK │
│                 ARE TO BE FREED, THE DOMAIN RESOURCE LIST IS SEARCHED TO FIND ALL OF │
│                 THE ALS'S  AND PERIPHERAL  PU'S CORRESPONDING TO  THE LINK,  AND THE │
│                 PERTINENT ALS_SEC_SUBTREE'S ARE CHECKED TO SEE THAT THEY ARE RESET.  │
│                                                                          │
│                 WHEN SPECIFIC PERIPHERAL  OR SUBAREA LU'S ARE TO BE  FREED, THEN THE │
│                 ADDRESSES ARE  CHECKED TO SEE THAT  THEY ARE IN THE  DOMAIN RESOURCE │
│                 LIST, AND THAT THEY ARE ASSOCIATED WITH THE TARGET RESOURCE.         │
│                                                                          │
│                 WHEN ALL  PERIPHERAL LU'S  ASSOCIATED WITH A  PERIPHERAL PU,  OR ALL │
│                 PRIMARY  PARALLEL-SESSION LU  NETWORK  ADDRESSES  ASSOCIATED WITH  A │
│                 SECONDARY PARALLEL-SESSION LU  NETWORK ADDRESS, OR ALL  SUBAREA LU'S │
│                 ASSOCIATED WITH A  SUBAREA PU ARE TO  BE FREED, THEN NO  CHECKING IS │
│                 REQUIRED.                                                │
│                                                                          │
│      INPUT:     THE  FNA  REQUEST  AND  THE ADDRESS  OF  THE  TARGET  RESOURCE  FROM │
│                 CS.FNA_PROC (PAGE 7-99)                                  │
│                                                                          │
│      OUTPUT:    0 IF THE FNA IS VALID; THE  APPROPRIATE SENSE CODE IF THE FNA CANNOT │
│                 BE EXECUTED                                              │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 CS.FNA_PROC                          PAGE 7-99           │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 SEC_ALS_SUBTREE_CHECK                PAGE 7-121          │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL TARGET_RES BIT(48);
DCL RC BIT(16);
DCL SUB_PTR POINTER;
DCL ASSOC_PTR POINTER;
DCL ALS_NA BIT(48);

RC = X'0000';
DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_RES);                    /* APPENDIX B          */

SELECT ANYORDER;
                                                                        /*
                 ┌──────────────────────────────────────────┐
                 │   WHEN SPECIFIC ALS'S AND PERIPHERAL PU'S ARE   │
                 │                  TO BE FREED                    │
                 └──────────────────────────────────────────┘
                                                                        */
. WHEN((DRCB.RESOURCE_CATEGORY = LINK) & (FNA_RQ.ENTRY_CNT ¬= ALL))
.   DO I = 0 TO FNA_RQ.ENTRY_CNT - 1 WHILE(RC = 0);
.   . SUB_PTR = FIND_DOMAIN_RESOURCE(FNA_RQ.SUBFIELD(I));       /* APPENDIX B          */
.   .
.   . IF SUB_PTR = NULL THEN
.   .   RC = X'0806';                                           /* RESOURCE UNKNOWN    */
.   .
.   . ELSE
.   .   IF SUB_PTR->DRCB.RESOURCE_CATEGORY = PERIPHERAL_PU THEN
.   .     SUB_PTR = SUB_PTR->DRCB.ASSOCIATED_RES_PTR;
.   .
.   . ALS_NA = SUB_PTR->DRCB.NETWORK_ADDRESS;
.   . ASSOC_PTR = SUB_PTR->DRCB.ASSOCIATED_RES_PTR;
.   .
.   . IF ASSOC_PTR->DRCB.NETWORK_ADDRESS ¬= TARGET_RES THEN
.   .   RC = X'0809';                                           /* MODE INCONSISTENCY  */
.   .
.   . ELSE
.   .   IF SEC_ALS_SUBTREE_CHECK(ALS_NA) = NG THEN              /* PAGE 7-121          */
.   .     RC = X'0809';                                         /* MODE INCONSISTENCY  */
.   END;
```

```
                    ┌──────────────────────────────────────────┐
                    │ WHEN ALL ALS'S AND PERIPHERAL PU'S ASSOCIATED │
                    │ WITH A LINK ARE TO BE FREED              │
                    └──────────────────────────────────────────┘

  . WHEN((DRCB.RESOURCE_CATEGORY = LINK) & (FNA_RQ.ENTRY_CNT = ALL))
  .   SCAN DRCB_LIST PTR(SUB_PTR) WHILE(RC = 0);
  .   . IF SUB_PTR->DRCB.RESOURCE_CATEGORY = (ALS | PERIPHERAL_PU) THEN
  .   .   DO;
  .   .   . IF SUB_PTR->DRCB.RESOURCE_CATEGORY = PERIPHERAL_PU THEN
  .   .   .   SUB_PTR = SUB_PTR->DRCB.ASSOCIATED_RES_PTR;
  .   .   .
  .   .   . ALS_NA = SUB_PTR->DRCB.NETWORK_ADDRESS;
  .   .   . ASSOC_PTR = SUB_PTR->DRCB.ASSOCIATED_RES_PTR;
  .   .   .
  .   .   . IF ASSOC_PTR->DRCB.NETWORK_ADDRESS = TARGET_RES THEN
  .   .   .   IF SEC_ALS_SUBTREE_CHECK(ALS_NA) = NG THEN          /* PAGE 7-121
  .   .   .     RC = X'0809';                                     /* MODE INCONSISTENCY
  .   .   END;
  .   SCANEND;

                    ┌──────────────────────────────────────────┐
                    │ WHEN SPECIFIC PERIPHERAL LU'S ASSOCIATED WITH │
                    │ A PERIPHERAL PU, OR SPECIFIC PRIMARY     │
                    │ PARALLEL-SESSION LU NETWORK ADDRESSES    │
                    │ ASSOCIATED WITH A SECONDARY PARALLEL-SESSION │
                    │ LU NETWORK ADDRESS, OR SPECIFIC SUBAREA LU'S │
                    │ ASSOCIATED WITH A SUBAREA PU ARE TO BE FREED │
                    └──────────────────────────────────────────┘

  . WHEN(DRCB.RESOURCE_CATEGORY = (PERIPHERAL_PU | SUBAREA_LU | SUBAREA_PU) &
  .      FNA_RQ.ENTRY_CNT ¬= ALL)
  .   DO I = 0 TO FNA_RQ.ENTRY_CNT - 1 WHILE(RC = 0);
  .   . SUB_PTR = FIND_DOMAIN_RESOURCE(FNA_RQ.SUBFIELD(I));       /* APPENDIX B              */
  .   . IF SUB_PTR = NULL THEN
  .   .   RC = X'0806';                                          /* RESOURCE UNKNOWN
  .   .
  .   . ELSE
  .   .   DO;
  .   .   . ASSOC_PTR = SUB_PTR->DRCB.ASSOCIATED_RES_PTR;
  .   .   . IF ASSOC_PTR->DRCB.NETWORK_ADDRESS ¬= TARGET_RES THEN
  .   .   .   RC = X'0809';                                      /* MODE INCONSISTENCY
  .   .   END;
  .   END;
  .
  .
  . OTHERWISE;
  .
  END;

  RETURN(RC);

END CS.FNA_VALIDITY_CHECK;
```

```
CS.FNA_RSP: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  WHEN THE INPUT  IS A POSITIVE RESPONSE TO FNA,  THIS PROCEDURE CALLS │
   │                 THE APPROPRIATE PROCEDURE, WHICH REMOVES THE DOMAIN RESOURCE ENTRIES │
   │                 FOR  THE  REQUESTED ADDRESSES  FROM  THE  DOMAIN RESOURCE  LIST  AND │
   │                 DISCARDS THEM.                                       │
   │                                                                      │
   │      INPUT:     POSITIVE OR  NEGATIVE RESPONSE  TO FNA FROM  SNS.RCV (CHAPTER  6), A │
   │                 COPY OF THE FNA REQUEST FROM UPM_RETRIEVE_FNA_RQ (PAGE 7-125), AND A │
   │                 COPY OF THE TARGET ADDRESS FROM UPM_RETRIEVE_TARGET_NA (PAGE 7-122)  │
   │                                                                      │
   │      OUTPUT:    ±RSP(FNA) TO UPM_TRANSLATION_SVC (CHAPTER 6)         │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 SSCP.SVC_MGR.CS.RCV              PAGE 7-50           │
   │                                                                      │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
   │                 CS.LU_FREE                       PAGE 7-105          │
   │                 CS.PERIPHERAL_LU_FREE            PAGE 7-104          │
   │                 CS.PERIPHERAL_PU_AND_ALS_FREE    PAGE 7-103          │
   │                 UPM_RETRIEVE_FNA_RQ              PAGE 7-125          │
   │                 UPM_RETRIEVE_TARGET_NA           PAGE 7-122          │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL 1 FNA_RQ_COPY,
        2 NS_REQUEST_CODE BIT(24),
        2 TARGET_ADDRESS BIT(16),
        2 ENTRY_CNT BIT(8),
        2 TYPE BIT(8),
        2 SUBFIELD(40) BIT(16);
  DCL TARGET_NA BIT(48);

  CALL UPM_RETRIEVE_FNA_RQ(FNA_RQ_COPY);                  /* PAGE 7-125   */
  TARGET_NA = UPM_RETRIEVE_TARGET_NA;                     /* PAGE 7-122   */
  DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);             /* APPENDIX B   */

  IF RTI = POSITIVE THEN                                  /* RESPONSE IS POSITIVE */

    SELECT ANYORDER(DRCB.RESOURCE_CATEGORY);
    .
    . WHEN(LINK)
    .   CALL CS.PERIPHERAL_PU_AND_ALS_FREE(FNA_RQ_COPY,TARGET_NA);
    .                                                     /* PAGE 7-103   */
    . WHEN(PERIPHERAL_PU)
    .   CALL CS.PERIPHERAL_LU_FREE(FNA_RQ_COPY,TARGET_NA);  /* PAGE 7-104   */
    .
    . WHEN(SUBAREA_PU | SUBAREA_LU)
    .   CALL CS.LU_FREE(FNA_RQ_COPY,TARGET_NA);           /* PAGE 7-105   */
    .
    END;

  SEND MU TO UPM_TRANSLATION_SVC;                         /* CHAPTER 6    */

END CS.FNA_RSP;
```

```
CS.PERIPHERAL_PU_AND_ALS_FREE: PROCEDURE(FNA_RQ_COPY,TARGET_RES);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE REMOVES  THE ENTRIES  FOR PERIPHERAL  PUS AND  THEIR │
   │                 ASSOCIATED ADJACENT LINK STATIONS FROM THE DOMAIN RESOURCE LIST.      │
   │                                                                      │
   │      INPUT:     THE ADDRESS  OF THE TARGET  RESOURCE AND A  COPY OF THE  FNA REQUEST │
   │                 FROM CS.FNA_RSP (PAGE 7-102)                         │
   │                                                                      │
   │      OUTPUT:    THE APPROPRIATE ADDRESSES ARE REMOVED FROM THE DOMAIN RESOURCE LIST.  │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 CS.FNA_RSP                          PAGE 7-102       │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL 1 FNA_RQ_COPY,
        2 NS_REQUEST_CODE BIT(24),
        2 TARGET_ADDRESS BIT(16),
        2 ENTRY_CNT BIT(8),
        2 TYPE BIT(8),
        2 SUBFIELD(40) BIT(16);
  DCL TARGET_RES BIT(48);
  DCL RES_NA BIT(48);
  DCL ALS_PTR POINTER;
  DCL P POINTER;

  SELECT ANYORDER(FNA_RQ_COPY.ENTRY_CNT);
  .
  . WHEN( ¬ALL)
  .   DO I = 0 TO(FNA_RQ_COPY.ENTRY_CNT - 1);
  .   . RES_NA = OSAF||(FNA_RQ_COPY.SUBFIELD(I) & NCB.NODE_ELEMENT_MASK);
  .   . P = FIND_DOMAIN_RESOURCE(RES_NA);                    /* APPENDIX B   */
  .   .
  .   . IF P->DRCB.RESOURCE_CATEGORY = PERIPHERAL_PU THEN
  .   .   ALS_PTR = P->DRCB.ASSOCIATED_RES_PTR;
  .   .
  .   . ELSE
  .   .   DO;
  .   .   . ALS_PTR = P;
  .   .   . P = FIND_SUBORDINATE_DOM_RES(RES_NA);           /* APPENDIX B   */
  .   .   END;
  .   .
  .   . REMOVE P->DRCB FROM DRCB_LIST DISCARD;
  .   . REMOVE ALS_PTR->DRCB FROM DRCB_LIST DISCARD;
  .   END;
  .
  . WHEN(ALL)
  .   SCAN DRCB_LIST PTR(DRCB_PTR);
  .   . P = DRCB.ASSOCIATED_RES_PTR;
  .   .
  .   . IF P->DRCB.NETWORK_ADDRESS = TARGET_RES &
  .   .      DRCB.RESOURCE_CATEGORY = ALS THEN
  .   .   DO;
  .   .   . RES_NA = DRCB.NETWORK_ADDRESS;
  .   .   . ALS_PTR = DRCB_PTR;
  .   .   . P = FIND_SUBORDINATE_DOM_RES(RES_NA);           /* APPENDIX B   */
  .   .   .
  .   .   . REMOVE P->DRCB FROM DRCB_LIST DISCARD;
  .   .   . REMOVE ALS_PTR->DRCB FROM DRCB_LIST DISCARD;
  .   .   END;
  .   SCANEND;
  .
  END;

  RETURN;

END CS.PERIPHERAL_PU_AND_ALS_FREE;
```

```
CS.PERIPHERAL_LU_FREE: PROCEDURE(FNA_RQ_COPY,TARGET_RES);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE REMOVES  ENTRIES FOR PERIPHERAL LU'S  FROM THE DOMAIN │
   │                 RESOURCE LIST IF THE LINK ASSOCIATED  WITH THE LU'S IS NOT SWITCHED. │
   │                 OTHERWISE, THE LU ENTRY REMAINS IN THE DOMAIN RESOURCE LIST, BUT THE │
   │                 NETWORK ADDRESS FIELD OF THE ENTRY IS SET TO 0.          │
   │                                                                          │
   │      INPUT:     THE ADDRESS  OF THE TARGET  RESOURCE AND A  COPY OF THE  FNA REQUEST │
   │                 FROM CS.FNA_RSP (PAGE 7-102)                             │
   │                                                                          │
   │      OUTPUT:    THE APPROPRIATE ADDRESS IS REMOVED FROM THE DOMAIN RESOURCE LIST. │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 CS.FNA_RSP                          PAGE 7-102           │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL 1 FNA_RQ_COPY,
        2 NS_REQUEST_CODE BIT(24),
        2 TARGET_ADDRESS BIT(16),
        2 ENTRY_CNT BIT(8),
        2 TYPE BIT(8),
        2 SUBFIELD(40) BIT(16);
  DCL TARGET_RES BIT(48);
  DCL PERIPHERAL_LU_NA BIT(48);
  DCL P POINTER;
  DCL LINK_PTR POINTER;

  SELECT ANYORDER(FNA_RQ_COPY.ENTRY_CNT);
  .
  . WHEN( ¬ALL)
  .   DO I = 0 TO(FNA_RQ_COPY.ENTRY_CNT - 1);
  .   . PERIPHERAL_LU_NA = OSAF||(FNA_RQ_COPY.SUBFIELD(I) & NCB.NODE_ELEMENT_MASK);
  .   . DRCB_PTR = FIND_DOMAIN_RESOURCE(PERIPHERAL_LU_NA);       /* APPENDIX B      */
  .   .
  .   . IF DRCB.RESOURCE_CATEGORY = PERIPHERAL_LU THEN
  .   .   DO;
  .   .   . LINK_PTR = FIND_LINK_FOR_DOM_RES(PERIPHERAL_LU_NA);  /* APPENDIX B      */
  .   .   .
  .   .   . IF LINK_PTR->DRCB.SWITCHED_LINK = SWITCHED THEN
  .   .   .   DRCB.NETWORK_ADDRESS = X'0';
  .   .   .
  .   .   . ELSE
  .   .   .   REMOVE DRCB FROM DRCB_LIST DISCARD;
  .   .   END;
  .   END;
  .
  . WHEN(ALL)
  .   SCAN DRCB_LIST PTR(DRCB_PTR);
  .   . P = DRCB.ASSOCIATED_RESOURCE_PTR;
  .   .
  .   . IF P->DRCB.NETWORK_ADDRESS = TARGET_RES &
  .   .      DRCB.RESOURCE_CATEGORY = PERIPHERAL_LU THEN
  .   .   DO;
  .   .   . LINK_PTR = FIND_LINK_FOR_DOM_RES(PERIPHERAL_LU_NA);  /* APPENDIX B      */
  .   .   .
  .   .   . IF LINK_PTR->DRCB.SWITCHED_LINK = SWITCHED THEN
  .   .   .   DRCB.NETWORK_ADDRESS = X'0';
  .   .   .
  .   .   . ELSE
  .   .   .   REMOVE DRCB FROM DRCB_LIST DISCARD;
  .   .   END;
  .   SCANEND;
  .
  END;

  RETURN;

END CS.PERIPHERAL_LU_FREE;
```

```
CS.LU_FREE: PROCEDURE(FNA_RQ_COPY,TARGET_RES);

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE  REMOVES ENTRIES  FOR LU'S  FROM THE  DOMAIN RESOURCE
  │                 LIST.
  │
  │      INPUT:     THE ADDRESS  OF THE TARGET  RESOURCE AND A  COPY OF THE  FNA REQUEST
  │                 FROM CS.FNA_RSP (PAGE 7-102)
  │
  │      OUTPUT:    THE APPROPRIATE ADDRESS IS REMOVED FROM THE DOMAIN RESOURCE LIST.
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 CS.FNA_RSP                          PAGE 7-102
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL 1 FNA_RQ_COPY,
        2 NS_REQUEST_CODE BIT(24),
        2 TARGET_ADDRESS BIT(16),
        2 ENTRY_CNT BIT(8),
        2 TYPE BIT(8),
        2 SUBFIELD(40) BIT(16);
  DCL TARGET_RES BIT(48);
  DCL P POINTER;

  SELECT ANYORDER(FNA_RQ_COPY.ENTRY_CNT);
  .
  . WHEN( ¬ALL)
  .   DO I = 0 TO(FNA_RQ_COPY.ENTRY_CNT - 1);
  .   . P = FIND_DOMAIN_RESOURCE(FNA_RQ_COPY.SUBFIELD(I));       /* APPENDIX B    */
  .   . IF P->DRCB.RESOURCE_CATEGORY = SUBAREA_LU THEN
  .   .   REMOVE P->DRCB FROM DRCB_LIST DISCARD;
  .   END;
  .
  . WHEN(ALL)
  .   SCAN DRCB_LIST PTR(DRCB_PTR);
  .   . P = DRCB.ASSOCIATED_RESOURCE_PTR;
  .   . IF P->DRCB.NETWORK_ADDRESS = TARGET_RES THEN
  .   .   IF DRCB.RESOURCE_CATEGORY = SUBAREA_LU THEN
  .   .     REMOVE DRCB FROM DRCB_LIST DISCARD;
  .   SCANEND;
  .
  END;

  RETURN;

END CS.LU_FREE;
```

```
CS.ADDLINK_ADDLINKSTA_PROC: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE HANDLES THE ADDITION OF LINK STATION  AND ADJACENT │
 │                 LINK STATION ENTRIES TO THE DOMAIN RESOURCE LIST.     │
 │                                                                       │
 │                 THE FIRST  CHECK MADE  IS TO  DETERMINE IF  THE SSCP  HAS SUFFICIENT │
 │                 RESOURCES TO ASSIGN THE NETWORK ADDRESS  SPECIFIED IN THE ADDLINK OR │
 │                 ADDLINKSTA REQUEST.  IF  NOT, THIS PROCEDURE GENERATES  A SEND_CHECK │
 │                 WITH  AN  APPROPRIATE  ERROR  CODE,  WHICH  IS  SENT  TO │
 │                 UPM_TRANSLATION_SVC.  IF THERE ARE  SUFFICIENT RESOURCES, THE TARGET │
 │                 PU OF THE ADDLINK  OR ADDLINKSTA REQUEST IS CHECKED TO  SEE IF IT IS │
 │                 ACTIVE.  IF  THE  PU  FSM  IS  NOT  ACTIVE,  THE  PROCEDURE │
 │                 RESOURCE_ACTIVE_CHECK,  WHICH  PERFORMS THE  CHECKING,  INSERTS  THE │
 │                 REQUEST INTO THE PU'S SAVE_MU_FOR_RETRY_LIST.  IF THE TARGET PU FSM │
 │                 IS ACTIVE, THE REQUEST IS SENT TO SNS.SEND.           │
 │                                                                       │
 │      INPUT:     ADDLINK OR ADDLINKSTA FROM UPM_TRANSLATION_SVC (CHAPTER 6) │
 │                                                                       │
 │      OUTPUT:    ADDLINK OR  ADDLINKSTA TO  SNS.SEND (CHAPTER  6) AND  A COPY  OF THE │
 │                 TARGET ADDRESS TO  UPM_SAVE_TARGET_NA (PAGE 7-122), OR  A SEND_CHECK │
 │                 WITH AN APPROPRIATE ERROR CODE TO UPM_TRANSLATION_SVC (CHAPTER 6) │
 │                                                                       │
 │      NOTE:      PROCESSING  OF  THIS REQUEST  RESUMES  IN  CS.ADDLINK_ADDLINKSTA_RSP │
 │                 (PAGE 7-107) WHEN SNS RETURNS A RESPONSE.             │
 │                                                                       │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
 │                 SSCP.SVC_MGR.CS.SEND              PAGE 7-48           │
 │                                                                       │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                            │
 │                 RESOURCE_ACTIVE_CHECK             PAGE 7-116          │
 │                 UPM_ADDLINK_RESOURCE_CHECK        PAGE 7-124          │
 │                 UPM_ADDLINKSTA_RESOURCE_CHECK     PAGE 7-124          │
 │                 UPM_SAVE_TARGET_NA                PAGE 7-122          │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL CHECK BIT(4);
  DCL RES_NA BIT(48);

  SELECT ANYORDER(NS_RQ_CODE);
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │              ADDLINK                 │
                    └──────────────────────────────────────┘
                                                                        */
  . WHEN(ADDLINK)
  .   DO;
  .   .
  .   . CHECK = UPM_ADDLINK_RESOURCE_CHECK;          /* PAGE 7-124      */
  .   .
  .   . IF CHECK ¬= X'0000' THEN
  .   .   SEND SEND_CHECK(CHECK) TO UPM_TRANSLATION_SVC;  /* CHAPTER 6    */
  .   .
  .   . ELSE
  .   .   DO;
  .   .   . RES_NA = DSAP||(NSC_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
  .   .   .                                          /* APPENDIX A      */
  .   .   . IF RESOURCE_ACTIVE_CHECK(RES_NA,PU) = OK THEN  /* PAGE 7-116  */
  .   .   .   DO;
  .   .   .   . CALL UPM_SAVE_TARGET_NA(RES_NA);     /* PAGE 7-122      */
  .   .   .   . SEND MU TO SNS.SEND;                 /* CHAPTER 6       */
  .   .   .   END;
  .   .   END;
  .   END;
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │            ADDLINKSTA                │
                    └──────────────────────────────────────┘
                                                                        */
  . WHEN(ADDLINKSTA)
  .   DO;
  .   .
  .   . CHECK = UPM_ADDLINKSTA_RESOURCE_CHECK;       /* PAGE 7-124      */
  .   .
  .   . IF CHECK ¬= X'0000' THEN
  .   .   SEND SEND_CHECK(CHECK) TO UPM_TRANSLATION_SVC;  /* CHAPTER 6    */
  .   .
  .   . ELSE
  .   .   DO;
  .   .   . RES_NA = DSAP||(NSC_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
  .   .   .                                          /* APPENDIX A      */
  .   .   . IF RESOURCE_ACTIVE_CHECK(RES_NA,PU) = OK THEN  /* PAGE 7-116  */
  .   .   .   DO;
  .   .   .   . CALL UPM_SAVE_TARGET_NA(RES_NA);     /* PAGE 7-122      */
  .   .   .   . SEND MU TO SNS.SEND;                 /* CHAPTER 6       */
  .   .   .   END;
  .   .   END;
  .   END;
  END;

END CS.ADDLINK_ADDLINKSTA_PROC;
```

```
CS.ADDLINK_ADDLINKSTA_RSP: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  WHEN THE INPUT IS A POSITIVE  RESPONSE TO ADDLINK OR ADDLINKSTA, THE
  │                 REQUESTED NETWORK  ADDRESS IS ASSIGNED AND  AN ENTRY IS  CREATED AND
  │                 ADDED TO THE DOMAIN RESOURCE LIST.
  │
  │      INPUT:     POSITIVE OR NEGATIVE RESPONSE TO  ADDLINK OR ADDLINKSTA FROM SNS.RCV
  │                 (CHAPTER   6)   AND   A   COPY   OF   THE   TARGET   ADDRESS   FROM
  │                 UPM_RETRIEVE_TARGET_NA (PAGE 7-122)
  │
  │      OUTPUT:    ±RSP(ADDLINK|ADDLINKSTA) TO UPM_TRANSLATION_SVC (CHAPTER 6)
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 SSCP.SVC_MGR.CS.RCV                PAGE 7-50
  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):
  │                 UPM_RETRIEVE_TARGET_NA             PAGE 7-122
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL TARGET_NA BIT(48);

  TARGET_NA = UPM_RETRIEVE_TARGET_NA;                          /* PAGE 7-122    */

  IF RTI = POSITIVE THEN                                       /* RESPONSE IS POSITIVE */

    SELECT ANYORDER(NS_RQ_CODE);
    .
    . WHEN(ADDLINK)
    .   DO;
    .   .
    .   . CREATE DRCB PTR(DRCB_PTR);
    .   .
    .   . DRCB.RESOURCE_CATEGORY = LINK;
    .   .
    .   . DRCB.ASSOCIATED_RES_PTR =
    .   .       FIND_DOMAIN_RESOURCE(TARGET_NA);                 /* APPENDIX B    */
    .   .
    .   . DRCB.NETWORK_ADDRESS =
    .   .       OSAF||(ADDLINK_RSP.LINK_ADDRESS & NCB.NODE_ELEMENT_MASK);
    .   .                                                        /* APPENDIX A    */
    .   . INSERT DRCB IN DRCB_LIST;
    .   .
    .   END;
    .
    .
    . WHEN(ADDLINKSTA)
    .   DO;
    .   .
    .   . CREATE DRCB PTR(DRCB_PTR);
    .   .
    .   . DRCB.RESOURCE_CATEGORY = ALS;
    .   .
    .   . DRCB.ASSOCIATED_RESOURCE_PTR =
    .   .       FIND_SUBORDINATE_DOM_RES(TARGET_NA);             /* APPENDIX B    */
    .   .
    .   . DRCB.NETWORK_ADDRESS =
    .   .       OSAF||(ADDLINKSTA_RSP.ALS_ADDRESS & NCB.NODE_ELEMENT_MASK);
    .   .                                                        /* APPENDIX A    */
    .   . INSERT DRCB IN DRCB_LIST;
    .   .
    .   END;
    .
    END;

  SEND MU TO UPM_TRANSLATION_SVC;                              /* CHAPTER 6     */

END CS.ADDLINK_ADDLINKSTA_RSP;
```

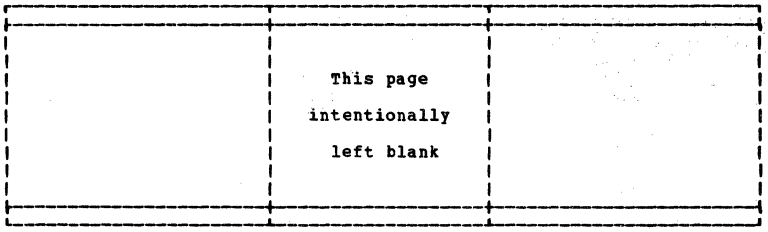
```
CS.DELETENR_PROC: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE  HANDLES THE  DELETION OF  LINK STATION  AND ADJACENT │
 │                 LINK STATION ENTRIES FROM THE DOMAIN RESOURCE LIST.   │
 │                                                                      │
 │                 THE  TARGET RESOURCE'S  ASSOCIATED PU  IS CHECKED  TO SEE  IF IT  IS │
 │                 ACTIVE.    IF   THE   PU   FSM  IS   NOT   ACTIVE,   THE   PROCEDURE │
 │                 RESOURCE_ACTIVE_CHECK,  WHICH  PERFORMS THE  CHECKING,  INSERTS  THE │
 │                 DELETENR REQUEST  INTO THE PU'S  SAVE_MU_FOR_RETRY_LIST.  IF  THE PU │
 │                 FSM IS ACTIVE, PROCESSING OF THE REQUEST CONTINUES IMMEDIATELY.      │
 │                                                                      │
 │                 IF THE  APPROPRIATE FSM'S CORRESPONDING  TO THE TARGET  RESOURCE ARE │
 │                 RESET, A COPY  OF THE DELETENR REQUEST RESOURCE ADDRESS  IS SAVED BY │
 │                 UPM_SAVE_TARGET_NA AND  THE DELETENR  REQUEST IS  SENT TO  SNS.SEND; │
 │                 OTHERWISE, THIS PROCEDURE  GENERATES A SEND_CHECK, WHICH  IS SENT TO │
 │                 UPM_TRANSLATION_SVC.                                 │
 │                                                                      │
 │      INPUT:     DELETENR FROM UPM_TRANSLATION_SVC (CHAPTER 6)         │
 │                                                                      │
 │      OUTPUT:    A COPY OF THE DELETENR  RESOURCE ADDRESS TO UPM_SAVE_TARGET_NA (PAGE │
 │                 7-122) AND  THE DELETENR  TO SNS.SEND (CHAPTER  6); OR  A SEND_CHECK │
 │                 WITH AN APPROPRIATE ERROR CODE TO UPM_TRANSLATION_SVC (CHAPTER 6)    │
 │                                                                      │
 │      NOTE:      PROCESSING OF THIS  REQUEST RESUMES IN CS.DELETENR_RSP  (PAGE 7-109) │
 │                 WHEN SNS RETURNS A RESPONSE.                         │
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 SSCP.SVC_MGR.CS.SEND              PAGE 7-48          │
 │                                                                      │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
 │                 FSM_LINK_ACT_DOM_RES              PAGE 7-129         │
 │                 RESOURCE_ACTIVE_CHECK             PAGE 7-116         │
 │                 SEC_ALS_SUBTREE_CHECK             PAGE 7-121         │
 │                 UPM_SAVE_TARGET_NA                PAGE 7-122         │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RES_NA BIT(48);
  DCL PU_NA BIT(48);
  DCL P POINTER;

  RES_NA = DSAF||(DELETENR_RQ.RESOURCE_ADDRESS & NCB.NODE_ELEMENT_MASK);
                                                    /* APPENDIX A       */
  DRCB_PTR = FIND_DOMAIN_RESOURCE(RES_NA);          /* APPENDIX B       */

  IF DRCB.RESOURCE_CATEGORY ¬= (LINK | ALS | PERIPHERAL_PU) THEN
    SEND SEND_CHECK(X'0806') TO UPM_TRANSLATION_SVC;
                                                    /* RESOURCE UNKNOWN */

  ELSE
    DO;
    . IF DRCB.RESOURCE_CATEGORY = PERIPHERAL_PU THEN
    .   DRCB_PTR = FIND_ALS_FOR_DOM_RES(RES_NA);    /* APPENDIX B       */
    .
    . P = FIND_PU_FOR_DOM_RES(DRCB.NETWORK_ADDRESS); /* APPENDIX B      */
    . PU_NA = P->DRCB.NETWORK_ADDRESS;
    .
    . IF RESOURCE_ACTIVE_CHECK(PU_NA,PU) = OK THEN  /* PAGE 7-116       */
    .
    .   IF (DRCB.RESOURCE_CATEGORY = LINK &
    .        FSM_LINK_ACT_DOM_RES ¬= RESET) |       /* PAGE 7-129       */
    .
    .        (DRCB.RESOURCE_CATEGORY = ALS &
    .        SEC_ALS_SUBTREE_CHECK(DRCB.NETWORK_ADDRESS) = NG) THEN
    .                                               /* PAGE 7-121       */
    .     SEND SEND_CHECK(X'081A') TO UPM_TRANSLATION_SVC;
                                                    /* REQUEST SEQUENCE ERROR */
    .
    .   ELSE
    .     DO;
    .     . CALL UPM_SAVE_TARGET_NA(RES_NA);        /* PAGE 7-122       */
    .     . SEND MU TO SNS.SEND;                    /* CHAPTER 6        */
    .     END;
    END;

END CS.DELETENR_PROC;
```

```
CS.DELETENR_RSP: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  WHEN THE  INPUT IS  A POSITIVE RESPONSE  TO DELETENR,  THE REQUESTED │
 │                 NETWORK ADDRESS  IS REMOVED  FROM THE  DOMAIN RESOURCE  LIST AND  IS │
 │                 DISCARDED.                                           │
 │                                                                      │
 │      INPUT:     POSITIVE OR NEGATIVE  RESPONSE TO DELETENR FROM  SNS.RCV (CHAPTER 6) │
 │                 AND   A    COPY   OF    THE   DELETENR    RESOURCE   ADDRESS    FROM │
 │                 UPM_RETRIEVE_TARGET_NA (PAGE 7-122)                  │
 │                                                                      │
 │      OUTPUT:    ±RSP(DELETENR) TO UPM_TRANSLATION_SVC (CHAPTER 6)    │
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 SSCP.SVC_MGR.CS.RCV              PAGE 7-50           │
 │                                                                      │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
 │                 UPM_RETRIEVE_TARGET_NA           PAGE 7-122          │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RES_NA BIT(48);

  IF RTI = POSITIVE THEN                                  /* RESPONSE IS POSITIVE    */
    DO;
    .
    . RES_NA = UPM_RETRIEVE_TARGET_NA;                    /* PAGE 7-122              */
    .
    . DRCB_PTR = FIND_DOMAIN_RESOURCE(RES_NA);            /* APPENDIX B              */
    .
    . IF DRCB.RESOURCE_CATEGORY = PERIPHERAL_PU THEN
    .   DRCB_PTR = FIND_ALS_FOR_DOM_RES(RES_NA);          /* APPENDIX B              */
    .
    . REMOVE DRCB FROM DRCB_LIST DISCARD;
    .
    END;

  SEND MU TO UPM_TRANSLATION_SVC;                         /* CHAPTER 6               */

END CS.DELETENR_RSP;
```

```
CS.INOP_PROC: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE RESETS THE APPROPRIATE  DOMAIN RESOURCE FSM'S FOR THE │
  │                 TYPE  OF  INOP  RECEIVED.  IT  CALLS  CS.DEACTIVATION_CLEANUP  (PAGE │
  │                 7-119), WHICH GENERATES DACTPU(CLEANUP)  OR DACTLU(CLEANUP) FOR EACH │
  │                 PERIPHERAL PU|LU ASSOCIATED WITH THE TARGET LINK OR ALS.      │
  │                                                                              │
  │      INPUT:     INOP(LINK  | ALS)  FROM SNS.RCV  (CHAPTER 6)  OR FROM  CS.DACTPU_RSP │
  │                 (PAGE 7-56)                                                  │
  │                                                                              │
  │      OUTPUT:    THE  APPROPRIATE FSM'S  ARE  RESET.  THE  INOP  REQUEST  IS SENT  TO │
  │                 UPM_TRANSLATION_SVC (CHAPTER  6).  IF THE  INOP IS  FOR AN ALS  ON A │
  │                 SWITCHED LINK, ABCONN IS GENERATED  AND THE APPROPRIATE PROCEDURE IS │
  │                 CALLED TO PROCESS IT.                                        │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 CS.DACTPU_RSP                    PAGE 7-56                   │
  │                 SSCP.SVC_MGR.CS.RCV              PAGE 7-50                   │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 CS.ALS_SUBTREE_RESET             PAGE 7-113                  │
  │                 CS.CONN_PROC                     PAGE 7-68                   │
  │                 CS.DEACTIVATION_CLEANUP          PAGE 7-119                  │
  │                 CS.LINK_RESET                    PAGE 7-111                  │
  │                 FSM_ALS_CONNECTED_DOM_RES        PAGE 7-133                  │
  │                 FSM_LINK_ACT_DOM_RES             PAGE 7-129                  │
  │                 FSM_LINK_CONNOUT_DOM_RES         PAGE 7-130                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL STATION_NA BIT(48);
  DCL ALS_NA BIT(48);
  DCL LINK_NA BIT(48);
  DCL P POINTER;

  STATION_NA = OSAF||(NSC_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
                                                   /* APPENDIX A              */
  DRCB_PTR = FIND_DOMAIN_RESOURCE(STATION_NA);     /* APPENDIX B              */

  IF DRCB.RESOURCE_CATEGORY = PERIPHERAL_PU THEN
    DRCB_PTR = FIND_ALS_FOR_DOM_RES(STATION_NA);   /* APPENDIX B              */

  IF DRCB.RESOURCE_CATEGORY ¬= (LINK | ALS) THEN
    DO;
    . CALL UPM_LOG;                                /* APPENDIX B              */
    . DISCARD MU;
    END;

  ELSE

    SELECT ANYORDER(DRCB.RESOURCE_CATEGORY);
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │                INOP LINK                 │
                  └──────────────────────────────────────────┘
                                                                                */
    . WHEN(LINK)
    .   DO;
    .   . LINK_NA = DRCB.NETWORK_ADDRESS;
    .   . CALL FSM_LINK_ACT_DOM_RES('RESET');      /* PAGE 7-129              */
    .   . CALL CS.LINK_RESET(LINK_NA);             /* PAGE 7-111              */
    .   END;
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │          INOP ADJACENT LINK STATION      │
                  └──────────────────────────────────────────┘
                                                                                */
    . WHEN(ALS)
    .   DO;
    .   . ALS_NA = DRCB.NETWORK_ADDRESS;
    .   . CALL FSM_ALS_CONNECTED_DOM_RES('RESET');  /* PAGE 7-133             */
    .   . LINK_NA = DRCB.ASSOCIATED_RES_PTR->DRCB.NETWORK_ADDRESS;
    .   . DRCB_PTR = FIND_DOMAIN_RESOURCE(LINK_NA); /* APPENDIX B             */
    .   . CALL FSM_LINK_CONNOUT_DOM_RES('RESET');   /* PAGE 7-130             */
    .   .
    .   . IF DRCB.SWITCHED_LINK = SWITCHED THEN
    .   .   DO;
    .   .   . P = MU_PTR;
    .   .   . MU_PTR = UPM_CREATE_RQ('ABCONN');    /* APPENDIX B             */
    .   .   . DSAP = OSAP;
    .   .   . CALL CS.CONN_PROC;                   /* PAGE 7-68              */
    .   .   . MU_PTR = P;
    .   .   END;
    .   .
    .   . CALL CS.ALS_SUBTREE_RESET(ALS_NA);       /* PAGE 7-113             */
    .   END;
    END;

  CALL CS.DEACTIVATION_CLEANUP(LINK_NA);           /* PAGE 7-119             */

  SEND MU TO UPM_TRANSLATION_SVC;                  /* CHAPTER 6              */

END CS.INOP_PROC;
```

```
CS.LINK_RESET: PROCEDURE(LINK_NA);

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE RESETS THE PRIMARY  OR SECONDARY LINK STATION SUBTREE
 │                 OF THE SPECIFIED LINK.
 │
 │      INPUT:     THE NETWORK ADDRESS OF THE LINK FOR WHICH THE SUBTREE IS TO BE RESET
 │
 │      OUTPUT:    THE FSM'S  ASSOCIATED WITH THE LINK  AND ITS ADJACENT  LINK STATIONS
 │                 ARE RESET.
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 CS.INOP_PROC                        PAGE 7-110
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 CS.ADJ_LINK_STATION_RESET           PAGE 7-111
 │                 FSM_LINK_CONNIN_DOM_RES             PAGE 7-129
 │                 FSM_LINK_CONNOUT_DOM_RES            PAGE 7-130
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL LINK_NA BIT(48);
  DCL SAVE_DRCB_PTR POINTER;

  SAVE_DRCB_PTR = DRCB_PTR;
  DRCB_PTR = FIND_DOMAIN_RESOURCE(LINK_NA);                  /* APPENDIX B      */

  CALL FSM_LINK_CONNOUT_DOM_RES('RESET');                    /* PAGE 7-130      */

  CALL FSM_LINK_CONNIN_DOM_RES('RESET');                     /* PAGE 7-129      */

  CALL CS.ADJ_LINK_STATION_RESET(LINK_NA);                   /* PAGE 7-111      */

  DRCB_PTR = SAVE_DRCB_PTR;

  RETURN;

END CS.LINK_RESET;
```

```
CS.ADJ_LINK_STATION_RESET: PROCEDURE(LINK_NA);

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS  PROCEDURE  SEARCHES  THE  DOMAIN RESOURCE  LIST  TO  FIND  ALL
 │                 ADJACENT LINK STATIONS THAT ARE  ASSOCIATED WITH THE SPECIFIED LINK.
 │                 THE PRIMARY OR  SECONDARY ALS SUBTREE AND  THE ALS_CONNECTED_DOM_RES
 │                 FSM ARE RESET FOR EACH ADJACENT LINK STATION.
 │
 │      INPUT:     THE NETWORK ADDRESS OF THE LINK FOR WHICH ALL CORRESPONDING ADJACENT
 │                 LINK STATIONS ARE TO BE RESET
 │
 │      OUTPUT:    THE  FSM'S  ASSOCIATED  WITH  THE  ADJACENT  LINK  STATIONS  OF  THE
 │                 SPECIFIED LINK ARE RESET.
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 CS.LINK_RESET                       PAGE 7-111
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 CS.ALS_SUBTREE_RESET                PAGE 7-113
 │                 FSM_ALS_CONNECTED_DOM_RES           PAGE 7-133
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL LINK_NA BIT(48);
  DCL SAVE_DRCB_PTR POINTER;

  SAVE_DRCB_PTR = DRCB_PTR;

  SCAN DRCB_LIST PTR(DRCB_PTR);
  .
  . IF DRCB.RESOURCE_CATEGORY = ALS &
  .       DRCB.ASSOCIATED_RES_PTR->DRCB.NETWORK_ADDRESS = LINK_NA THEN
  .    DO;
  .    .
  .    . CALL FSM_ALS_CONNECTED_DOM_RES('RESET');            /* PAGE 7-133      */
  .    .
  .    . CALL CS.ALS_SUBTREE_RESET(DRCB.NETWORK_ADDRESS);    /* PAGE 7-113      */
  .    .
  .    END;
  .
  SCANEND;

  DRCB_PTR = SAVE_DRCB_PTR;

  RETURN;

END CS.ADJ_LINK_STATION_RESET;
```


This page
intentionally
left blank

```
CS.ALS_SUBTREE_RESET: PROCEDURE(ALS_NA);

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS  PROCEDURE  RESETS   THE  PRIMARY  OR  SECONDARY   ALS  SUBTREE │
 │                 ASSOCIATED WITH THE ADDRESS PASSED IN ALS_NA.        │
 │                                                                      │
 │      INPUT:     THE NETWORK  ADDRESS OF AN ADJACENT  LINK STATION FOR WHICH  THE FSM │
 │                 SUBTREE IS TO BE RESET                               │
 │                                                                      │
 │      OUTPUT:    RESET TO ALL FSM'S IN THE ALS SUBTREE                │
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 CS.ADJ_LINK_STATION_RESET          PAGE 7-111        │
 │                 CS.CONN_RSP                        PAGE 7-70         │
 │                 CS.INOP_PROC                       PAGE 7-110        │
 │                                                                      │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
 │                 FSM_ALS_CONTACT_DOM_RES            PAGE 7-130        │
 │                 FSM_ALS_DUMP_DOM_RES               PAGE 7-131        │
 │                 FSM_ALS_IPL_DOM_RES                PAGE 7-131        │
 │                 FSM_ALS_RPO_DOM_RES                PAGE 7-132        │
 │                 FSM_PU_T2_IPL_DOM_RES              PAGE 7-133        │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL ALS_NA BIT(48);
  DCL SAVE_DRCB_PTR POINTER;

  SAVE_DRCB_PTR = DRCB_PTR;
  DRCB_PTR = FIND_ALS_FOR_DOM_RES(ALS_NA);                  /* APPENDIX B      */

  CALL FSM_ALS_CONTACT_DOM_RES('RESET');                    /* PAGE 7-130      */

  IF DRCB.LINK_DLC_ROLE = SECONDARY THEN
    DO;
    .
    . CALL FSM_ALS_IPL_DOM_RES('RESET');                    /* PAGE 7-131      */
    .
    . CALL FSM_ALS_DUMP_DOM_RES('RESET');                   /* PAGE 7-131      */
    .
    . CALL FSM_ALS_RPO_DOM_RES('RESET');                    /* PAGE 7-132      */
    .
    . CALL FSM_PU_T2_IPL_DOM_RES('RESET');                  /* PAGE 7-133      */
    .
    END;

  DRCB_PTR = SAVE_DRCB_PTR;

  RETURN;

END CS.ALS_SUBTREE_RESET;
```

```
CS.REQCONT_REQDISCONT_PROC: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  WHEN REQCONT IS THE INPUT, THE LINK IS A SWITCHED LINK.  │
│                 THE XID CARRIED IN THE REQCONT REQUEST IS INSPECTED. IF  │
│                 THE XID IS VALID AND THE PERTINENT FSM'S ARE IN APPROPRI- │
│                 ATE STATES, THE REQCONT IS SENT TO THE REQCONT FSM AND   │
│                 TO UPM_TRANSLATION_SVC. THE NETWORK ADDRESS FIELD OF THE │
│                 APPROPRIATE PERIPHERAL PU DOMAIN RESOURCE ENTRY IS SET   │
│                 TO THE VALUE OF THE NETWORK ADDRESS OF THE LINK PLUS ONE, │
│                 AND THE ASSOCIATED RESOURCE POINTER FIELD OF THE PU ENTRY │
│                 IS SET TO POINT TO THE ALS ENTRY. SETCV AND CONTACT REQ-  │
│                 UESTS ARE GENERATED AND THE CORRESPONDING PROCEDURES ARE  │
│                 CALLED.                                                  │
│                                                                          │
│                 IF THE XID IS INVALID, -RSP(REQCONT) IS SENT TO SNS.SEND,│
│                 AND DISCONTACT AND ABCONN ARE GENERATED AND THE CORRES-  │
│                 PONDING PROCEDURES ARE CALLED.                           │
│                                                                          │
│                 WHEN REQDISCONT IS THE INPUT, THE REQUEST IS SENT TO     │
│                 SSCP.SVC_MGR.SS.RCV. IN ADDITION, IF THE ASSOCIATED LINK │
│                 IS NONSWITCHED AND THE REQDISCONT REQUEST INDICATES THAT │
│                 A CONTACT IS TO BE SENT IMMEDIATELY AFTER DISCONTACT,    │
│                 THEN THE SEND_CONTACT_IMMEDIATELY FIELD OF THE ADJACENT  │
│                 LINK STATION ENTRY IS SET ACCORDINGLY. THE CONTACT IS    │
│                 GENERATED IN CS.DACTPU_RSP (PAGE 7-56) AFTER OTHER PROC- │
│                 ESSING HAS TAKEN PLACE.                                  │
│                                                                          │
│      INPUT:     REQCONT OR REQDISCONT FROM SNS.RCV (CHAPTER 6)            │
│                                                                          │
│      OUTPUT:    REQCONT TO UPM_TRANSLATION_SVC (CHAPTER 6) AND TO FSM_   │
│                 ALS_CONNECTED_DOM_RES (PAGE 7-133), +RSP(REQCONT) TO     │
│                 SNS.SEND (CHAPTER 6), SETCV TO SNS.SEND, CONTACT TO      │
│                 CS.CONTACT_PROC (PAGE 7-72), AND THE NETWORK ADDRESS     │
│                 FIELD OF THE PERIPHERAL PU AND ALS DOMAIN RESOURCE ENTRIES│
│                 AND THE ASSOCIATED RESOURCE POINTER FIELD OF THE PU ENTRY │
│                 ARE ALL INITIALIZED; OR REQCONT TO UPM_LOG (APPENDIX B), │
│                 -RSP(REQCONT) TO SNS.SEND, AND DISCONTACT AND ABCONN TO  │
│                 THE APPROPRIATE PROCEDURES; OR REQDISCONT TO SSCP.SVC_   │
│                 MGR.SS.RCV (CHAPTER 8)                                   │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 SSCP.SVC_MGR.CS.RCV              PAGE 7-50               │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 CONTACT_DISCONTACT_SEND_CHECK    PAGE 7-118              │
│                 CS.CONN_PROC                     PAGE 7-68               │
│                 CS.CONTACT_PROC                  PAGE 7-72               │
│                 CS.DISCONTACT_PROC               PAGE 7-74               │
│                 FSM_ALS_CONNECTED_DOM_RES        PAGE 7-133              │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

DCL TARGET_NA BIT(48);
DCL PERIPHERAL_PU_NA BIT(48);
DCL PU_PTR POINTER;
DCL ALS_PTR POINTER;

SELECT ANYORDER(NS_RQ_CODE);
```

```
                                                                                /*
                    ┌──────────────────────────────────────┐
                    │               REQCONT                │
                    └──────────────────────────────────────┘
                                                                                */
  . WHEN(REQCONT)
  .   DO;
  .   . TARGET_NA = OSAF||(MSC_RQ.TARGET_ADDRESS & MCB.NODE_ELEMENT_MASK);
  .   . DRCB_PTR = FIND_DOMAIN_RESOURCE(TARGET_NA);              /* APPENDIX B      */
  .   .
  .   . IF (DRCB.RESOURCE_CATEGORY ¬= LINK |
  .   .     DRCB.SWITCHED_LINK ¬= SWITCHED) THEN
  .   .   DO;
  .   .   . CALL UPM_LOG;                                       /* APPENDIX B      */
  .   .   . CALL CHANGE_MU_TO_NEG_RSP(0840);  /* APPENDIX B, PROC INVALID FOR RESOURCE */
  .   .   . SEND MU TO SNS.SEND;                                /* CHAPTER 6       */
  .   .   END;
  .   .
  .   . ELSE
  .   .   DO;
  .   .   . ALS_PTR = FIND_SUBORDINATE_DOM_RES(TARGET_NA);      /* APPENDIX B      */
  .   .   .
  .   .   . FIND DRCB IN DRCB_LIST
  .   .   .      WHERE(DRCB.RESOURCE_CATEGORY = PERIPHERAL_PU &
  .   .   .      DRCB.XID_IMAGE.NODE_ID = REQCONT_RQ.XID_IMAGE.NODE_ID &
  .   .   .      DRCB.XID_IMAGE.PU_TYPE = (PU_T1 | PU_T2));
  .   .   .
  .   .   . IF DRCB_PTR ¬= NULL THEN                            /* XID IS VALID    */
  .   .   .   DO;
  .   .   .   . PU_PTR = DRCB_PTR;
  .   .   .   . DRCB_PTR = ALS_PTR;
  .   .   .   . IF (FSM_ALS_CONNECTED_DOM_RES ¬= RESET |        /* PAGE 7-133      */
  .   .   .   .     CONTACT_DISCONTACT_SEND_CHECK ¬= OK) THEN   /* PAGE 7-118      */
  .   .   .   .   DO;
  .   .   .   .   . CALL UPM_LOG;                              /* APPENDIX B      */
  .   .   .   .   . CALL CHANGE_MU_TO_NEG_RSP(0815);  /* APPENDIX B, FUNCTION ACTIVE */
  .   .   .   .   . SEND MU TO SNS.SEND;                       /* CHAPTER 6       */
  .   .   .   .   END;
  .   .   .   .
  .   .   .   . ELSE
  .   .   .   .   DO;
  .   .   .   .   . CALL FSM_ALS_CONNECTED_DOM_RES;            /* PAGE 7-133      */
  .   .   .   .   . SEND MU TO UPM_TRANSLATION_SVC;            /* CHAPTER 6       */
  .   .   .   .   . PU_PTR->DRCB.ASSOCIATED_RES_PTR = ALS_PTR;
  .   .   .   .   . PU_PTR->DRCB.NETWORK_ADDRESS = TARGET_NA + 1;
  .   .   .   .   .                                           /* PERIPHERAL PU ADDRESS */
  .   .   .   .   . MU_PTR = UPM_CREATE_RSP('REQCONT');       /* APPENDIX B      */
  .   .   .   .   . RTI = POSITIVE;
  .   .   .   .   . SEND MU TO SNS.SEND;                      /* CHAPTER 6       */
  .   .   .   .   . MU_PTR = UPM_CREATE_RQ('SETCV');          /* APPENDIX B      */
  .   .   .   .   . SEND MU TO SNS.SEND;                      /* CHAPTER 6       */
  .   .   .   .   . MU_PTR = UPM_CREATE_RQ('CONTACT');        /* APPENDIX B      */
  .   .   .   .   . DSAF = OSAF;
  .   .   .   .   . CALL CS.CONTACT_PROC;                     /* PAGE 7-72       */
  .   .   .   .   END;
  .   .   .   END;
  .   .   .
  .   .   . ELSE                                              /* XID IS INVALID  */
  .   .   .   DO;
  .   .   .   . CALL UPM_LOG;                                 /* APPENDIX B      */
  .   .   .   . CALL CHANGE_MU_TO_NEG_RSP(0806);  /* APPENDIX B, RESOURCE UNKNOWN */
  .   .   .   . SEND MU TO SNS.SEND;                          /* CHAPTER 6       */
  .   .   .   . MU_PTR = UPM_CREATE_RQ('DISCONTACT');         /* APPENDIX B      */
  .   .   .   . DSAF = OSAF;
  .   .   .   . CALL CS.DISCONTACT_PROC;                      /* PAGE 7-74       */
  .   .   .   . MU_PTR = UPM_CREATE_RQ('ABCONN');             /* APPENDIX B      */
  .   .   .   . DSAF = OSAF;
  .   .   .   . CALL CS.CONN_PROC;                            /* PAGE 7-68       */
  .   .   .   END;
  .   .   END;
  .   END;
                                                                                /*
                    ┌──────────────────────────────────────┐
                    │             REQDISCONT               │
                    └──────────────────────────────────────┘
                                                                                */
  . WHEN(REQDISCONT)
  .   DO;
  .   . PERIPHERAL_PU_NA = OSAF||OEF;
  .   . ALS_PTR = FIND_ALS_FOR_DOM_RES(PERIPHERAL_PU_NA);       /* APPENDIX B      */
  .   .
  .   . IF (ALS_PTR->DRCB.SWITCHED_LINK = NONSWITCHED &
  .   .     REQDISCONT_RQ.SEND_CONTACT_IMMEDIATELY = YES) THEN
  .   .   ALS_PTR->DRCB.SEND_CONTACT_IMMEDIATELY = YES;
  .   . ELSE
  .   .   ALS_PTR->DRCB.SEND_CONTACT_IMMEDIATELY = NO;
  .   .
  .   . SEND MU TO SSCP.SVC_MGR.SS.RCV;                        /* CHAPTER 8       */
  .   END;
  . END;

END CS.REQCONT_REQDISCONT_PROC;
```

```
RESOURCE_ACTIVE_CHECK: PROCEDURE(RES_NA,RES_TYPE) RETURNS(BIT(1));

                                                                        /*
┌──────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE  CHECKS TO SEE IF  THE RESOURCE CORRESPONDING  TO THE │
│                 ADDRESS PASSED IN RES_NA IS ACTIVE.                  │
│                                                                      │
│                 IF THE RESOURCE IS ACTIVE, OK  IS RETURNED TO THE CALLING PROCEDURE. │
│                 OTHERWISE,   THE   REQUEST   IS   INSERTED   INTO   THE   RESOURCE'S │
│                 SAVE_MU_FOR_RETRY_LIST AND NG IS RETURNED TO THE CALLING PROCEDURE. │
│                                                                      │
│      INPUT:     THE NETWORK ADDRESS AND RESOURCE TYPE OF THE DOMAIN RESOURCE THAT IS │
│                 TO BE CHECKED (I.E., PU, LINK, ALS, OR LU)           │
│                                                                      │
│      OUTPUT:    OK IF THE ASSOCIATED DOMAIN RESOURCE IS ACTIVE; OTHERWISE, NG │
│                                                                      │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
│                 CS.ADDLINK_ADDLINKSTA_PROC          PAGE 7-106       │
│                 CS.CONN_PROC                        PAGE 7-68        │
│                 CS.CONTACT_PROC                     PAGE 7-72        │
│                 CS.DELETENR_PROC                    PAGE 7-108       │
│                 CS.DISCONTACT_PROC                  PAGE 7-74        │
│                 CS.DUMP_PROC                        PAGE 7-80        │
│                 CS.FNA_PROC                         PAGE 7-99        │
│                 CS.INITPROC_PROC                    PAGE 7-87        │
│                 CS.LINK_PROC                        PAGE 7-62        │
│                 CS.LOAD_PROC                        PAGE 7-78        │
│                 CS.LU_PROC                          PAGE 7-58        │
│                 CS.PU_PROC                          PAGE 7-52        │
│                 CS.RNAA_PROC                        PAGE 7-94        │
│                 CS.RPO_PROC                         PAGE 7-83        │
│                                                                      │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
│                 FSM_ALS_CONTACT_DOM_RES             PAGE 7-130       │
│                 FSM_LINK_ACT_DOM_RES                PAGE 7-129       │
│                 FSM_LU_ACT_DOM_RES                  PAGE 7-128       │
│                 FSM_PU_ACT_DOM_RES                  PAGE 7-128       │
└──────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL RES_NA BIT(48);
DCL RES_TYPE BIT(4);
DCL CHECK BIT(1);
DCL LIST_PTR POINTER;
DCL SAVE_DRCB_PTR POINTER;

SAVE_DRCB_PTR = DRCB_PTR;
CHECK = NG;
DRCB_PTR = FIND_DOMAIN_RESOURCE(RES_NA);                /* APPENDIX B      */

SELECT ANYORDER(RES_TYPE);
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │                  PU                      │
                  └──────────────────────────────────────────┘
                                                                        */
. WHEN(PU | SUBAREA_PU | PERIPHERAL_PU)
.   DO;
.   . IF FSM_PU_ACT_DOM_RES ¬= ACTIVE THEN              /* PAGE 7-128      */
.   .   DO;
.   .   . IF DRCB.SAVE_MU_FOR_RETRY_LIST = NULL THEN
.   .   .   DO;
.   .   .   . NEWLIST LIST_PTR ENTRY_NAME(MU);
.   .   .   . DRCB.SAVE_MU_FOR_RETRY_LIST = LIST_PTR;
.   .   .   END;
.   .   . INSERT MU IN DRCB.SAVE_MU_FOR_RETRY_LIST;
.   .   END;
.   .
.   . ELSE
.   .   CHECK = OK;
.   END;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │                 LINK                     │
                  └──────────────────────────────────────────┘
                                                                        */
. WHEN(LINK)
.   DO;
.   . IF FSM_LINK_ACT_DOM_RES ¬= ACTIVE THEN            /* PAGE 7-129      */
.   .   DO;
.   .   . IF DRCB.SAVE_MU_FOR_RETRY_LIST = NULL THEN
.   .   .   DO;
.   .   .   . NEWLIST LIST_PTR ENTRY_NAME(MU);
.   .   .   . DRCB.SAVE_MU_FOR_RETRY_LIST = LIST_PTR;
.   .   .   END;
.   .   . INSERT MU IN DRCB.SAVE_MU_FOR_RETRY_LIST;
.   .   END;
.   .
.   . ELSE
.   .   CHECK = OK;
.   END;
```

```
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │                  ALS                     │
                    └──────────────────────────────────────────┘
                                                                        */
   . WHEN(ALS)
   .   DO;
   .   . DRCB_PTR = FIND_ALS_FOR_DOM_RES(RES_NA);          /* APPENDIX B      */
   .   . IF FSM_ALS_CONTACT_DOM_RES ¬= ACTIVE THEN         /* PAGE 7-130      */
   .   .   DO;
   .   .   . IF DRCB.SAVE_MU_FOR_RETRY_LIST = NULL THEN
   .   .   .   DO;
   .   .   .   . NEWLIST LIST_PTR ENTRY_NAME(MU);
   .   .   .   . DRCB.SAVE_MU_FOR_RETRY_LIST = LIST_PTR;
   .   .   .   END;
   .   .   . INSERT MU IN DRCB.SAVE_MU_FOR_RETRY_LIST;
   .   .   END;
   .   .
   .   . ELSE
   .   .   CHECK = OK;
   .   END;
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │                  LU                      │
                    └──────────────────────────────────────────┘
                                                                        */
   . WHEN(LU | SUBAREA_LU | PERIPHERAL_LU)
   .   DO;
   .   . IF FSM_LU_ACT_DOM_RES ¬= ACTIVE THEN              /* PAGE 7-128      */
   .   .   DO;
   .   .   . IF DRCB.SAVE_MU_FOR_RETRY_LIST = NULL THEN
   .   .   .   DO;
   .   .   .   . NEWLIST LIST_PTR ENTRY_NAME(MU);
   .   .   .   . DRCB.SAVE_MU_FOR_RETRY_LIST = LIST_PTR;
   .   .   .   END;
   .   .   . INSERT MU IN DRCB.SAVE_MU_FOR_RETRY_LIST;
   .   .   END;
   .   .
   .   . ELSE
   .   .   CHECK = OK;
   .   END;
   END;

   DRCB_PTR = SAVE_DRCB_PTR;

   RETURN(CHECK);

END RESOURCE_ACTIVE_CHECK;
```

```
CONTACT_DISCONTACT_SEND_CHECK: PROCEDURE(ALS_NA) RETURNS(BIT(1));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE CHECKS THE STATE OF A GROUP OF FSM'S CORRESPONDING TO
   │                 AN ADJACENT LINK STATION.
   │
   │      INPUT:     THE NETWORK ADDRESS OF THE ALS
   │
   │      OUTPUT:    OK IF ALL FSM'S ARE IN APPROPRIATE STATES; ELSE, NG
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 CS.ACTPU_RSP                    PAGE 7-54
   │                 CS.DACTPU_RSP                   PAGE 7-56
   │                 CS.REQCONT_REQDISCONT_PROC      PAGE 7-114
   │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):
   │                 FSM_ALS_CONTACT_DOM_RES         PAGE 7-130
   │                 FSM_ALS_DUMP_DOM_RES            PAGE 7-131
   │                 FSM_ALS_IPL_DOM_RES             PAGE 7-131
   │                 FSM_ALS_RPO_DOM_RES             PAGE 7-132
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL ALS_NA BIT(48);
  DCL CHECK BIT(1);
  DCL SAVE_DRCB_PTR POINTER;

  SAVE_DRCB_PTR = DRCB_PTR;
  CHECK = OK;
  DRCB_PTR = FIND_ALS_FOR_DOM_RES(ALS_NA);                  /* APPENDIX B      */

  IF (FSM_ALS_CONTACT_DOM_RES = ACTIVE) &                   /* PAGE 7-130      */

       ((DRCB.LINK_DLC_ROLE = PRIMARY) |

       (DRCB.LINK_DLC_ROLE = SECONDARY &
       FSM_ALS_IPL_DOM_RES = RESET &                        /* PAGE 7-131      */
       FSM_ALS_DUMP_DOM_RES = RESET &                       /* PAGE 7-131      */
       FSM_ALS_RPO_DOM_RES = RESET)) THEN                   /* PAGE 7-132      */

    CHECK = OK;

  ELSE
    CHECK = NG;

  DRCB_PTR = SAVE_DRCB_PTR;

  RETURN(CHECK);

END CONTACT_DISCONTACT_SEND_CHECK;
```

```
CS.DEACTIVATION_CLEANUP: PROCEDURE(LINK_NA);

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE SCANS THE DOMAIN RESOURCE LIST TO FIND ALL PERIPHERAL │
 │                 PU'S AND  LU'S ASSOCIATED WITH THE  LINK ADDRESS PASSED  IN LINK_NA. │
 │                 FOR  EACH  PERIPHERAL  PU  OR LU  FOUND,  THIS  PROCEDURE  GENERATES │
 │                 DACTPU(CLEANUP)  OR  DACTLU(CLEANUP),  AND   CALLS  THE  APPROPRIATE │
 │                 PROCEDURE TO PROCESS THE RU.                         │
 │                                                                      │
 │      INPUT:     THE NETWORK ADDRESS OF A LINK                        │
 │                                                                      │
 │      OUTPUT:    DACTPU(CLEANUP)  TO CS.PU_PROC  (PAGE  7-52)  OR DACTLU(CLEANUP)  TO │
 │                 CS.LU_PROC (PAGE 7-58)                               │
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 CS.CONTACT_DISCONTACT_RSP          PAGE 7-76         │
 │                 CS.INOP_PROC                       PAGE 7-110        │
 │                                                                      │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
 │                 CS.LU_PROC                         PAGE 7-58         │
 │                 CS.PU_PROC                         PAGE 7-52         │
 │                 FSM_LU_ACT_DOM_RES                 PAGE 7-128        │
 │                 FSM_PU_ACT_DOM_RES                 PAGE 7-128        │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL LINK_NA BIT(48);
  DCL PERIPHERAL_PU_NA BIT(48);
  DCL LINK_PTR POINTER;
  DCL PU_PTR POINTER;
  DCL LU_PTR POINTER;
  DCL SAVE_MU_PTR POINTER;

  SAVE_MU_PTR = MU_PTR;

  SCAN DRCB_LIST PTR(PU_PTR);
  .
  . IF PU_PTR->DRCB.RESOURCE_TYPE = PERIPHERAL_PU THEN
  .   DO;
  .   . PERIPHERAL_PU_NA = PU_PTR->DRCB.NETWORK_ADDRESS;
  .   . LINK_PTR = FIND_LINK_FOR_DOM_RES(PERIPHERAL_PU_NA);      /* APPENDIX B          */
  .   .
  .   . IF LINK_PTR->DRCB.NETWORK_ADDRESS = LINK_NA &
  .   .      FSM_PU_ACT_DOM_RES ¬= RESET THEN
  .   .   DO;
  .   .   . SCAN DRCB_LIST PTR(LU_PTR);
  .   .   . .
  .   .   . . IF LU_PTR->DRCB.RESOURCE_TYPE = PERIPHERAL_LU &
  .   .   . .      LU_PTR->DRCB.ASSOCIATED_RES_PTR->DRCB.NETWORK_ADDRESS = PERIPHERAL_PU_NA &
  .   .   . .      LU_PTR->FSM_LU_ACT_DOM_RES ¬= RESET THEN
  .   .   . .   DO;
  .   .   . .   . MU_PTR = UPM_CREATE_RQ('DACTLU(CLEANUP)');      /* APPENDIX B          */
  .   .   . .   . DSAF = LU_PTR->DRCB.NETWORK_ADDRESS(0:31);
  .   .   . .   . DEF = LU_PTR->DRCB.NETWORK_ADDRESS(32:47);
  .   .   . .   . CALL CS.LU_PROC;                               /* PAGE 7-58           */
  .   .   . .   END;
  .   .   . .
  .   .   . SCANEND;
  .   .   .
  .   .   . MU_PTR = UPM_CREATE_RQ('DACTPU(CLEANUP)');          /* APPENDIX B          */
  .   .   . DSAF = PU_PTR->DRCB.NETWORK_ADDRESS(0:31);
  .   .   . DEF = PU_PTR->DRCB.NETWORK_ADDRESS(32:47);
  .   .   . CALL CS.PU_PROC;                                     /* PAGE 7-52           */
  .   .   END;
  .   END;
  SCANEND;

  MU_PTR = SAVE_MU_PTR;

END CS.DEACTIVATION_CLEANUP;
```

```
SEC_ALS_SUBTREE_INTERRUPT: PROCEDURE(ALS_NA) RETURNS(BIT(1));
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE CHECKS  THAT  THE  SEC_SUBTREE ASSOCIATED  WITH  THE │
   │                 NETWORK ADDRESS PASSED IS IN AN INTERRUPTIBLE STATE.  │
   │                                                                      │
   │      INPUT:     THE NETWORK ADDRESS OF THE ADJACENT LINK STATION TO BE CHECKED │
   │                                                                      │
   │      OUTPUT:    OK IF THE SEC_ALS_SUBTREE IS IN AN  INTERRUPTIBLE STATE; NG IT IT IS │
   │                 NOT                                                  │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 CS.DUMP_PROC                    PAGE 7-80            │
   │                 CS.LOAD_PROC                    PAGE 7-78            │
   │                                                                      │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
   │                 FSM_ALS_DUMP_DOM_RES            PAGE 7-131           │
   │                 FSM_ALS_IPL_DOM_RES             PAGE 7-131           │
   │                 FSM_ALS_RPO_DOM_RES             PAGE 7-132           │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL ALS_NA BIT(48);
  DCL RC BIT(1);
  DCL SAVE_DRCB_PTR POINTER;

  SAVE_DRCB_PTR = DRCB_PTR;
  RC = OK;

  DRCB_PTR = FIND_ALS_FOR_DOM_RES(ALS_NA);                  /* APPENDIX B      */

  IF FSM_ALS_DUMP_DOM_RES = (PEND_INDUMP | PEND_INDUMP_TEXT | PEND_RESET) THEN
                                                            /* PAGE 7-131      */
    RC = NG;

  IF FSM_ALS_IPL_DOM_RES = (PEND_INIPL | PEND_INIPL_TEXT | PEND_RESET) THEN
                                                            /* PAGE 7-131      */
    RC = NG;

  IF FSM_ALS_RPO_DOM_RES = PEND THEN                        /* PAGE 7-132      */
    RC = NG;

  DRCB_PTR = SAVE_DRCB_PTR;

  RETURN(RC);

END SEC_ALS_SUBTREE_INTERRUPT;
```

```
SEC_ALS_SUBTREE_CHECK: PROCEDURE(ALS_NA) RETURNS(BIT(1));

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE  CHECKS TO SEE  THAT THE SEC_SUBTREE  ASSOCIATED WITH
 │                 THE NETWORK ADDRESS PASSED IS IN THE RESET STATE.            │
 │                                                                              │
 │      INPUT:     THE  NETWORK ADDRESS  OF THE  ADJACENT  LINK STATION  FOR WHICH  THE
 │                 SUBTREE IS TO BE CHECKED                                     │
 │                                                                              │
 │      OUTPUT:    OK IF THE SEC_ALS_SUBTREE IS RESET; NG IF IT IS IN ANY OTHER STATE
 │                                                                              │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
 │                 CS.DELETENR_PROC                    PAGE 7-108               │
 │                 CS.FNA_VALIDITY_CHECK               PAGE 7-100               │
 │                 CS.RPO_PROC                         PAGE 7-83                │
 │                                                                              │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
 │                 FSM_ALS_CONTACT_DOM_RES             PAGE 7-130               │
 │                 FSM_ALS_DUMP_DOM_RES                PAGE 7-131               │
 │                 FSM_ALS_IPL_DOM_RES                 PAGE 7-131               │
 │                 FSM_ALS_RPO_DOM_RES                 PAGE 7-132               │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL ALS_NA BIT(48);
  DCL RC BIT(1);
  DCL SAVE_DRCB_PTR POINTER;

  SAVE_DRCB_PTR = DRCB_PTR;
  RC = OK;

  DRCB_PTR = FIND_ALS_FOR_DOM_RES(ALS_NA);                    /* APPENDIX B      */

  IF FSM_ALS_CONTACT_DOM_RES ¬= RESET |                       /* PAGE 7-130      */
       FSM_ALS_IPL_DOM_RES ¬= RESET |                         /* PAGE 7-131      */
       FSM_ALS_DUMP_DOM_RES ¬= RESET |                        /* PAGE 7-131      */
       FSM_ALS_RPO_DOM_RES ¬= RESET THEN                      /* PAGE 7-132      */

    RC = NG;


  DRCB_PTR = SAVE_DRCB_PTR;

  RETURN(RC);

END SEC_ALS_SUBTREE_CHECK;
```

```
UPM_SAVE_TARGET_NA: PROCEDURE(TARGET_NA);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM  ADDS ENTRIES  TO A  CORRELATION TABLE  THAT CONTAINS  FID4 │
   │                 TARGET  ADDRESSES  AND MU  SEQUENCE  NUMBER  FIELDS.  (THE  FID1  RU │
   │                 ADDRESSES  WERE  CONVERTED  INTO  FID4   ADDRESSES  IN  THE  CALLING │
   │                 PROCEDURE AND  THE CONVERTED  FID4 ADDRESS IS  PASSED TO  THIS UPM.) │
   │                 THE REASON FOR SAVING  THE TARGET ADDRESS IS SO THAT  THE ADDRESS IS │
   │                 AVAILABLE WHEN A RESPONSE, WHICH DOES  NOT CARRY THE TARGET ADDRESS, │
   │                 IS RECEIVED.                                         │
   │                                                                      │
   │      INPUT:     THE CURRENT REQUEST                                  │
   │                                                                      │
   │      OUTPUT:    THE TARGET ADDRESS  AND THE SEQUENCE NUMBER FIELD IN  THE CURRENT RQ │
   │                 ARE ADDED TO A CORRELATION TABLE.                    │
   │                                                                      │
   │      NOTE:      UPM_RETRIEVE_TARGET_NA  (PAGE  7-122)  REMOVES  ENTRIES  FROM  THE │
   │                 CORRELATION TABLE.                                   │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 CS.ADDLINK_ADDLINKSTA_PROC          PAGE 7-106       │
   │                 CS.CONN_PROC                        PAGE 7-68        │
   │                 CS.CONTACT_PROC                     PAGE 7-72        │
   │                 CS.DELETENR_PROC                    PAGE 7-108       │
   │                 CS.DISCONTACT_PROC                  PAGE 7-74        │
   │                 CS.DUMP_PROC                        PAGE 7-80        │
   │                 CS.FNA_PROC                         PAGE 7-99        │
   │                 CS.INITPROC_PROC                    PAGE 7-87        │
   │                 CS.LINK_PROC                        PAGE 7-62        │
   │                 CS.LOAD_PROC                        PAGE 7-78        │
   │                 CS.LU_PROC                          PAGE 7-58        │
   │                 CS.PU_PROC                          PAGE 7-52        │
   │                 CS.RNAA_PROC                        PAGE 7-94        │
   │                 CS.RPO_PROC                         PAGE 7-83        │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL TARGET_NA BIT(48);

                                                        /* UNDEFINED PROCEDURE      */

  RETURN;

END UPM_SAVE_TARGET_NA;



UPM_RETRIEVE_TARGET_NA: PROCEDURE RETURNS(BIT(48));

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM  SEARCHES A CORRELATION  TABLE BUILT  BY UPM_SAVE_TARGET_NA │
   │                 (PAGE 7-122)  TO FIND AN  ENTRY IN  WHICH THE SEQUENCE  NUMBER FIELD │
   │                 MATCHES THE SEQUENCE NUMBER FIELD CONTAINED IN THE CURRENT RESPONSE. │
   │                 IT RETURNS TO THE CALLING PROCEDURE  THE TARGET ADDRESS CONTAINED IN │
   │                 THAT ENTRY.  THE ENTRY IS THEN REMOVED FROM THE TABLE.               │
   │                                                                      │
   │      INPUT:     THE CURRENT RESPONSE                                 │
   │                                                                      │
   │      OUTPUT:    THE TARGET ADDRESS OF THE CURRENT RESPONSE'S CORRESPONDING REQUEST    │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 CS.ACTPU_RSP                        PAGE 7-54        │
   │                 CS.ADDLINK_ADDLINKSTA_RSP           PAGE 7-107       │
   │                 CS.CONN_RSP                         PAGE 7-70        │
   │                 CS.CONTACT_DISCONTACT_RSP           PAGE 7-76        │
   │                 CS.DACTPU_RSP                       PAGE 7-56        │
   │                 CS.DELETENR_RSP                     PAGE 7-109       │
   │                 CS.FNA_RSP                          PAGE 7-102       │
   │                 CS.INITPROC_RSP                     PAGE 7-88        │
   │                 CS.LINK_RSP                         PAGE 7-67        │
   │                 CS.LOAD_DUMP_RPO_RSP                PAGE 7-84        │
   │                 CS.LU_ADD                           PAGE 7-98        │
   │                 CS.LU_RSP                           PAGE 7-60        │
   │                 CS.PERIPHERAL_LU_ADD                PAGE 7-97        │
   │                 CS.PERIPHERAL_PU_AND_ALS_ADD        PAGE 7-96        │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL ADDRESS BIT(48);

                                                        /* UNDEFINED PROCEDURE      */

  RETURN(ADDRESS);

END UPM_RETRIEVE_TARGET_NA;
```

```
UPM_RNAA_RESOURCE_CHECK: PROCEDURE RETURNS(BIT(1));

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM CHECKS  TO SEE IF SUFFICIENT STORAGE  AND   │
   │                 NETWORK ADDRESSES ARE AVAILABLE  TO ALLOW  ALL THE   │
   │                 REQUESTED  NETWORK ADDRESSES  TO BE ASSIGNED.  IF    │
   │                 THERE ARE  SUFFICIENT RESOURCES,  IT SETS  THE       │
   │                 RETURN CODE TO OK; OTHERWISE, IT SETS THE RETURN     │
   │                 CODE TO NG.                                          │
   │                                                                      │
   │      INPUT:     RNAA FROM CS.RNAA_PROC (PAGE 7-94)                   │
   │                                                                      │
   │      OUTPUT:    OK, IF THERE ARE SUFFICIENT RESOURCES; OTHERWISE, NG  │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 CS.RNAA_PROC                    PAGE 7-94            │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RC BIT(1);
                                                  /* UNDEFINED PROCEDURE.  */

  RC = OK;

  RETURN(RC);

END UPM_RNAA_RESOURCE_CHECK;



UPM_SLOW_PROC: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM PROCESSES ESLOW AND EXSLOW REQUESTS.        │
   │                                                                      │
   │      INPUT:     ESLOW OR EXSLOW FROM CS.RCV (PAGE 7-50)              │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 SSCP.SVC_MGR.CS.RCV             PAGE 7-50            │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                  /* UNDEFINED PROCEDURE   */

  RETURN;

END UPM_SLOW_PROC;



UPM_ANA_PROC: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM PROCESSES THE ANA REQUEST.                  │
   │                                                                      │
   │      INPUT:     ANA FROM CS.SEND (PAGE 7-48)                         │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 SSCP.SVC_MGR.CS.SEND            PAGE 7-48            │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                  /* UNDEFINED PROCEDURE   */

  RETURN;

END UPM_ANA_PROC;



UPM_ANA_RSP: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM PROCESSES THE ANA RESPONSE.                 │
   │                                                                      │
   │      INPUT:     ±RSP(ANA) FROM CS.RCV (PAGE 7-50)                    │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 SSCP.SVC_MGR.CS.RCV             PAGE 7-50            │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                  /* UNDEFINED PROCEDURE   */

  RETURN;

END UPM_ANA_RSP;
```

```
UPM_NS_LSA_PROC: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM PROCESSES THE NS_LSA REQUEST.               │
   │                                                                      │
   │      INPUT:     NS_LSA FROM CS.RCV (PAGE 7-50)                       │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 SSCP.SVC_MGR.CS.RCV              PAGE 7-50           │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

                                                   /* UNDEFINED PROCEDURE    */

  RETURN;

END UPM_NS_LSA_PROC;



UPM_ADDLINK_RESOURCE_CHECK: PROCEDURE RETURNS(BIT(16));

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM CHECKS  TO SEE IF SUFFICIENT STORAGE  AND NETWORK ADDRESSES
   │                 ARE AVAILABLE AND IF THE LOCAL LINK ID IS VALID.     │
   │                                                                      │
   │      INPUT:     ADDLINK FROM CS.ADDLINK_ADDLINKSTA_PROC (PAGE 7-106)  │
   │                                                                      │
   │      OUTPUT:    IF THE CHECK  IS OK, THE RETURN  CODE IS SET TO  X'0000'; OTHERWISE,
   │                 THE RETURN CODE  IS SET TO X'0806' (RESOURCE UNKNOWN)  OR TO X'0812'
   │                 (INSUFFICIENT RESOURCES).                            │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 CS.ADDLINK_ADDLINKSTA_PROC       PAGE 7-106          │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RETURN_CODE BIT(16);

                                                   /* UNDEFINED PROCEDURE    */

  RETURN_CODE = X'0000';

  RETURN(RETURN_CODE);

END UPM_ADDLINK_RESOURCE_CHECK;



UPM_ADDLINKSTA_RESOURCE_CHECK: PROCEDURE RETURNS(BIT(16));

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM CHECKS  TO SEE IF SUFFICIENT STORAGE  AND NETWORK ADDRESSES
   │                 ARE AVAILABLE AND IF THE FID TYPE IS CORRECTLY SPECIFIED.
   │                                                                      │
   │      INPUT:     ADDLINK FROM CS.ADDLINK_ADDLINKSTA_PROC (PAGE 7-106)  │
   │                                                                      │
   │      OUTPUT:    IF THE CHECK  IS OK, THE RETURN  CODE IS SET TO  X'0000'; OTHERWISE,
   │                 THE RETURN CODE  IS SET TO X'0806' (RESOURCE UNKNOWN)  OR TO X'0812'
   │                 (INSUFFICIENT RESOURCES) OR TO X'0835' (INVALID PARAMETER).
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 CS.ADDLINK_ADDLINKSTA_PROC       PAGE 7-106          │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RETURN_CODE BIT(16);

                                                   /* UNDEFINED PROCEDURE    */

  RETURN_CODE = X'0000';

  RETURN(RETURN_CODE);

END UPM_ADDLINKSTA_RESOURCE_CHECK;
```

```
UPM_SAVE_FNA_RQ: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM ADDS  THE CURRENT FNA REQUEST  TO A TABLE.  THE  PURPOSE OF │
   │                 THIS IS TO SAVE THE FNA REQUEST  SO THAT WHEN A POSITIVE RESPONSE TO │
   │                 FNA  IS RETURNED,  THE  NETWORK ADDRESSES  OF  THE  RESOURCES TO  BE │
   │                 DELETED FROM THE DOMAIN RESOURCE LIST ARE AVAILABLE TO THE SSCP.     │
   │                                                                      │
   │      INPUT:     THE CURRENT FNA REQUEST                              │
   │                                                                      │
   │      OUTPUT:    THE FNA REQUEST IS ADDED TO A TABLE.                 │
   │                                                                      │
   │      NOTE:      UPM_RETRIEVE_FNA_RQ (PAGE 7-125) REMOVES REQUESTS FROM THE TABLE.  │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 CS.FNA_PROC                          PAGE 7-99       │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

                                                  /* UNDEFINED PROCEDURE    */

  RETURN;

END UPM_SAVE_FNA_RQ;



UPM_RETRIEVE_FNA_RQ: PROCEDURE(FNA_RQ_COPY);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM SEARCHES  A TABLE BUILT BY UPM_SAVE_FNA_RQ  (PAGE 7-125) TO │
   │                 FIND THE FNA REQUEST CORRESPONDING TO  THE CURRENT FNA RESPONSE.  IT │
   │                 RETURNS TO THE  CALLING PROCEDURE THE APPROPRIATE  FNA REQUEST.  THE │
   │                 REQUEST IS THEN REMOVED FROM THE TABLE.              │
   │                                                                      │
   │      INPUT:     THE CURRENT FNA RESPONSE                             │
   │                                                                      │
   │      OUTPUT:    THE FNA REQUEST CORRESPONDING TO THE CURRENT RESPONSE │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 CS.FNA_RSP                           PAGE 7-102      │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL 1 FNA_RQ_COPY,
        2 NS_REQUEST_CODE BIT(24),
        2 TARGET_ADDRESS BIT(16),
        2 ENTRY_CNT BIT(8),
        2 TYPE BIT(8),
        2 SUBFIELD(40) BIT(16);

                                                  /* UNDEFINED PROCEDURE    */

  RETURN;

END UPM_RETRIEVE_FNA_RQ;



UPM_SAVE_RNAA_RQ: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM ADDS THE  CURRENT RNAA REQUEST TO A TABLE.   THE PURPOSE OF │
   │                 THIS IS TO SAVE THE RNAA REQUEST SO THAT WHEN A POSITIVE RESPONSE TO │
   │                 RNAA IS RETURNED,  THE LOCAL ADDRESSES OF THE PERIPHERAL  LU'S TO BE │
   │                 DELETED FROM THE DOMAIN RESOURCE LIST ARE AVAILABLE TO THE SSCP.     │
   │                                                                      │
   │      INPUT:     THE CURRENT RNAA REQUEST                             │
   │                                                                      │
   │      OUTPUT:    THE RNAA REQUEST IS ADDED TO A TABLE.                │
   │                                                                      │
   │      NOTE:      UPM_RETRIEVE_RNAA_RQ (PAGE 7-126) REMOVES REQUESTS FROM THE TABLE. │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 CS.RNAA_PROC                         PAGE 7-94       │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

                                                  /* UNDEFINED PROCEDURE    */

  RETURN;

END UPM_SAVE_RNAA_RQ;
```

```
UPM_RETRIEVE_RNAA_RQ: PROCEDURE(RNAA_RQ_COPY);

                                                                              /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM SEARCHES A TABLE BUILT  BY UPM_SAVE_RNAA_RQ (PAGE 7-125) TO
   │                 FIND THE  RNAA REQUEST CORRESPONDING  TO THE CURRENT  RNAA RESPONSE.
   │                 IT RETURNS  TO THE CALLING  PROCEDURE THE APPROPRIATE  RNAA REQUEST.
   │                 THE REQUEST IS THEN REMOVED FROM THE TABLE.
   │
   │      INPUT:     THE CURRENT RNAA RESPONSE
   │
   │      OUTPUT:    THE RNAA REQUEST CORRESPONDING TO THE CURRENT RESPONSE
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 CS.PERIPHERAL_LU_ADD                PAGE 7-97
   └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL 1 RNAA_RQ_COPY,
        2 NS_REQUEST_CODE BIT(24),
        2 TARGET_ADDRESS BIT(16),
        2 ASSIGNMENT_TYPE BIT(8),
        2 ENTRY_CNT BIT(8),
        2 SUBFIELD(40) BIT(16);

                                                        /* UNDEFINED PROCEDURE      */
  RETURN;

END UPM_RETRIEVE_RNAA_RQ;



UPM_MANUAL_DIAL: PROCEDURE;

                                                                              /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE SENDS  TO THE OPERATOR THE PHONE NUMBER  TO BE DIALED
   │                 DURING THE CURRENT MANUAL CONNECT OUT SEQUENCE.
   │
   │      INPUT:     THE PHONE NUMBER OF THE LINK TO BE DIALED
   │
   │      OUTPUT:    THE PHONE NUMBER IS SENT TO THE OPERATOR
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 CS.CONN_PROC                        PAGE 7-68
   └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL DIAL_DIGITS CHAR(20);

                                                        /* UNDEFINED PROCEDURE      */
  RETURN;

END UPM_MANUAL_DIAL;



UPM_CAN_SSCP_IPL_PU_T2: PROCEDURE(PU_T2_NA) RETURNS(BIT(1));

                                                                              /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM  DETERMINES WHETHER THE  SSCP HAS  THE ABILITY TO  LOAD THE
   │                 PU_T2 NODE.
   │
   │      INPUT:     NETWORK ADDRESS OF THE PU_T2 IN THE NODE TO BE LOADED
   │
   │      OUTPUT:    A RETURN CODE OF  YES IF THE SSCP CAN LOAD THE  PU_T2 NODE; A RETURN
   │                 CODE OF NO IF THE SSCP CANNOT IPL THE PU_T2
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 CS.INITIATE_IPL_PROC                PAGE 7-91
   │                 CS.INITPROC_RSP                     PAGE 7-88
   └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL RETURN_VALUE BIT(1);
  DCL PU_T2_NA BIT(48);
                                                        /* UNDEFINED PROCEDURE      */
  RETURN_VALUE = YES;

  RETURN(RETURN_VALUE);

END UPM_CAN_SSCP_IPL_PU_T2;
```

```
UPM_BUILD_TEXT_OR_FINAL: PROCEDURE RETURNS(PTR);
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  THIS  PROCEDURE  CREATES  AN NS_IPL_TEXT  OR  NS_IPL_FINAL  REQUEST.  │
   │                NS_IPL_TEXT IS CREATED  IF THERE IS MORE IPL TEXT  TO BE TRANSMITTED  │
   │                TO  THE  PU_T2  NODE.   IF  ALL OF  THE  IPL  TEXT  HAS  BEEN  SENT,  │
   │                NS_IPL_FINAL IS CREATED.                              │
   │                                                                      │
   │     INPUT:     NONE                                                  │
   │                                                                      │
   │     OUTPUT:    POINTER TO AN NS_IPL_TEXT OR NS_IPL_FINAL REQUEST     │
   │                                                                      │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
   │                CS.PU_T2_LOAD_RSP                 PAGE 7-92           │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL NEW_PTR POINTER;

  CREATE NEW_PTR->NU;
                                                    /* UNDEFINED PROCEDURE   */
  RETURN(NEW_PTR);

END UPM_BUILD_TEXT_OR_FINAL;



UPM_ER_VR_INOP_PROC: PROCEDURE;
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  THIS UPM PROCESSES ER_INOP AND VR_INOP REQUESTS.      │
   │                                                                      │
   │     INPUT:     ER_INOP OR VR_INOP FROM CS.RCV (PAGE 7-50)            │
   │                                                                      │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
   │                SSCP.SVC_MGR.CS.RCV               PAGE 7-50           │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                    /* UNDEFINED PROCEDURE   */
  RETURN;

END UPM_ER_VR_INOP_PROC;



UPM_LCP_PROC: PROCEDURE;
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  THIS UPM PROCESSES THE LCP REQUEST.                   │
   │                                                                      │
   │     INPUT:     LCP FROM CS.RCV (PAGE 7-50)                           │
   │                                                                      │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
   │                SSCP.SVC_MGR.CS.RCV               PAGE 7-50           │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                    /* UNDEFINED PROCEDURE   */
  RETURN;

END UPM_LCP_PROC;
```

```
FSM_PU_ACT_DOM_RES: FSM_DEFINITION CONTEXT(DMCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO REMEMBER THE STATUS OF A PHYSICAL  UNIT WITH RESPECT TO ACTPU AND │
│                DACTPU REQUESTS.                                          │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                CS.ACTPU_RSP                    PAGE 7-54                 │
│                CS.DACTPU_RSP                   PAGE 7-56                 │
│                CS.DEACTIVATION_CLEANUP         PAGE 7-119                │
│                CS.LDREQD_PROC                  PAGE 7-86                 │
│                CS.PROCSTAT_PROC                PAGE 7-89                 │
│                CS.PU_PROC                      PAGE 7-52                 │
│                CS.PU_T2_IPL_ABORT              PAGE 7-93                 │
│                CS.PU_T2_LOAD_RSP               PAGE 7-92                 │
│                RESOURCE_ACTIVE_CHECK           PAGE 7-116                │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES------>	RESET	PEND ACTIVE	PEND IPL	ACTIVE	PEND RESET
INPUT STATE NUMBERS---->	01	02	03	04	05
S,ACTPU	2	/	/	/	/
R,+RSP(ACTPU,IPL_REQUIRED)	/	3	/	/	-
R,+RSP(ACTPU)	/	4	/	/	-
R,-RSP(ACTPU)	/	1	/	/	-
DACTPU	/	5	5	5	-
R,±RSP(DACTPU)	-	/	/	/	1
R,+RSP(NS_IPL_FINAL)	/	/	4	-	-
R,PROCSTAT(IPL_SUCCESSFUL)	/	/	4	-	-
'RESET'	-	1	1	1	1

```
END FSM_PU_ACT_DOM_RES;

FSM_LU_ACT_DOM_RES: FSM_DEFINITION CONTEXT(DMCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO REMEMBER THE STATUS  OF A LOGICAL UNIT WITH RESPECT  TO ACTLU AND │
│                DACTLU REQUESTS.                                          │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                CS.DEACTIVATION_CLEANUP         PAGE 7-119                │
│                CS.LU_PROC                      PAGE 7-58                 │
│                CS.LU_RSP                       PAGE 7-60                 │
│                CS.PERIPHERAL_LU_ADD            PAGE 7-97                 │
│                RESOURCE_ACTIVE_CHECK           PAGE 7-116                │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES------>	RESET	PEND ACTIVE	ACTIVE	PEND RESET
INPUT STATE NUMBERS---->	01	02	03	04
S,ACTLU	2	/	/	/
R,+RSP(ACTLU)	/	3	/	-
R,-RSP(ACTLU)	/	1	/	-
DACTLU	/	4	4	-
R,±RSP(DACTLU)	-	/	/	1
'RESET'	-	1	1	1

```
END FSM_LU_ACT_DOM_RES;
```

```
FSM_LINK_ACT_DOM_RES: FSM_DEFINITION CONTEXT(DRCB);

                                                                        /*
   FUNCTION:  TO  REMEMBER THE  STATUS  OF  A LINK  WITH  RESPECT  TO ACTLINK  AND
              DACTLINK REQUESTS.

   REFERENCED BY THE FOLLOWING PROCEDURE(S):
              CS.DELETENR_PROC                    PAGE 7-108
              CS.INOP_PROC                        PAGE 7-110
              CS.LINK_PROC                        PAGE 7-62
              CS.LINK_RSP                         PAGE 7-67
              RESOURCE_ACTIVE_CHECK               PAGE 7-116
                                                                        */
```

STATE NAMES----->	RESET	PEND ACTIVE	ACTIVE	PEND RESET
INPUT STATE NUMBERS--->	01	02	03	04
S,ACTLINK	2	/	/	/
R,+RSP(ACTLINK)	/	3	/	-
R,-RSP(ACTLINK)	/	1	/	-
S,DACTLINK	/	4	4	/
R,+RSP(DACTLINK)	/	/	/	1
R,-RSP(DACTLINK)	/	/	/	3
'RESET'	-	1	1	1

```
END FSM_LINK_ACT_DOM_RES;

FSM_LINK_CONNIN_DOM_RES: FSM_DEFINITION CONTEXT(DRCB);

                                                                        /*
   FUNCTION:  TO REMEMBER THE STATUS OF A  SWITCHED LINK WITH RESPECT TO ACTCONNIN
              AND DACTCONNIN REQUESTS ASSOCIATED WITH DIAL-IN.

   REFERENCED BY THE FOLLOWING PROCEDURE(S):
              CS.CONN_PROC                        PAGE 7-68
              CS.CONN_RSP                         PAGE 7-70
              CS.DACTLINK_SEND_CHECKS             PAGE 7-64
              CS.LINK_RESET                       PAGE 7-111
                                                                        */
```

STATE NAMES----->	RESET	PEND ACTIVE	ACTIVE	PEND RESET
INPUT STATE NUMBERS--->	01	02	03	04
S,ACTCONNIN	2	/	/	/
R,+RSP(ACTCONNIN)	/	3	/	/
R,-RSP(ACTCONNIN)	/	1	/	/
S,DACTCONNIN	/	/	4	/
R,+RSP(DACTCONNIN)	/	/	/	1
R,-RSP(DACTCONNIN)	/	/	/	3
'RESET'	-	1	1	1

```
END FSM_LINK_CONNIN_DOM_RES;
```

```
FSM_LINK_CONNOUT_DOM_RES: FSM_DEFINITION CONTEXT(DRCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO REMEMBER  THE STATUS OF A  SWITCHED LINK WITH RESPECT  TO CONNOUT
│                AND ABCONNOUT REQUESTS ASSOCIATED WITH DIAL-OUT.          │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                CS.CONN_PROC                        PAGE 7-68             │
│                CS.CONN_RSP                         PAGE 7-70             │
│                CS.DACTLINK_SEND_CHECKS             PAGE 7-64             │
│                CS.INOP_PROC                        PAGE 7-110            │
│                CS.LINK_RESET                       PAGE 7-111            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----->	RESET	PEND ACTIVE	ACTIVE	PEND RESET
INPUT STATE NUMBERS--->	01	02	03	04
S,CONNOUT	2	/	/	/
R,+RSP(CONNOUT)	/	3	/	/
R,-RSP(CONNOUT)	/	1	/	/
S,ABCONNOUT	/	/	4	/
R,+RSP(ABCONNOUT)	/	/	/	1
R,-RSP(ABCONNOUT)	/	/	/	3
'RESET'	-	1	1	1

```
END FSM_LINK_CONNOUT_DOM_RES;

FSM_ALS_CONTACT_DOM_RES: FSM_DEFINITION CONTEXT(DRCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO REMEMBER THE  STATUS OF AN ADJACENT LINK STATION  WITH RESPECT TO
│                CONTACT AND DISCONTACT REQUESTS.                          │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                CONTACT_DISCONTACT_SEND_CHECK       PAGE 7-118            │
│                CS.ALS_SUBTREE_RESET                PAGE 7-113            │
│                CS.CONTACT_DISCONTACT_RSP           PAGE 7-76             │
│                CS.CONTACT_PROC                     PAGE 7-72             │
│                CS.CONTACTED_PROC                   PAGE 7-77             │
│                CS.DACTLINK_SEND_CHECKS             PAGE 7-64             │
│                CS.DISCONTACT_PROC                  PAGE 7-74             │
│                RESOURCE_ACTIVE_CHECK               PAGE 7-116            │
│                SEC_ALS_SUBTREE_CHECK               PAGE 7-121            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----->	RESET	PEND ACTIVE RSP	PEND ACTIVE CONTACTED	ACTIVE	PEND RESET CONTACTED	PEND RESET RSP
INPUT STATE NUMBERS--->	01	02	03	04	05	06
S,CONTACT	2	/	/	/	/	/
R,+RSP(CONTACT)	/	3	/	/	/	/
R,-RSP(CONTACT)	/	1	/	/	/	/
R,CONTACTED(ERROR)	/	/	1	/	1	/
R,CONTACTED(LOAD_REQUIRED)	/	/	1	/	1	/
R,CONTACTED	/	/	4	/	6	/
S,DISCONTACT	-	/	5	6	/	/
R,+RSP(DISCONTACT)	-	/	/	/	1	1
R,-RSP(DISCONTACT,0822)	-	/	/	/	1	1
R,-RSP(DISCONTACT)	-	/	/	/	1	4
'RESET'	-	1	1	1	1	1

```
END FSM_ALS_CONTACT_DOM_RES;
```

```
FSM_ALS_DUMP_DOM_RES: FSM_DEFINITION CONTEXT(DRCB);

/*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO REMEMBER  THE STATUS  OF A SECONDARY  ADJACENT LINK  STATION WITH
│                RESPECT TO DUMPINIT, DUMPTEXT, AND DUMPFINAL REQUESTS.
│
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                CONTACT_DISCONTACT_SEND_CHECK      PAGE 7-118
│                CS.ALS_SUBTREE_RESET               PAGE 7-113
│                CS.CONTACT_PROC                    PAGE 7-72
│                CS.DACTLINK_SEND_CHECKS            PAGE 7-64
│                CS.DISCONTACT_PROC                 PAGE 7-74
│                CS.DUMP_PROC                       PAGE 7-80
│                CS.LOAD_DUMP_RPO_RSP               PAGE 7-84
│                SEC_ALS_SUBTREE_CHECK              PAGE 7-121
│                SEC_ALS_SUBTREE_INTERRUPT          PAGE 7-120
└──────────────────────────────────────────────────────────────────────────┘
*/
```

STATE NAMES----->	RESET	PEND INDUMP	INDUMP	PEND INDUMP TEXT	PEND RESET
INPUT STATE NUMBERS--->	01	02	03	04	05
S,DUMPINIT	2	-	2	2	2
R,+RSP(DUMPINIT)	/	3	/	/	/
R,-RSP(DUMPINIT)	/	1	/	/	/
S,DUMPTEXT	/	/	4	/	/
R,±RSP(DUMPTEXT)	/	-	/	3	/
S,DUMPFINAL	/	/	5	/	/
R,+RSP(DUMPFINAL)	/	-	/	/	1
R,-RSP(DUMPFINAL)	/	-	/	/	3
'RESET'	-	1	1	1	1

```
END FSM_ALS_DUMP_DOM_RES;

FSM_ALS_IPL_DOM_RES: FSM_DEFINITION CONTEXT(DRCB);

/*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO REMEMBER  THE STATUS  OF A SECONDARY  ADJACENT LINK  STATION WITH
│                RESPECT TO IPLINIT, IPLTEXT, AND IPLFINAL REQUESTS.
│
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                CONTACT_DISCONTACT_SEND_CHECK      PAGE 7-118
│                CS.ALS_SUBTREE_RESET               PAGE 7-113
│                CS.CONTACT_PROC                    PAGE 7-72
│                CS.DACTLINK_SEND_CHECKS            PAGE 7-64
│                CS.DISCONTACT_PROC                 PAGE 7-74
│                CS.LOAD_DUMP_RPO_RSP               PAGE 7-84
│                CS.LOAD_PROC                       PAGE 7-78
│                SEC_ALS_SUBTREE_CHECK              PAGE 7-121
│                SEC_ALS_SUBTREE_INTERRUPT          PAGE 7-120
└──────────────────────────────────────────────────────────────────────────┘
*/
```

STATE NAMES----->	RESET	PEND INIPL	INIPL	PEND INIPL TEXT	PEND RESET
INPUT STATE NUMBERS--->	01	02	03	04	05
S,IPLINIT	2	-	2	2	2
R,+RSP(IPLINIT)	/	3	/	/	/
R,-RSP(IPLINIT)	/	1	/	/	/
S,IPLTEXT	/	/	4	/	/
R,±RSP(IPLTEXT)	/	-	/	3	/
S,IPLFINAL	/	/	5	/	/
R,+RSP(IPLFINAL)	/	-	/	/	1
R,-RSP(IPLFINAL)	/	-	/	/	3
'RESET'	-	1	1	1	1

```
END FSM_ALS_IPL_DOM_RES;
```

```
FSM_PROC_DOM_RES: FSM_DEFINITION CONTEXT(DRCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO REMEMBER THE STATUS OF A PU_T4  OR PU_T5 WITH RESPECT TO INITPROC
│                AND PROCSTAT REQUESTS.
│
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                CS.INITPROC_PROC                    PAGE 7-87
│                CS.INITPROC_RSP                     PAGE 7-88
│                CS.PROCSTAT_PROC                    PAGE 7-89
│                SSCP.SVC_MGR.CS.SEND                PAGE 7-48
└──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----->	RESET	PEND ACTIVE	ACTIVE	PEND RESET
INPUT STATE NUMBERS--->	01	02	03	04
S,INITPROC	2	/	/	/
R,+RSP(INITPROC)	/	3	/	/
R,-RSP(INITPROC)	/	1	/	/
R,PROCSTAT	/	/	4	/
S,±RSP(PROCSTAT)	/	/	/	1
'RESET'	-	1	1	1

```
END FSM_PROC_DOM_RES;

FSM_ALS_RPO_DOM_RES: FSM_DEFINITION CONTEXT(DRCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO REMEMBER  THE STATUS  OF A SECONDARY  ADJACENT LINK  STATION WITH
│                RESPECT TO RPO REQUESTS.
│
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                CONTACT_DISCONTACT_SEND_CHECK       PAGE 7-118
│                CS.ALS_SUBTREE_RESET                PAGE 7-113
│                CS.CONTACT_PROC                     PAGE 7-72
│                CS.DACTLINK_SEND_CHECKS             PAGE 7-64
│                CS.DISCONTACT_PROC                  PAGE 7-74
│                CS.LOAD_DUMP_RPO_RSP                PAGE 7-84
│                CS.RPO_PROC                         PAGE 7-83
│                SEC_ALS_SUBTREE_CHECK               PAGE 7-121
│                SEC_ALS_SUBTREE_INTERRUPT           PAGE 7-120
└──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----->	RESET	PEND
INPUT STATE NUMBERS--->	01	02
S,RPO	2	/
R,±RSP(RPO)	/	1
'RESET'	-	1

```
END FSM_ALS_RPO_DOM_RES;
```

```
FSM_ALS_CONNECTED_DOM_RES: FSM_DEFINITION CONTEXT(DRCB);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  TO REMEMBER THE CONNECTED STATUS OF AN ADJACENT LINK  │
   │                STATION THAT IS ON A SWITCHED LINK.                   │
   │                                                                      │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
   │                CS.ADJ_LINK_STATION_RESET          PAGE 7-111         │
   │                CS.CONN_PROC                       PAGE 7-68          │
   │                CS.CONN_RSP                        PAGE 7-70          │
   │                CS.DACTLINK_SEND_CHECKS            PAGE 7-64          │
   │                CS.DUMP_PROC                       PAGE 7-80          │
   │                CS.INOP_PROC                       PAGE 7-110         │
   │                CS.LOAD_PROC                       PAGE 7-78          │
   │                CS.REQCONT_REQDISCONT_PROC         PAGE 7-114         │
   │                CS.RPO_PROC                        PAGE 7-83          │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----->	RESET	ACTIVE	PEND RESET
INPUT STATE NUMBERS--->	01	02	03
R,REQCONT	2	/	/
S,+RSP(REQCONT)	/	-	/
S,ABCONN	-	3	-
R,+RSP(ABCONN)	-	/	1
R,-RSP(ABCONN)	-	/	2
'RESET'	-	1	1

```
END FSM_ALS_CONNECTED_DOM_RES;


FSM_PU_T2_IPL_DOM_RES: FSM_DEFINITION CONTEXT(DRCB);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  TO REMEMBER THE STATUS OF A  PU_T2 NODE WITH RESPECT  │
   │                TO NS_IPL_INIT, NS_IPL_TEXT, NS_IPL_FINAL, AND       │
   │                NS_IPL_ABORT REQUESTS.                                │
   │                                                                      │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
   │                CS.ALS_SUBTREE_RESET               PAGE 7-113         │
   │                CS.INITIATE_IPL_PROC               PAGE 7-91          │
   │                CS.PU_T2_LOAD_RSP                  PAGE 7-92          │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----->	RESET	TEXT
INPUT STATE NUMBERS--->	01	02
S,NS_IPL_INIT	2	/
S,NS_IPL_TEXT	/	-
S,NS_IPL_FINAL	/	1
S,NS_IPL_ABORT	/	1
'RESET'	-	1

```
END FSM_PU_T2_IPL_DOM_RES;
```

```
FSM_INPUT_DEFINITION:

 ABCONN                      NS_RQ_CODE=ABCONN        & RRI=RQ;
 ABCONNOUT                   NS_RQ_CODE=ABCONNOUT     & RRI=RQ;
 ACTCONNIN                   NS_RQ_CODE=ACTCONNIN     & RRI=RQ;
 ACTLINK                     NS_RQ_CODE=ACTLINK       & RRI=RQ;
 ACTLU                       RQ_CODE=ACTLU            & RRI=RQ;
 ACTPU                       RQ_CODE=ACTPU            & RRI=RQ;
 CONNOUT                     NS_RQ_CODE=CONNOUT       & RRI=RQ;
 CONTACT                     NS_RQ_CODE=CONTACT       & RRI=RQ;
 CONTACTED                   NS_RQ_CODE=CONTACTED     & RRI=RQ;
 CONTACTED(LOAD_REQUIRED)    NS_RQ_CODE=CONTACTED     & RRI=RQ &
                               CONTACTED_RQ.STATUS=LOAD_REQUIRED;
 CONTACTED(ERROR)            NS_RQ_CODE=CONTACTED     & RRI=RQ &
                               CONTACTED_RQ.STATUS=ERROR;
 DACTCONNIN                  NS_RQ_CODE=DACTCONNIN    & RRI=RQ;
 DACTLINK                    NS_RQ_CODE=DACTLINK      & RRI=RQ;
 DACTLU                      NS_RQ_CODE=DACTLU        & RRI=RQ;
 DACTPU                      RQ_CODE=DACTPU           & RRI=RQ;
 DISCONTACT                  NS_RQ_CODE=DISCONTACT    & RRI=RQ;
 DUMPFINAL                   NS_RQ_CODE=DUMPFINAL     & RRI=RQ;
 DUMPINIT                    NS_RQ_CODE=DUMPINIT      & RRI=RQ;
 DUMPTEXT                    NS_RQ_CODE=DUMPTEXT      & RRI=RQ;
 INITPROC                    NS_RQ_CODE=INITPROC      & RRI=RQ;
 IPLFINAL                    NS_RQ_CODE=IPLFINAL      & RRI=RQ;
 IPLINIT                     NS_RQ_CODE=IPLINIT       & RRI=RQ;
 IPLTEXT                     NS_RQ_CODE=IPLTEXT       & RRI=RQ;
 NS_IPL_ABORT                NS_RQ_CODE=NS_IPL_ABORT  & RRI=RQ;
 NS_IPL_FINAL                NS_RQ_CODE=NS_IPL_FINAL  & RRI=RQ;
 NS_IPL_INIT                 NS_RQ_CODE=NS_IPL_INIT   & RRI=RQ;
 NS_IPL_TEXT                 NS_RQ_CODE=NS_IPL_TEXT   & RRI=RQ;
 PROCSTAT                    NS_RQ_CODE=PROCSTAT      & RRI=RQ;
 PROCSTAT(IPL_SUCCESSFUL)    NS_RQ_CODE=PROCSTAT      & RRI=RQ  &
                               PROCSTAT_RQ.PROCEDURE_STATUS=IPL_SUCCESSFUL;
 R                           MUCB.DIRECTION=RECEIVE;
 REQCONT                     NS_RQ_CODE=REQCONT       & RRI=RQ;
 'RESET'                     FSMINPUT = 'RESET';
 RPO                         NS_RQ_CODE=RPO           & RRI= RQ;
 +RSP(ABCONN)                NS_RQ_CODE=ABCONN        & RRI=RSP & RTI=POS;
 -RSP(ABCONN)                NS_RQ_CODE=ABCONN        & RRI=RSP & RTI=NEG;
 +RSP(ABCONNOUT)             NS_RQ_CODE=ABCONNOUT     & RRI=RSP & RTI=POS;
 -RSP(ABCONNOUT)             NS_RQ_CODE=ABCONNOUT     & RRI=RSP & RTI=NEG;
 +RSP(ACTCONNIN)             NS_RQ_CODE=ACTCONNIN     & RRI=RSP & RTI=POS;
 -RSP(ACTCONNIN)             NS_RQ_CODE=ACTCONNIN     & RRI=RSP & RTI=NEG;
 +RSP(ACTLINK)               NS_RQ_CODE=ACTLINK       & RRI=RSP & RTI=POS;
 -RSP(ACTLINK)               NS_RQ_CODE=ACTLINK       & RRI=RSP & RTI=NEG;
 +RSP(ACTLU)                 RQ_CODE=ACTLU            & RRI=RSP & RTI=POS;
 -RSP(ACTLU)                 RQ_CODE=ACTLU            & RRI=RSP & RTI=NEG;
 +RSP(ACTPU)                 RQ_CODE=ACTPU            & RRI=RSP & RTI=POS;
 -RSP(ACTPU)                 RQ_CODE=ACTPU            & RRI=RSP & RTI=NEG;
 +RSP(CONNOUT)               NS_RQ_CODE=CONNOUT       & RRI=RSP & RTI=POS;
 -RSP(CONNOUT)               NS_RQ_CODE=CONNOUT       & RRI=RSP & RTI=NEG;
 +RSP(CONTACT)               NS_RQ_CODE=CONTACT       & RRI=RSP & RTI=POS;
 -RSP(CONTACT)               NS_RQ_CODE=CONTACT       & RRI=RSP & RTI=NEG;
 +RSP(DACTCONNIN)            NS_RQ_CODE=DACTCONNIN    & RRI=RSP & RTI=POS;
 -RSP(DACTCONNIN)            NS_RQ_CODE=DACTCONNIN    & RRI=RSP & RTI=NEG;
 +RSP(DACTLINK)              NS_RQ_CODE=DACTLINK      & RRI=RSP & RTI=POS;
 -RSP(DACTLINK)              NS_RQ_CODE=DACTLINK      & RRI=RSP & RTI=NEG;
 ±RSP(DACTLU)                NS_RQ_CODE=DACTLU        & RRI=RSP;
 ±RSP(DACTPU)                NS_RQ_CODE=DACTPU        & RRI=RSP;
 +RSP(DISCONTACT)            NS_RQ_CODE=DISCONTACT    & RRI=RSP & RTI=POS;
 -RSP(DISCONTACT)            NS_RQ_CODE=DISCONTACT    & RRI=RSP & RTI=NEG;
 +RSP(DUMPFINAL)             NS_RQ_CODE=DUMPFINAL     & RRI=RSP & RTI=POS;
 -RSP(DUMPFINAL)             NS_RQ_CODE=DUMPFINAL     & RRI=RSP & RTI=NEG;
 +RSP(DUMPINIT)              NS_RQ_CODE=DUMPINIT      & RRI=RSP & RTI=POS;
 -RSP(DUMPINIT)              NS_RQ_CODE=DUMPINIT      & RRI=RSP & RTI=NEG;
 ±RSP(DUMPTEXT)              NS_RQ_CODE=DUMPTEXT      & RRI=RSP;
 +RSP(INITPROC)              NS_RQ_CODE=INITPROC      & RRI=RSP & RTI=POS;
 -RSP(INITPROC)              NS_RQ_CODE=INITPROC      & RRI=RSP & RTI=NEG;
 +RSP(IPLFINAL)              NS_RQ_CODE=IPLFINAL      & RRI=RSP & RTI=POS;
 -RSP(IPLFINAL)              NS_RQ_CODE=IPLFINAL      & RRI=RSP & RTI=NEG;
 +RSP(IPLINIT)               NS_RQ_CODE=IPLINIT       & RRI=RSP & RTI=POS;
 -RSP(IPLINIT)               NS_RQ_CODE=IPLINIT       & RRI=RSP & RTI=NEG;
 ±RSP(IPLTEXT)               NS_RQ_CODE=IPLTEXT       & RRI=RSP;
 +RSP(NS_IPL_FINAL)          NS_RQ_CODE=NS_IPL_FINAL  & RRI=RSP & RTI=POS;
 ±RSP(PROCSTAT)              NS_RQ_CODE=PROCSTAT      & RRI=RSP;
 +RSP(REQCONT)               NS_RQ_CODE=REQCONT       & RRI=RSP;
 ±RSP(RPO)                   NS_RQ_CODE=RPO           & RRI=RSP;
 S                           MUCB.DIRECTION=SEND;
 -RSP(0822)                  RRI=RSP & RTI=NEG & SNC=0822;
 +RSP(IPL_REQUIRED)          RRI=RSP & RTI=POS & ACTPU_RSP.TYPE_ACTIVATION=X'3';

END FSM_INPUT_DEFINITION;
```

INTRODUCTION

An SNA node contains an SSCP (in PU_T5 nodes), a PU, and (optionally) one or more LUs. These are collectively called NAUs. Every NAU, in turn, contains a NAU services layer, designated SSCP.SVC, PU.SVC, and LU.SVC, respectively. Distributed among the NAU services layers within a network are service and control components. These components control the network operation by exchanging RUs with one another. Additional information about the NAU services layer is contained in Chapters 1 and 6. (PU_T1, PU_T2, and PU_T4 nodes contain a PUCP instead of an SSCP. The PUCP is a subset of the SSCP and is known only within its node. See Chapters 1 and 7 for more details.)

Distributed among each SSCP.SVC and LU.SVC are session services, which coordinate initiation and termination of LU-LU sessions. (There are no session services PU.SVC.) This coordination is accomplished by exchanging session services RUs on SSCP-LU sessions, and on SSCP-SSCP sessions when the two LUs are in different domains. The activation and deactivation of SSCP-LU and SSCP-SSCP sessions, which is not a function of session services, is described elsewhere in this book; see, for example, the descriptions of ACTLU and ACTCDRM in Chapter 13.

The process of initiating or terminating an LU-LU session begins with a session initiation or termination request from an LU to an SSCP, and culminates in the activation or deactivation of the session. Activation and deactivation of an LU-LU session is accomplished by exchanging activation and deactivation requests and responses. These requests and responses—BIND, RSP(BIND), UNBIND, and RSP(UNBIND)—belong to the category of session control (SC) RUs. Information pertaining to these RUs is contained in Chapter 13. Session services includes that part of the initiation or termination process up to, but not including, the activation or deactivation of the session.

As shown in Figures 8-1 and 8-2, the session services for each SSCP.SVC and LU.SVC consists of a services manager component, and one or more half-session components (one per half-session). The services manager component for the SSCP.SVC is designated SSCP.SVC_MGR.SS; for the LU.SVC, it is designated LU.SVC_MGR.SS. The half-session components for both the SSCP.SVC and LU.SVC are designated SNS.SS. The SSCP.SVC_MGR.SS, LU.SVC_MGR.SS, and SNS.SS components are each made up of two main subcomponents: a send subcomponent and a receive subcomponent.

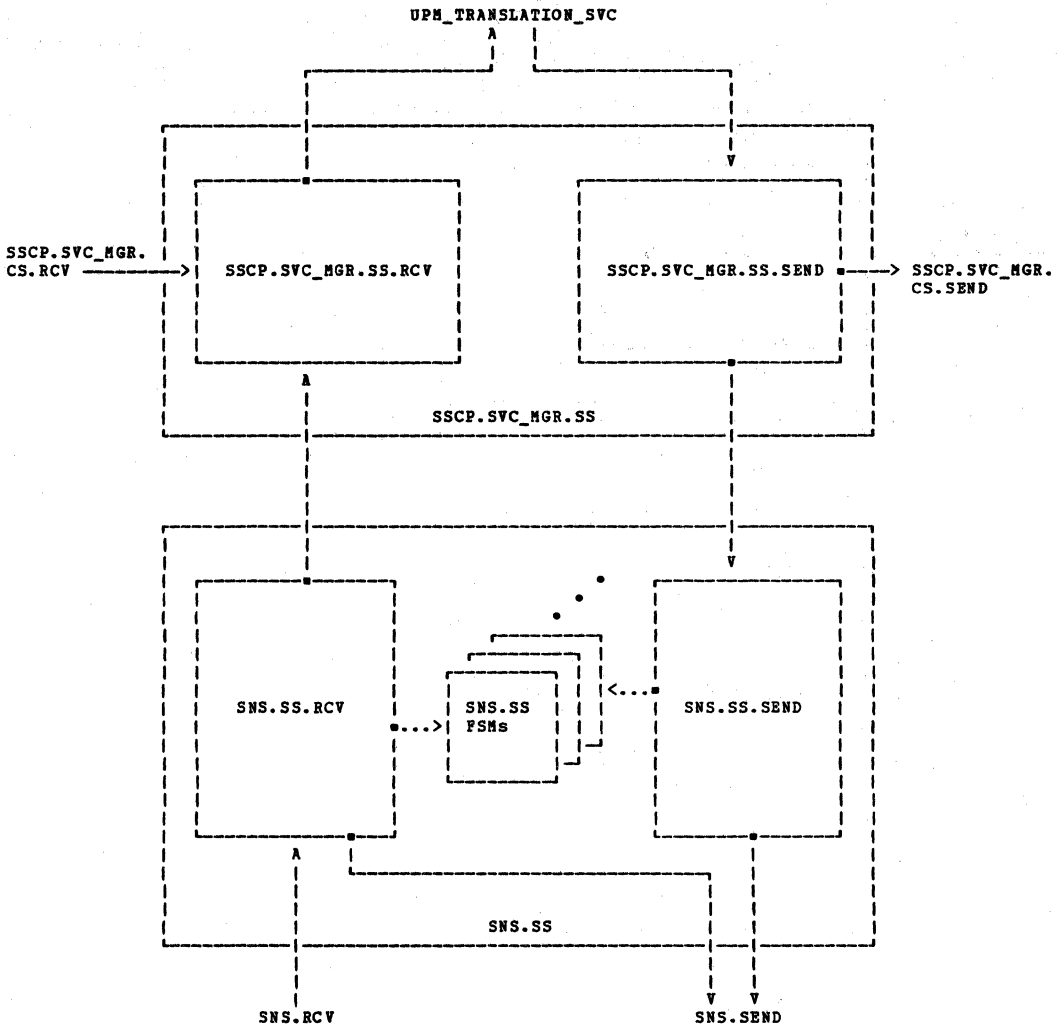


Figure 8-1. Structure of SSCP Session Services

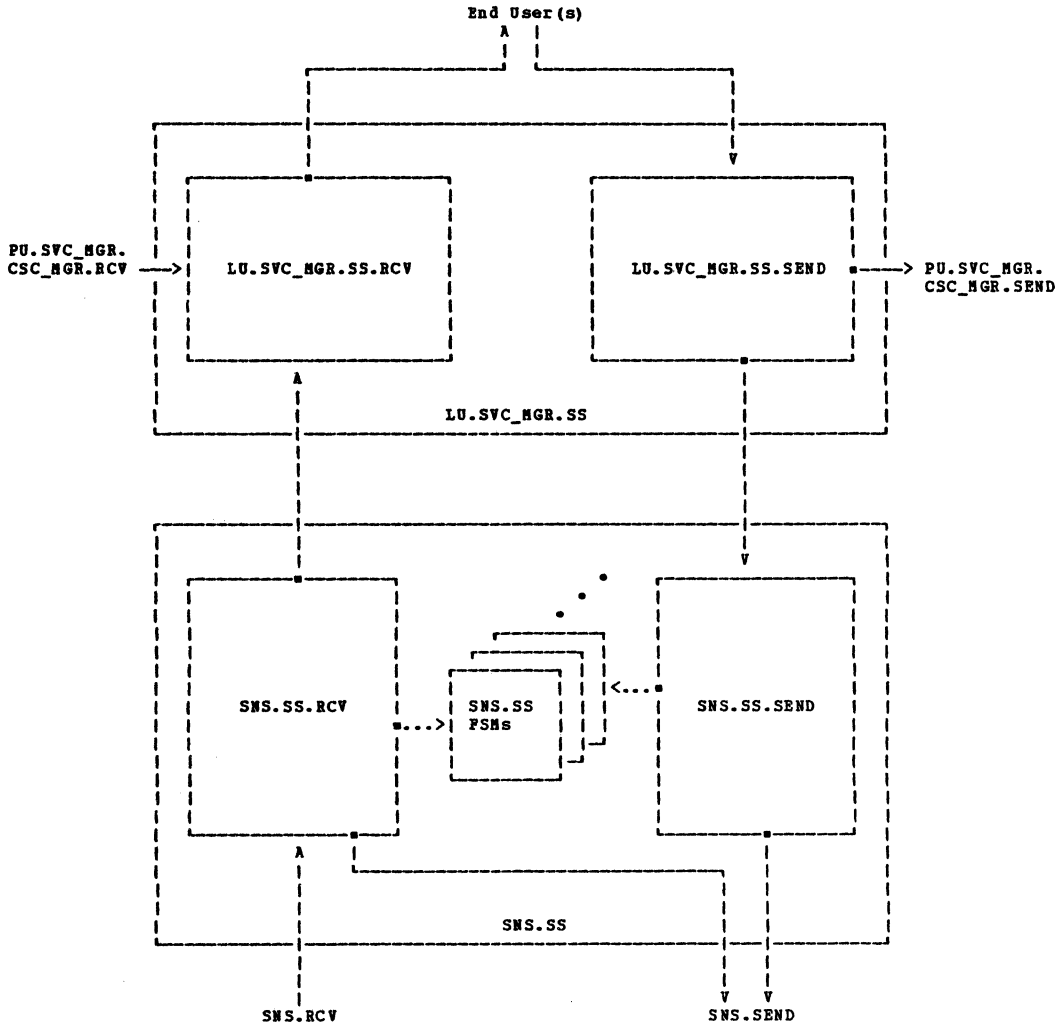


Figure 8-2. Structure of LU Session Services

The SSCP.SVC_MGR.SS component includes a variety of services related to assisting LUs in initiating and terminating LU-LU sessions. These services include:

- Verifying the authority of the LU requesting initiation or termination of a session
- Translating network names to network addresses
- Queuing and dequeuing requests for session initiation
- Selecting appropriate session parameters
- Synchronizing the initiation or termination process

The LU.SVC_MGR.SS component includes services, complementary to those of the SSCP.SVC_MGR.SS, to assist the LU in initiating or terminating a session with another LU. Further details of the SSCP.SVC_MGR.SS and LU.SVC_MGR.SS are given later in this chapter within the descriptions of the session services RUs.

NETWORK CONTEXT FOR SESSION SERVICES

Figures 8-3 and 8-4 show examples of the network contexts of session services, and illustrate the concepts:

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

The half-sessions involve at least one SSCP and have the following identifications:

- (SSCP,K).PRI|SEC, where K = LU|ILU|TLU|PLU|SLU
- (SSCP(X),SSCP(Y)).SSCP(X|Y), where X,Y = ILU|TLU|OLU|DLU|PLU|SLU, and SSCP(X|Y) is the SSCP that has an active session with X or Y
- (SSCP,SSCP').SSCP, where SSCP and SSCP' are any two SSCPs having an active session

The concepts ILU, TLU, PLU, SLU, OLU, and DLU refer to the role of an LU with respect to a given LU-LU session.

PLU AND SLU

PLU and SLU refer, respectively, to the role of the LU in providing the primary or secondary half-session support for an active session of which it is a partner.

ILU AND TLU

ILU and TLU refer to the role of an LU in initiating or terminating a given LU-LU session; the ILU sends an INIT-SELF or INIT-OTHER request for the session, and the TLU sends a TERM-SELF or TERM-OTHER request for the session. For INIT-SELF and TERM-SELF, the LU sending the request participates as a session partner; for INIT-OTHER and TERM-OTHER, the LU may or may not participate as a partner.

OLU AND DLU

The OLU and DLU concepts are more involved, and are defined in terms of the role of the LU or its SSCP with respect to initiation and termination of LU-LU sessions as follows:

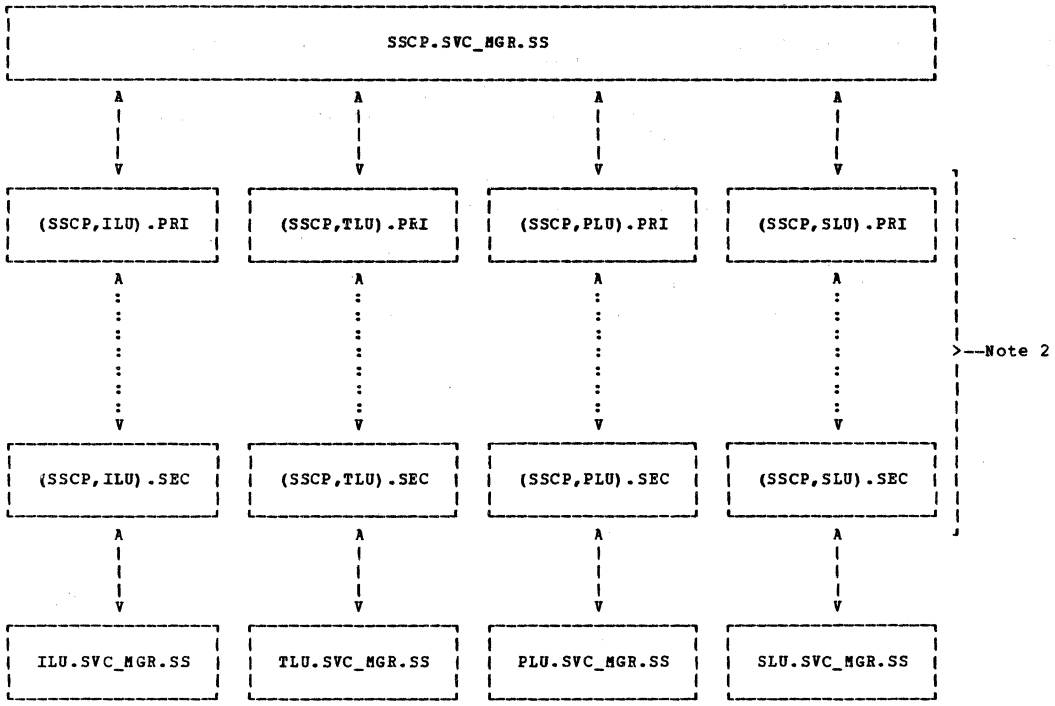
- For a same-domain LU-LU session, the OLU:
  - -- For initiation, is the LU that sends INIT-SELF (i.e., the ILU), or that is named second in INIT-OTHER (i.e., LU2)
  - -- For termination, is the LU that sends TERM-SELF (i.e., the TLU), or that is named second in TERM-OTHER (i.e., LU2)

  while the DLU:
  - -- For initiation, is the LU that is named in INIT-SELF, or named first in INIT-OTHER (i.e., LU1)
  - -- For termination, is the LU that is named in TERM-SELF, or named first in TERM-OTHER (i.e., LU1)

- For a cross-domain LU-LU session, the OLU:
  - -- For initiation, is the LU whose SSCP (referred to as the SSCP(OLU)) sends CDINIT
  - -- For termination, is the LU whose SSCP (again referred to as the SSCP(OLU)) sends CDTERM

  while the DLU:
  - -- For initiation, is the LU whose SSCP (the SSCP(DLU)) receives CDINIT
  - -- For termination, is the LU whose SSCP (the SSCP(DLU)) receives CDTERM

In the same-domain context, notice that SSCP(OLU) = SSCP(DLU). Figure 8-5 illustrates the concepts in the cross-domain context.

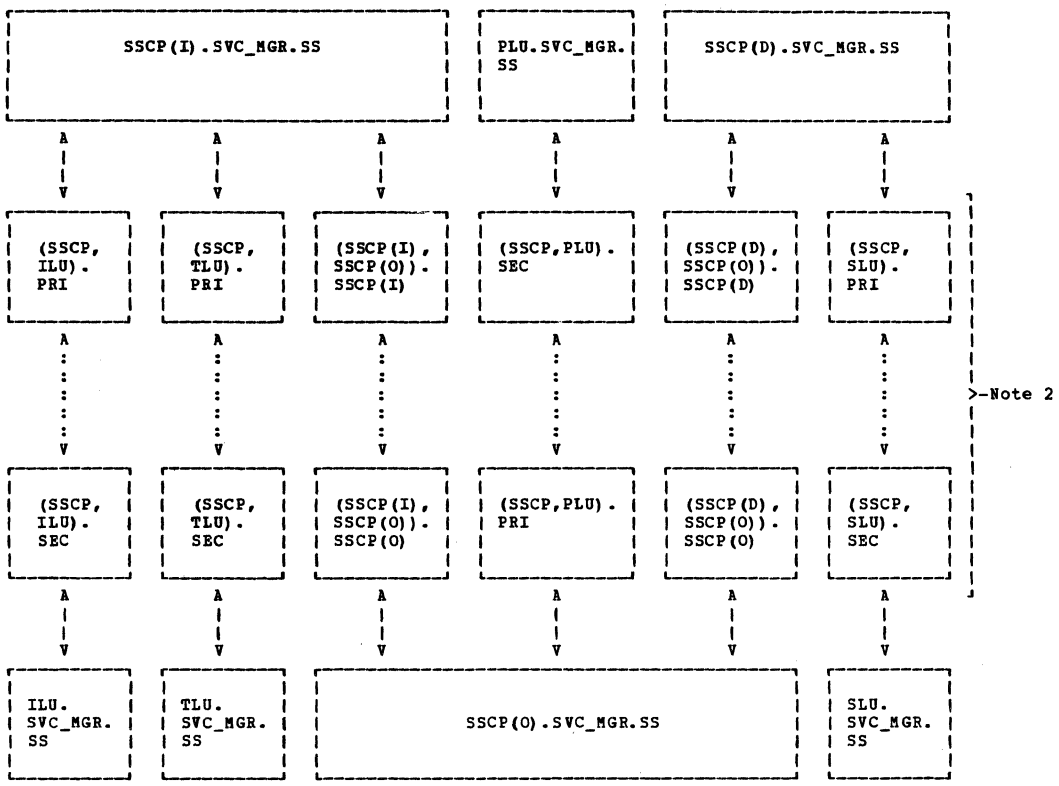


Notes:

1. Let:
   - ILU denote the LU requesting an LU-LU session initiation via the INIT request.
   - TLU denote the LU requesting an LU-LU session termination via the TERM request.
   - PLU denote the primary LU in the referenced session.
   - SLU denote the secondary LU in the referenced session.

2. Half-sessions, connected by NTWK.PC

Figure 8-3. System Context for Session Services--Single Domain

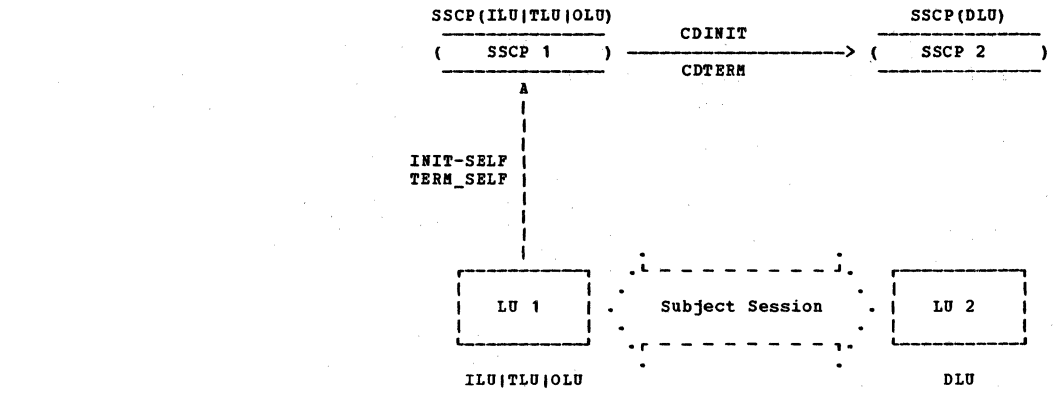


Notes:

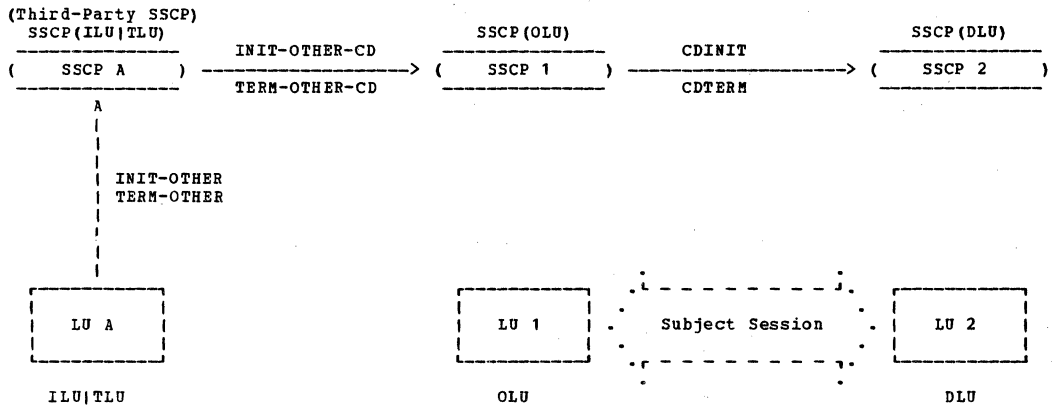
1. I = ILU; O = OLU; D = DLU.

2. Half-sessions, connected by NTWK.PC

Figure 8-4. System Context for Session Services--Multiple Domain



A. "SELF" Example



B. "OTHER" Example

Figure 8-5. OLU and DLU in the Cross-Domain Context

SESSION NETWORK SERVICES FOR SESSION SERVICES

A session network services (SNS) component for session services (SNS.SS) exists for each half-session, and consists of two main subcomponents, SNS.SS.RCV and SNS.SS.SEND. These subcomponents are coupled by a set of SNS.SS FSMs, as shown in Figures 8-1 and 8-2. The SNS.SS FSMs are initialized to the reset state when the half-session is activated.

The SNS.SS.RCV subcomponent receives all session services requests and responses from SNS.RCV. The receipt of session services RUs is handled by the following protocol machines within SNS.SS.RCV:

- In the SSCP:

  -- (SSCP,SSCP').SSCP.SNS.SS.RQ_RCV
  -- (SSCP,SSCP').SSCP.SNS.SS.RSP_RCV
  -- (SSCP,LU).PRI.SNS.SS.RQ_RCV
  -- (SSCP,LU).PRI.SNS.SS.RSP_RCV

- In the LU:

  -- (SSCP,LU).SEC.SNS.SS.RQ_RCV
  -- (SSCP,LU).SEC.SNS.SS.RSP_RCV

The SNS.SS.SEND subcomponent sends all session services requests and responses to SNS.SEND. The sending of session services RUs is handled by the following protocol machines within SNS.SS.SEND:

- In the SSCP:

  -- (SSCP,SSCP').SSCP.SNS.SS.RQ_SEND
  -- (SSCP,SSCP').SSCP.SNS.SS.RSP_SEND
  -- (SSCP,LU).PRI.SNS.SS.RQ_SEND
  -- (SSCP,LU).PRI.SNS.SS.RSP_SEND

- In the LU:

  -- (SSCP,LU).SEC.SNS.SS.RQ_SEND
  -- (SSCP,LU).SEC.SNS.SS.RSP_SEND

These protocol machines perform usage and state checks and interact with the SNS.SS FSMs to control the sending and receiving of session services RUs, as described in the following sections.

(SSCP,LU).SEC.SNS.SS.RQ_SEND

(SSCP,LU).SEC.SNS.SS.RQ_SEND helps to issue and regulate session services requests sent to the SSCP by, or on behalf of, an LU. In addition to handling the reply requests issued by SNS.SS FSMs associated with (SSCP,LU).SEC.SNS.SS.RQ_RCV (i.e., SESSST, SESSEND, BINDF, and UNBINDF, discussed later in this chapter), this protocol machine processes INIT-SELF, INIT-OTHER, TERM-SELF, TERM-OTHER, and NOTIFY(Vector Key X'0C'), which are initiated by an LU.

(SSCP,LU).SEC.SNS.SS.RQ_SEND applies the appropriate usage and state send checks to each such request, and, if valid, routes it to the appropriate SNS.SS FSM. Details of the checking and routing are not defined.

(SSCP,LU).SEC.SNS.SS.RQ_RCV

(SSCP,LU).SEC.SNS.SS.RQ_RCV assists in checking for the proper receipt of requests sent by the SSCP to the LU. It handles CINIT and CTERM for PLUs, CLEANUP for SLUs, and NOTIFY(Vector Keys X'01', X'03', and X'04') and NSPE for all LUs. It uses the state receive checks of Figure 8-6 and the destination table of Figure 8-7.

(SSCP,LU).SEC.SNS.SS.RSP_SEND AND (SSCP,LU).SEC.SNS.SS.RSP_RCV

(SSCP,LU).SEC.SNS.SS.RSP_SEND and (SSCP,LU).SEC.SNS.SS.RSP_RCV assist in the proper sending and receiving of responses by SNS.SS. They apply the appropriate usage and state checks and route the responses to the appropriate SNS.SS FSMs. Details are not defined.

Request	FSM = State Condition (if OK)	Sense Code (if NG)
CINIT(PLU,SLU)	CSESS_RCV = RESET	0809
CLEANUP	CLEANUP_RCV = RESET	0809
CTERM(PLU,SLU) (test in order a,b)	a) CSESS_RCV = RESET | CTERM_RCV = RESET	0809
	b) CSESS_RCV = ¬RESET	0816
NOTIFY	none (no error)	-
NSPE	none (no error)	-

Note: If the state condition is true for a given request, the request is OK. Otherwise, the request is "no good" (NG), and a negative response with the specified sense code is generated.

Figure 8-6. State Receive Checks for (SSCP,LU).SEC.SNS.SS.RQ_RCV

Request	Destination FSM	Page
CINIT(PLU,SLU)	FSM_SSCP_PLU_SEC_CSESS_RCV	8-38
CLEANUP	(SSCP,SLU).SEC.CLEANUP_RCV	8-42
CTERM(PLU,SLU)	FSM_SSCP_PLU_SEC_CTERM_RCV	8-40
NOTIFY	((SSCP,SSCP').SSCP')|((SSCP,LU).SEC)| ((SSCP,LU).PRI).NOTIFY_RCV	8-47
NSPE	none	

Figure 8-7. Destination Table for (SSCP,LU).SEC.SNS.SS.RQ_RCV

(SSCP,LU).PRI.SNS.SS AND (SSCP,SSCP').SSCP.SNS.SS

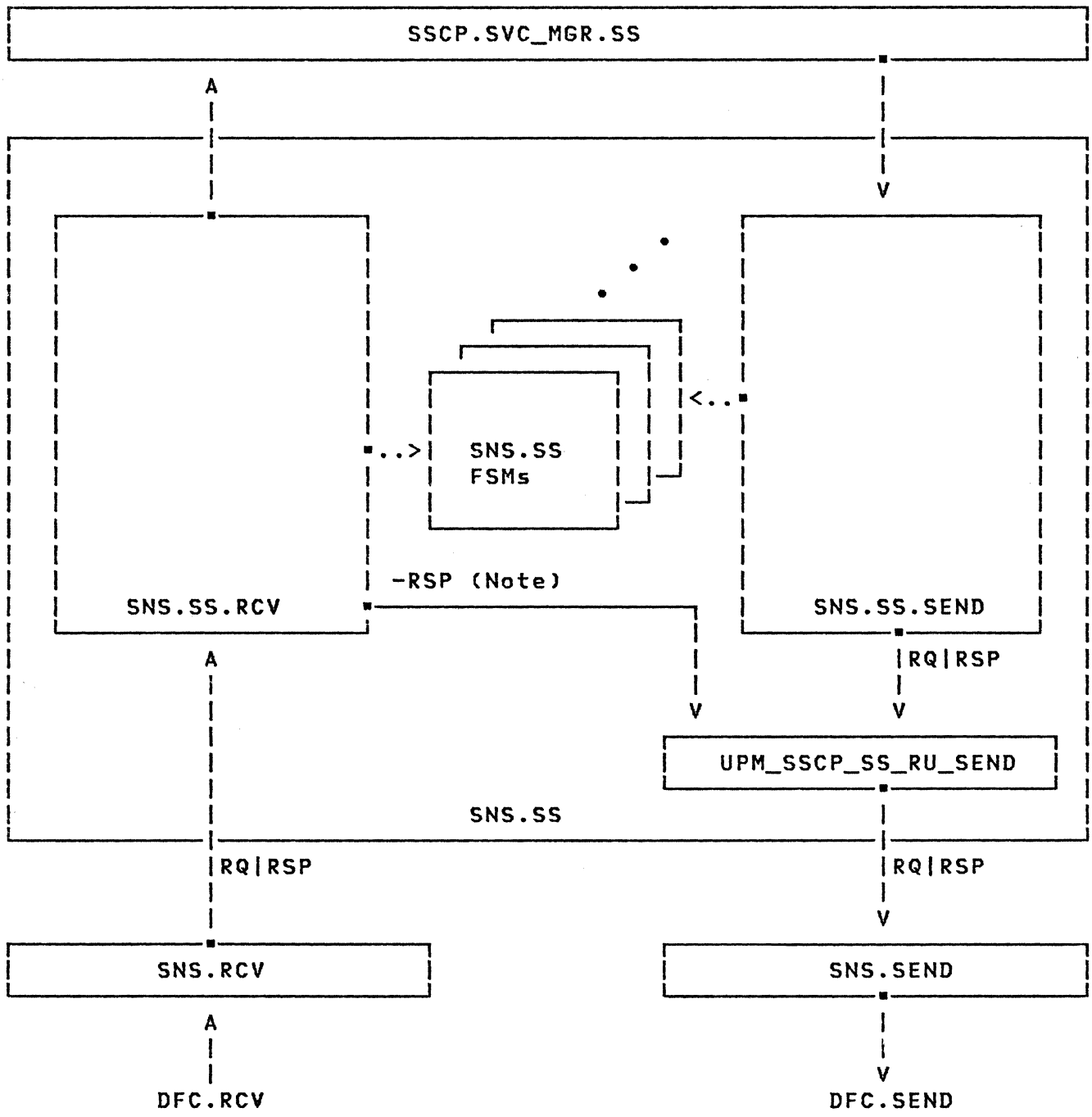
The SNS.SS for an SSCP-based half-session (i.e., (SSCP,LU).PRI.SNS.SS or (SSCP,SSCP').SSCP.SNS.SS) is shown in Figure 8-8. The session services protocol machines handle field-formatted RUs only; character-coded RUs received from an LU are translated to the field-formatted form by SNS.RCV (see Chapter 6), and character-coded RUs sent to an LU are translated from the field-formatted form by UPM_SSCP_SS_RU_SEND.

The SSCP-based half-session component, SNS.SS.RQ_RCV, assists in checking for proper receipt of all requests sent by an LU or another SSCP. It uses the state receive checks of Figures 8-9 and 8-10, and the destination table of Figure 8-11.

The SSCP-based half-session component, SNS.SS.RSP_RCV, assists in the proper receiving of responses by SNS.SS. It applies the appropriate usage and state checks and routes the responses to the appropriate SNS.SS FSMs. Details are not defined.

The SSCP-based half-session components, SNS.SS.RQ_SEND and SNS.SS.RSP_SEND, help to issue and regulate session services requests and responses sent by the SSCP. They apply the appropriate usage and state send checks to each outgoing request and response, and, if valid, route it to the appropriate SNS.SS FSM. Other details of the checking and routing are not defined.

UPM_SSCP_SS_RU_SEND monitors all requests and responses leaving (SSCP,LU).PRI.SNS.SS. It translates all field-formatted requests and responses to character-coded, as appropriate. Additionally, after a negative response is sent, it may create a character-coded request that carries additional error information.



Note: SNS.SS.RCV returns negative responses to all requests found to be "no good" (NG) as a result of the usage and state receive checks.

Figure 8-8. (SSCP,SSCP').SSCP.SNS.SS and (SSCP,LU).PRI.SNS.SS

Request	FSM = State Condition (if OK)	Sense Code (if NG)
INIT-SELF & INIT-OTHER	Unique FSM per Request	—
SESSST & BINDF	CSESS_SEND = PEND_ACTIVE_SESSST	if RESET: 0816 else: 0809
TERM-SELF & TERM-OTHER	Unique FSM per Request	—
SESSEND & UNBINDF	CSESS_SEND = ACTIVE | PEND_ACTIVE_SESSST	if RESET: 0816 else: 0809
NOTIFY	Unique FSM per Request	—

Note: If the state condition is true for a given request, the request is OK. Otherwise, the request is "no good" (NG), and a negative response with the specified sense code is generated.

Figure 8-9. State Receive Checks for (SSCP,LU).PRI.SNS.SS.RQ_RCV

Request	FSM = State Condition (if OK)	Sense Code (if NG)
CDCINIT	SSCP(PLU).CDCSESS = RESET	0815
CDINIT(DQ)	CDINIT_RCV | CDINIT_SEND = PEND_CDINIT_DQ | PEND_DQ_SEND	0809
CDINIT(¬DQ)	Unique FSM per Request	—
CDSESSEND	SSCP(PLU).CDCSESS = ACTIVE | PEND_TAKED_ACT | PEND_TAKED_PEND | PEND_RSP_CDSESSEND_ACT | PEND_RSP_CDSESSEND_TAKED	if RESET: 0816 else: 0809
	SSCP(SLU).CDCSESS = ACTIVE | PEND_TAKED_ACT | PEND_RSP_CDSESSEND_ACT | PEND_RSP_CDSESSEND_TAKED	if RESET: 0816 else: 0809
CDSESSST & CDSESSSF	SSCP(SLU).CDCSESS = PEND_SETUP | PEND_TAKED_PEND	if RESET: 0816 else: 0809
CDSESSTF	SSCP(SLU).CDCSESS = ACTIVE | PEND_TAKED_ACT	if RESET: 0816 else: 0809
CDTAKED	CDTAKED(Type) = RESET | PEND_ACT_SEND	0815
	CDTAKED(CU) = RESET | ACT_SEND	0815
CDTAKEDC	CDTAKED(Type) = PEND_ACT_RQ_RCV | ACTIVE_SEND	if RESET: 0816 else: 0809
CDTERM	CDTERM = RESET	0809
DSRLST	DSRLST = RESET	0815
INIT-OTHER-CD	Unique FSM per Request	—
NOTIFY	Unique FSM per Request	—
TERM-OTHER-CD	Unique FSM per Request	—

Note: If the state condition is true for a given request, the request is OK. Otherwise, the request is "no good" (NG), and a negative response with the specified sense code is generated.

Figure 8-10. State Receive Checks for (SSCP,SSCP').SSCP.SNS.SS.RQ_RCV

Request	Destination FSM	Page
BINDF	FSM_SSCP_PLU_PRI_CSESS_SEND	8-37
CDCINIT	(SSCP(PLU),SSCP(SLU)).SSCP(PLU).CDCSESS(PLU,SLU,PCID)	8-59
CDINIT(DQ)	(SSCP(DLU),SSCP(OLU)).SSCP(OLU).CDINIT(OLU,DLU,PCID)_SEND (SSCP(DLU),SSCP(OLU)).SSCP(DLU).CDINIT(OLU,DLU,PCID)_RCV	8-51 8-52
CDINIT(¬DQ)	(SSCP(DLU),SSCP(OLU)).SSCP(DLU).CDINIT(OLU,DLU,PCID)_RCV	8-52
CDSESSEND	(SSCP(PLU),SSCP(SLU)).SSCP(SLU).CDCSESS(PLU,SLU,PCID)	8-58
CDSESSSF	(SSCP(PLU),SSCP(SLU)).SSCP(SLU).CDCSESS(PLU,SLU,PCID)	8-58
CDSESSST	(SSCP(PLU),SSCP(SLU)).SSCP(SLU).CDCSESS(PLU,SLU,PCID)	8-58
CDSESSTF	(SSCP(PLU),SSCP(SLU)).SSCP(SLU).CDCSESS(PLU,SLU,PCID)	8-58
CDTAKED	(SSCP,SSCP').SSCP.CDTAKED(Type,PCID)_SEND-RCV (SSCP,SSCP').SSCP.CDTAKED(CU)_SEND-RCV	8-68 8-69
CDTAKEDC	(SSCP,SSCP').SSCP.CDTAKED(Type,PCID)_SEND-RCV	8-68
CDTERM	(SSCP(DLU),SSCP(OLU)).SSCP(OLU|DLU).CDTERM(SESSION_KEY_CONTENT, PCID)_SEND-RCV	8-62
DSRLST	((SSCP,SSCP').SSCP).DSRLST_RCV	8-71
INIT	(SSCP,ILU).PRI.INIT(OLU,DLU)|(LU1,LU2)_RCV	8-27
INIT-OTHER-CD	(SSCP(OLU),SSCP(ILU)).SSCP(OLU).INIT-OTHER-CD(OLU,DLU,PCID)_RCV	8-54
NOTIFY	((SSCP,SSCP').SSCP')|((SSCP,LU).SEC)|((SSCP,LU).PRI).NOTIFY_RCV	8-47
SESSEND	FSM_SSCP_PLU_PRI_CSESS_SEND	8-37
SESSST	FSM_SSCP_PLU_PRI_CSESS_SEND	8-37
TERM	(SSCP,TLU).PRI.TERM.(SESSION_KEY_CONTENT|URC)_RCV	8-33
TERM-OTHER-CD	(SSCP(OLU),SSCP(TLU)).SSCP(OLU).TERM-OTHER-CD(SESSION_KEY_ CONTENT,PCID)_RCV	8-64
UNBINDF	FSM_SSCP_PLU_PRI_CSESS_SEND	8-37

Figure 8-11. Destination Table for (SSCP,LU).PRI.SNS.SS.RQ_RCV and (SSCP,SSCP').SSCP.SNS.SS.RQ_RCV

SESSION SERVICES 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 FMD. See Chapter 6 for additional details about NS request and response formats, including the NS header and related RH parameters.

Session services requests flowing from an LU to an SSCP, or from an SSCP to an LU, may be field-formatted (RH Format indicator set to NSH) or character-coded (RH Format indicator set to ¬NSH). Character-coded requests contain RUs consisting of character strings that can be translated into equivalent field-formatted RUs. A translation protocol is provided by (SSCP,LU).PRI.SNS.SS (described elsewhere in this chapter); the translation rules are implementation- and installation-dependent, and are not defined in this book. Session services requests flowing from an SSCP to another SSCP are always field-formatted.

Full details of the RU formats for field-formatted session services requests (and responses) are given in Appendix E. In the following paragraphs, some fields that are common to many session services RUs are defined.

CLASS OF SERVICE

End users can request from the network, as part of an INIT request, a class of service for a 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.

COS NAME

A user specifies a class of service for a session by means of a class of service (COS) name. The COS name resolves to an ordered list of (virtual route number, transmission priority field), or (VRN,TPF), pairs, each identifying a virtual route (VR). The list, called a VR identifier list, allows the session to be assigned to the first available virtual route identified in the list. (A session is assigned to a virtual route at session activation time.)

For LU-LU sessions, the COS name is specified explicitly as a parameter of an INIT request, or it is derived from the mode name (also carried in, or implied by, the INIT). The derivation of this default COS name is performed by the SSCP(SLU). The COS name is resolved to a VR identifier list by the SSCP(PLU). See Chapter 12 for additional information on the use of the VR identifier list for activating a VR.

NETWORK NAME

A network name is the name by which a PU, an LU, or a link is known throughout a multiple-domain network. Network names used across various domains must be unique within the multiple-domain network.

UNINTERPRETED NAME

An uninterpreted name is any name by which one LU is known to another LU and its SSCP for the purpose of initiating or terminating an LU-LU session. It can be used by an ILU or TLU to identify an OLU and DLU, or by an OLU to identify a DLU. An uninterpreted name requires interpretation (or transformation) by the SSCP(ILU|TLU|OLU) in order to yield the network name; interpretation of an uninterpreted name that is the same as a network name is an identity transformation. An SSCP may support only identity transformations.

PROCEDURE CORRELATION IDENTIFICATION

A procedure correlation identification (PCID) is generated by an SSCP originating a cross-domain procedure. The first cross-domain request issued for a procedure causes generation of a unique PCID, which is then retained and used in all cross-domain requests dealing with the same procedure until it is completed. An SSCP maintains correlation between PCID and a user request correlation (URC) value (discussed in the section, "User Request Correlation"), when the latter has been provided in an INIT-SELF, INIT-OTHER, TERM-SELF, or TERM-OTHER request.

For LU-LU session initiation, a PCID is generated by the SSCP(ILU), and identifies the initiation procedure for that session. Similarly, for LU-LU session termination, a PCID is generated by the SSCP(TLU), and identifies the termination procedure for that session. For cross-domain takedown (initiated via CDTAKED), a PCID is generated by the SSCP sending the CDTAKED, and identifies the takedown procedure that is used for the duration of the takedown processing (until CDTAKEDC requests are exchanged).

A PCID has a fixed length and consists of two fields: a two-byte field containing the network address of the originating SSCP (possibly a third-party SSCP), and a 6-byte field that provides a unique value identifying the originating SSCP's procedure. When bytes 0-1 contain the value 0, bytes 2-7 are reserved.

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 or TERM request. Its usage allows subsequent requests within a given procedure involving the SSCP(ILU|TLU) and the ILU|TLU to be associated with the request that originally initiated the procedure. Associated requests either contain a field specifically defined for this purpose or use a session key (discussed in the section, "Session Key and Session Key Content"). When a URC is assigned, the SSCP is responsible for maintaining correlation between this URC and the SSCP-generated PCID for the same procedure. A value of 0 in the Length field indicates no URC is present.

MODE TABLE AND MODE NAME

The SSCP(SLU) has information about the SLU that aids in the construction of the BIND image (carried, for example, in CINIT). This information is contained in a mode table. The mode table is indexed by the mode name supplied in INIT requests and carried, in the case where ILU=PLU, to the SSCP(SLU) in CDINIT. The format of the mode table and of the data contained in each entry is implementation- and installation-dependent. The data associated with each mode name consists of:

- Bytes 1 through 27 of the BIND
- The (optional) User Data field of the BIND image to be carried in CINIT (and CDCINIT, if SSCP(PLU) ¬= SSCP(SLU))
- The Device Characteristics field in CINIT (and CDCINIT, if SSCP(PLU) ¬= SSCP(SLU))

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 PCID. 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 field(s) denoted by the session key. The format description of a request specifying a session key and session key content also specifies the list of keys permitted (or required) with that request.

When session key content contains a pair, e.g., name pair, address pair, or address-name 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 other LU designations, i.e., (OLU,DLU) and (LU1,LU2). For these formats the session key content order is (OLU,DLU) or (LU1,LU2) and other related fields specify which is PLU and which is SLU. The following table shows, by key value, the session key content and the requests that can carry the session key and its content:

Session Key	Session Key Content and Applicable Request(s)
X'01'	Uninterpreted name: carried in TERM-SELF
X'05'	PCID: carried in CDTERM, NOTIFY, and TERM-OTHER-CD
X'06'	Uninterpreted name pair: carried in BINDF, CLEANUP, NSPE, SESSEND(Format 0), SESSST, TERM-OTHER, and UNBINDF; or network name pair: carried in CDSESSEND, CDSESSSF, CDSESSST, CDSESSTF, CDTERM, NOTIFY, SESSEND(Format 2), and TERM-OTHER-CD
X'07'	Network address pair: carried in BINDF, CDSESSEND, CDSESSSF, CDSESSST, CDSESSTF, CDTERM, CINIT, CLEANUP, CTERM, NOTIFY, SESSEND, SESSST, TERM-OTHER, TERM-OTHER-CD, TERM-SELF, and UNBINDF
X'08'	Network address of PLU, network name of SLU: carried in CDTERM
X'0A'	URC: carried in NOTIFY, TERM-OTHER, and TERM-SELF

SESSION SERVICES REQUESTS

Listed below are the session services requests, grouped according to their use, and the page on which the description of the request begins. Each description of a request includes the RU flow; a list of the applicable FSMs; a discussion of the function, use, and protocols of the request; and a definition of the associated FSMs. Refer to Appendix E for specifications of the RU formats.

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

Request	Name	Page
INIT-SELF	INITIATE-SELF	8-22
INIT-OTHER	INITIATE-OTHER	8-22
CINIT	CONTROL INITIATE	8-34
SESSST	SESSION STARTED	8-34
BINDF	BIND FAILURE	8-34
INIT-OTHER-CD	INITIATE-OTHER CROSS-DOMAIN	8-53
CDINIT	CROSS-DOMAIN INITIATE	8-48
CDCINIT	CROSS-DOMAIN CONTROL INITIATE	8-55
CDSESSST	CROSS-DOMAIN SESSION STARTED	8-55
CDSESSSF	CROSS-DOMAIN SESSION SETUP FAILURE	8-55

Requests relating to session termination are:

Request	Name	Page
TERM-SELF	TERMINATE-SELF	8-28
TERM-OTHER	TERMINATE-OTHER	8-28
CTERM	CONTROL TERMINATE	8-34
CLEANUP	CLEANUP SESSION	8-41
SESSEND	SESSION ENDED	8-34
UNBINDF	UNBIND FAILURE	8-34
TERM-OTHER-CD	TERMINATE-OTHER CROSS-DOMAIN	8-63
CDTERM	CROSS-DOMAIN TERMINATE	8-60
CDSESSEND	CROSS-DOMAIN SESSION ENDED	8-55
CDSESSTF	CROSS-DOMAIN SESSION TAKEDOWN FAILURE	8-55

Requests pertaining to termination of all cross-domain LU-LU sessions involving the domains of both SSCPs are:

Request	Name	Page
CDTAKED	CROSS-DOMAIN TAKEDOWN	8-65
CDTAKEDC	CROSS-DOMAIN TAKEDOWN COMPLETE	8-65

Requests pertaining to reporting the status of the session initiation or termination, or of the LU are:

Request	Name	Page
NOTIFY	NOTIFY	8-44
NSPE	NETWORK SERVICES PROCEDURE ERROR	8-43

The following request pertains to obtaining the status of an LU located in another domain:

Request	Name	Page
DSRLST	DIRECT SEARCH LIST	8-70

INITIATE-SELF (INIT-SELF)
INITIATE-OTHER (INIT-OTHER)

Flow: From ILU to SSCP(ILU) (Normal)

Principal FSMs:
(SSCP,ILU).SEC.INIT((OLU,DLU)|(LU1,LU2))_SEND
(Page 8-27)
(SSCP,ILU).PRI.INIT((OLU,DLU)|(LU1,LU2))_RCV
(Page 8-27)

INIT-SELF from the ILU requests that the SSCP authorize and assist in the initiation of a session between the LU sending the request (i.e., the ILU, which also becomes the OLU) and the LU named in the request (the DLU).

The session to be initiated may be between logical units in the same domain or in different domains. The INIT-SELF request indicates, among other parameters, the uninterpreted name of the other LU in the session to be initiated, together with optional URC and COS name fields.

The SSCP retains sufficient information (e.g., the network address of the ILU) in order later to be able to send NOTIFY(Vector Key X'03') to the ILU to report the status of the initiation, if requested by the NOTIFY specification in INIT-SELF(Format 1 or 2). If an initiation failure occurs, NOTIFY(Vector Key X'03') or NSPE is sent to the ILU, independent of the NOTIFY specification in INIT-SELF (NSPE is sent in lieu of NOTIFY only if INIT-SELF(Format 0) was received).

(SSCP,ILU).PRI.INIT(OLU,DLU)_RCV receives the INIT-SELF request and, if it is valid, passes it to SSCP.SVC_MGR.SS, which may perform the following processing (when the PLU and SLU are in different domains, the processing is distributed between the two SSCPs, each SSCP processing the portion that relates to the LU in its domain):

- Resolve the uninterpreted name of the DLU to a network name (performed by the SSCP(ILU).SVC_MGR.SS).

- Resolve the network name of the DLU to a network address (performed by the SSCP(DLU).SVC_MGR.SS).

- Assign an additional network address to the PLU, if required; the SSCP(PLU) issues RNAA to the PU of the PLU if the PLU supports parallel sessions and an additional network address is required. This network address is carried to the PLU in the CINIT RU. If a network address cannot be assigned by the PU, then a negative response--Insufficient Resource (X'0812')--is returned to the ILU via NOTIFY(Vector Key X'03'). The SSCP(PLU) issues FNA to the PU of the PLU to free the

previously assigned network address when all sessions associated with the network address to be freed have been terminated.

- Determine that a path exists between the DLU and SSCP(DLU); this may require configuration services to establish a connection via a switched link.

- Determine that the necessary SSCP-SSCP session is active.

- Establish the authority of the end user and the ILU to access the DLU. The requester ID and password may be used for this purpose.

- Establish the availability of the DLU for activation of an LU-LU session. An LU may be unavailable because it is not currently able to comply with the PLU|SLU specification, or because it is at its session limit. An LU informs the SSCP of its availability at SSCP-LU session activation time via control vector X'0C' carried in its +RSP(ACTLU). Subsequently, during the active SSCP-LU session, the LU reports changes in its availability (e.g., changes in its PLU|SLU capability or its session limit) by sending NOTIFY(Vector Key X'0C') to the SSCP.

- Retain (at the SSCP(ILU).SVC_MGR.SS) the URC, if supplied in the request, for later inclusion within any NOTIFY RU sent back to the ILU.

- When an INIT-SELF is issued by the PLU (ILU=OLU=PLU) and the COS name is specified, the SSCP verifies that the COS name is a valid entry in the "COS name to VR identifier list" table. If not valid, it sends a -RSP(INIT-SELF, X'08610001'), thereby indicating the invalid COS name.

- When an INIT-SELF is issued by the PLU (ILU=OLU=PLU) and a COS name is not specified (INIT-SELF, not Format 2), the SSCP(ILU) specifies in CDINIT that COS name was not received from the ILU and that the SSCP(DLU=SLU) is to choose the COS name. The SSCP(DLU=SLU) selects the COS name and sends it to the SSCP(OLU=PLU) via the RSP(CDINIT).

- When an INIT-SELF is issued by the SLU (ILU=OLU=SLU) and a COS name is not specified, then the SSCP(OLU=SLU) derives COS name from the mode table, and specifies in CDINIT that COS name was not received from the ILU and that it, the SSCP(OLU), has chosen the COS name.

- When an INIT-SELF is issued by the SLU (ILU=OLU=SLU), the SSCP indicates in CDINIT whether the SLU or BF supports sending UNBIND and SESSEND. (The indication is in the OLU status byte of CDINIT, byte 6, bit 5.)

- When an INIT-SELF is issued by the PLU (ILU=PLU) and a URC is supplied, the SSCP(PLU) places the URC in the BIND image of CINIT to allow the PLU to correlate the CINIT with the INIT-SELF. When an INIT-SELF is issued by the SLU (ILU=SLU) and a URC is supplied, the URC is carried by CDCINIT if SSCP(SLU)¬=SSCP(PLU), by CINIT, and by BIND to allow the SLU to correlate the BIND with the INIT-SELF.

- Save the User Data field (at the SSCP(ILU)) for inclusion in the User Data field in CINIT or to pass to the SSCP(PLU) via CDINIT.

- Determine which LU is to be the primary LU (PLU) for the session, as specified in the INIT-SELF request.

- Select (at the SSCP(SLU)) BIND parameters based on the mode name in the request.

- Generate a PCID to identify the initiation procedure used in initiating a cross-domain LU-LU session. It is generated by the SSCP receiving the INIT-SELF request, the SSCP(ILU).

- Queue the initiation request if queuing is requested by the end user, it is supported by the SSCP(s), and the DLU is currently unavailable. For a same-domain session, the INIT-SELF request is first processed and then queued until the LU(s) become available. For a cross-domain session, the SSCP(OLU) sends a CDINIT(Format 0 or 2) to the SSCP(DLU); after the CDINIT is processed and a positive response is returned, both SSCPs queue the CDINIT.

- Send a CDINIT, in the case of a cross-domain session, to transport the INIT-SELF request and resolved setup information to the SSCP(DLU) for distributed processing (see CDINIT for details). The CDINIT RU is format 2 (includes COS specification) if the SSCP(OLU) and SSCP(DLU) both support COS.

- When processing RSP(CDINIT), if the SSCP(DLU=SLU) has selected a COS name from the mode table because the ILU had not specified a COS name in the INIT-SELF request, then the SSCP(OLU=PLU) verifies that the COS name is a valid entry in the "COS name to VR identifier list" table. If not valid, it sends a -RSP(INIT-SELF, X'08610000'), thereby indicating the invalid COS name.

- Return a positive response to the INIT-SELF request once the resource availability, mode name, COS name, password, and requester ID are verified; a +RSP(CDINIT) is received (for a cross-domain session); and, if applicable, the initiation request is queued. The activation of the LU-LU session is completed sometime later. If an error occurs after a positive response has been sent, the ILU is notified via either NOTIFY(Vector Key X'03') or NSPE.

INIT-OTHER from the ILU requests the initiation of a session between the two LUs named in the RU. The requester may be a third-party LU or one of the two named LUs.

The session to be initiated may involve LUs in the same domain or in different domains. The INIT-OTHER request indicates, among other parameters, uninterpreted names of both LUs in the session to be initiated, together with optional URC and COS name fields.

The SSCP retains sufficient information (e.g., the network address of the ILU) in order later to be able to send NOTIFY(Vector Key X'03') to the ILU to report the status of the initiation, if requested by the NOTIFY specification in INIT-OTHER. If an initiation failure occurs, NOTIFY(Vector Key X'03') is sent to the ILU, independent of the NOTIFY specification in INIT-OTHER.

(SSCP,ILU).PRI.INIT(LU1,LU2)_RCV receives the INIT-OTHER request and, if it is valid, passes it to SSCP.SVC_MGR.SS, which may perform the following processing (when the ILU and OLU, or OLU and DLU, are in different domains, the processing is distributed among the corresponding SSCPs, each SSCP processing the portion that relates to the LU in its domain):

- Perform the same processing as described for INIT-SELF, except that relating to the ILU and SSCP(ILU).

- Resolve the uninterpreted names specified in the INIT-OTHER request to network names (performed by the SSCP(ILU)).

- Resolve the network name of the OLU to a network address if either LU1 or LU2 is in the same domain as the ILU.

- Retain the URC, if supplied in the request, for inclusion within any NOTIFY RU sent back to the ILU.

- When an INIT-OTHER is issued by a third-party ILU and a URC is specified, the URC is carried to neither the PLU nor the SLU.

- Determine that the necessary SSCP-SSCP session is active.

- Generate a PCID for the initiation procedure if at least one of the two LUs (LU1 or LU2) is not in the same domain as the ILU.

- Save the User Data field (at the SSCP(ILU)) for inclusion in the User Data field in CINIT or to pass to the SSCP(PLU) via INIT-OTHER-CD and/or CDINIT.

- Queue the initiation request if queuing is requested by the end user, it is supported by the SSCP(s), and LU1 or LU2 is currently unavailable. For a same-domain session, the INIT-OTHER (if SSCP(ILU) = SSCP(OLU)) or INIT-OTHER-CD (if SSCP(ILU) ¬= SSCP(OLU)) is first processed and then queued until the LU(s) become available. For a cross-domain session, the SSCP(OLU) sends a CDINIT(Format 0 or 2) to the SSCP(DLU); after the CDINIT is processed and a positive response is returned, both SSCPs queue the CDINIT.

- Send an INIT-OTHER-CD request to the SSCP(LU1|LU2), if neither LU1 nor LU2 is in the same domain as the ILU. The INIT-OTHER-CD RU is format 2 (includes COS specification) if the SSCP(ILU) and SSCP(OLU) both support COS.

- Send a CDINIT request to the SSCP(LU1|LU2), if either LU1 or LU2, but not both, is in the same domain as the ILU. (Note that in this case SSCP(ILU) = SSCP(OLU) and thus the SSCP.SVC_MGR.SS performs the same processing as that described under INIT-SELF.)

- Return a positive response to the INIT-OTHER request once the resource availability, mode name, COS name, password, and requester ID for both LUs are verified; a +RSP(CDINIT) is received (for a cross domain session); a +RSP(INIT-OTHER-CD) is received (for third-party SSCP); and, if applicable, the initiation request is queued. The activation of the LU-LU session is completed sometime later. If an error occurs after a positive response has been sent, the ILU is notified via NOTIFY(Vector Key X'03').

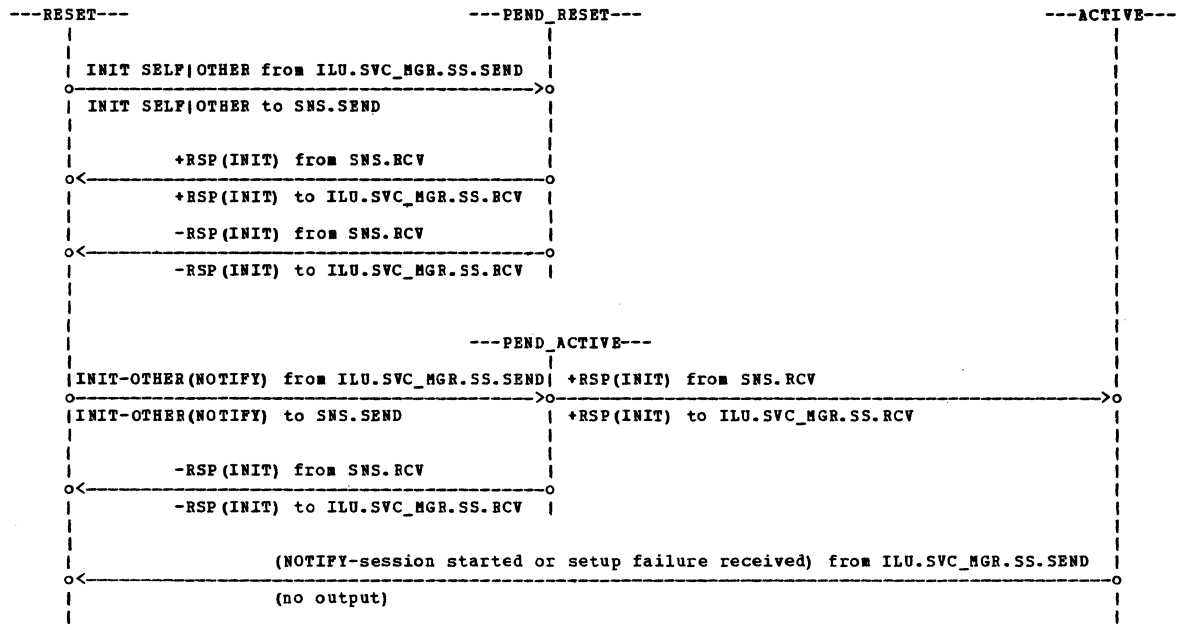
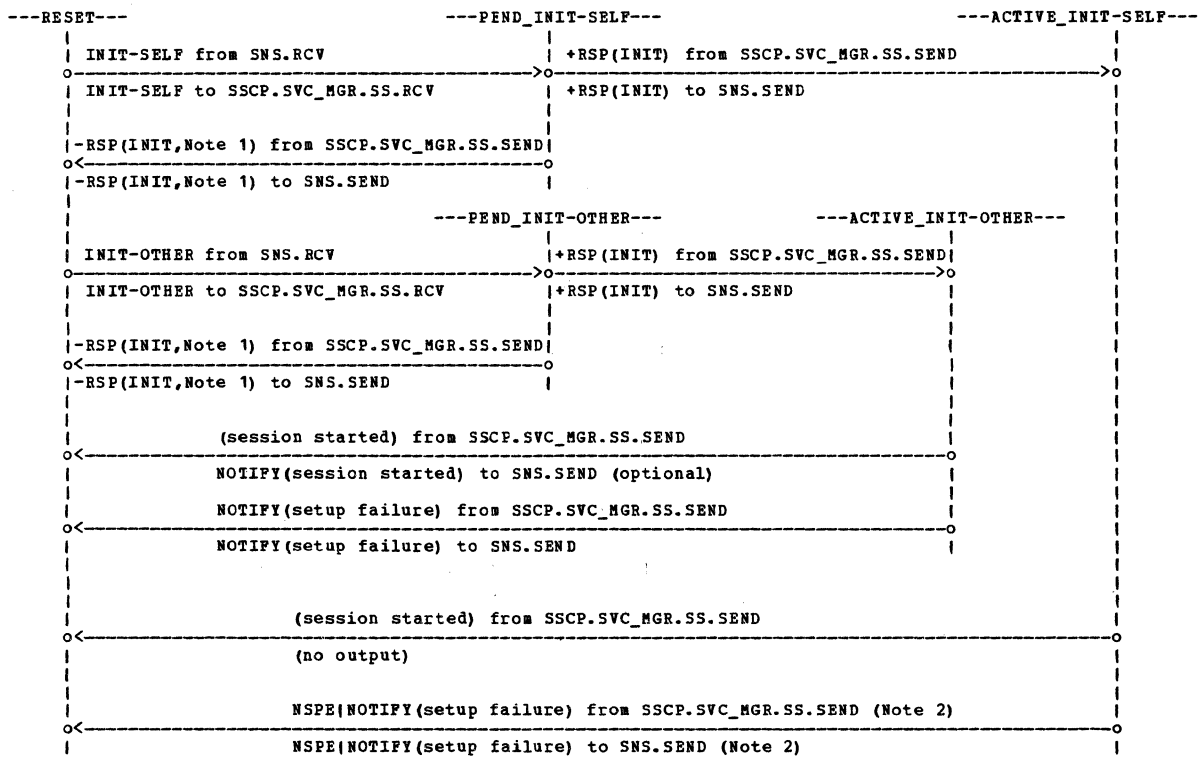


Figure 8-12. (SSCP,ILU).SEC.INIT((OLU,DLU)|(LU1,LU2))_SEND



Notes:

1. Sense codes: 0801,0803,0804,0805,0806,0809,080E,080F,0810,0812,0818,0826,0836,0837, 0838,0839,083A,083B,0841,0842,0844

2. NSPE is used if INIT-SELF format 0 has been issued; otherwise, NOTIFY is used.

Figure 8-13. (SSCP,ILU).PRI.INIT((OLU,DLU)|(LU1,LU2))_RCV

TERMINATE-SELF (TERM-SELF)
TERMINATE-OTHER (TERM-OTHER)

Flow: From TLU to SSCP(TLU) (Normal)

Principal FSMs:
(SSCP,TLU).SEC.TERM(SESSION_KEY_CONTENT|URC)_SEND
(Page 8-33)
(SSCP,TLU).PRI.TERM(SESSION_KEY_CONTENT|URC)_RCV
(Page 8-33)

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(s). The TERM-SELF request can explicitly indicate the uninterpreted name of the other LU with which the session(s) is to be terminated or can request, by not specifying the uninterpreted name, that all LU-LU sessions with the OLU be terminated. The sessions to be terminated may involve LUs in the same or in different domains.

TERM-SELF(Format 1) can also identify the session to be terminated via a network address pair or a URC session key. When the TERM-SELF is sent to terminate a parallel session after receipt of the CINIT or BIND carrying the assigned network address pair, the TERM-SELF carries the network-address-pair session key to identify the parallel session to be terminated; otherwise, when the TERM-SELF is sent prior to receipt of the CINIT or the BIND, the TERM-SELF carries the URC session key to identify the parallel session to be terminated. The optional URC field (distinct from the URC session key) can be specified in TERM-SELF(Format 1) for the TLU to correlate a TERM-SELF with NOTIFY(s).

The TERM-SELF request specifies (via the Type byte) the state(s) of the session(s) to be terminated:

- Active and pending active sessions
- Active, pending active, and queued sessions
- Queued sessions only

The TERM-SELF request designates (via the Type byte) the type of termination to be performed: Orderly, Forced, or Cleanup.

TERM-SELF(Orderly) requests that the SSCP(s) (via CDTERM(Orderly) and/or CTERM(Orderly) discussed later in this chapter) allow the PLU to execute an end-of-session procedure before the session is deactivated.

TERM-SELF(Forced) requests that the SSCP(s) (via CDTERM(Forced) and/or CTERM(Forced)) request the PLU to initiate session deactivation immediately and unconditionally. (The PLU user is also to be notified of the action.)

TERM-SELF(Cleanup) requests the SSCP(s) to initiate cleanup procedures for the PLU, boundary function, and SLU. The SSCP(OLU) and the SSCP(DLU) also clean up their LU-LU session-related information. In case of cross-domain session cleanup, the SSCP(OLU) begins the OLU-related cleanup procedure independently of the response to CDTERM.

The Type byte identifies which class of sessions involving the two LUs (OLU and DLU) are to be terminated when more than one session is active, pending active, or queued:

- Session(s) for which DLU is PLU
- Session(s) for which DLU is SLU
- Session(s) regardless of whether DLU is PLU or SLU

The SSCP(TLU) retains sufficient information (e.g., the network address of the TLU) in order later to be able to send NOTIFY(Vector Key X'03') to the TLU to report the status of the termination, if requested by the NOTIFY specification in TERM-SELF(Format 1). If a termination failure occurs, NOTIFY(Vector Key X'03') or NSPE is sent to the TLU, independent of the NOTIFY specification in TERM-SELF (NSPE is sent in lieu of NOTIFY only if TERM-SELF(Format 0) was received).

(SSCP,TLU).PRI.TERM(SESSION_KEY_CONTENT|URC)_RCV receives the TERM-SELF request and, if it is valid, passes it to SSCP.SVC_MGR.SS, which may perform the following processing (when LUs are in different domains, the processing is distributed among both SSCPs, each SSCP processing the portion that relates to the LU in its domain):

- Establish the authority of the end user and the TLU to request the termination of the specified session.
- Send a -RSP(TERM-SELF, X'0853'--Cleanup Required), if the TERM-SELF did not specify Cleanup and the SSCP-SSCP session with the SSCP having an active SSCP-LU session with the cross-domain LU is not active.
- If SSCP(TLU): Resolve the uninterpreted name of the DLU to a network name.

- If SSCP(DLU): Resolve network name of the DLU to a network address. This network address is returned to the SSCP(OLU) by the SSCP(DLU) in its response to a subsequent CDTERM (although the SSCP(OLU) may have saved the DLU network address from the LU-LU session initiation procedure).

- If SSCP(TLU): Retain the URC field, if one is supplied, for later inclusion within any NOTIFY RU sent back to the TLU.

- If SSCP(TLU): Generate a PCID for the termination procedure for a cross-domain LU-LU session.

For each session:

- If SSCP(OLU) and the two LUs are in different domains: Send a CDTERM(Orderly|Forced|Cleanup), as specified in the TERM-SELF, to transport the termination request to the SSCP(DLU) for distributed processing of the TERM-SELF.

- If SSCP(OLU): Determine which LU is the PLU and which, the SLU, based on information retained from the session initiation.

- If SSCP(OLU): Determine session(s) to be terminated based on the OLU being the PLU or SLU for each session (indicated by the Type byte of TERM-SELF).

- If SSCP(PLU): Send a CTERM(Orderly|Forced|Cleanup), as specified in the TERM-SELF, to the PLU.

- If SSCP(SLU) and the TERM-SELF specified Cleanup: Send a CLEANUP to the SLU (in a subarea node), or either DACTLU or ACTLU(Cold) to the SLU (in a peripheral node).

- If SSCP(OLU) and multiple sessions are to be terminated (the TERM-SELF carries the DLU Uninterpreted Name field and either the length value is 0, or it is non-0 and more than one parallel session is active with the specified DLU): Determine the network addresses and/or PCID session keys of the session partners for each of the sessions that the OLU is involved in for the class of sessions indicated by the Type byte of TERM-SELF. CDTERM, CTERM, and/or CLEANUP (or, either DACTLU or ACTLU(Cold)) is sent to each SSCP(DLUi)), PLUi, and/or SLUi, respectively, depending on which domain these LUs are in, as described previously. More than one CDTERM, CTERM, and/or CLEANUP is sent to the same SSCP or LU if more than one parallel session is to be terminated. Errors encountered during the processing of the individual session terminations are reported by

NOTIFY(Vector Key X'03') (or NSPE if TERM-SELF(Format 0) is used). If NOTIFY reply is specified in TERM-SELF(Format 1), a NOTIFY(Vector Key X'03') is sent when all termination procedures are completed.

- Return a positive response once the TERM-SELF request has been validated (e.g., password and authorization) and when at least one session has been recognized. If the TERM-SELF(Format 0) request is used and SSCP(OLU) ¬= SSCP(DLU), then a response may be delayed until the SSCP(OLU) receives a response to CDTERM from the SSCP(DLU). The deactivation of the LU-LU session is completed sometime later. If an error occurs after a positive response has been sent, the TLU is notified by either NOTIFY(Vector Key X'03') or NSPE.

TERM-OTHER from the TLU requests that the SSCP assist in terminating session(s) between the two LUs named in the RU. The requester may be a third-party LU or one of the two named LUs. The session(s) to be terminated may be between LUs in the same or in different domains. The TERM-OTHER indicates, via a session key, the uninterpreted names of both LUs (LU1 and LU2), the network address pair, or the URC for the session(s) to be terminated. The optional URC field (distinct from the URC session key) can be specified in TERM-OTHER for the TLU to correlate a TERM-OTHER with NOTIFY(s).

The TERM-OTHER request specifies (via the Type byte) the state(s) of session(s) to be terminated:

- Active and pending active sessions
- Active, pending active, and queued sessions
- Queued sessions only

The TERM-OTHER request designates (via the Type byte) Orderly, Forced, or Cleanup and the class of sessions to be terminated (as described for TERM-SELF).

The SSCP(TLU) retains sufficient information (e.g., the network address of the TLU) in order later to be able to send NOTIFY(Vector Key X'03') to the TLU to report the status of the termination, if requested by the NOTIFY specification in TERM-OTHER. If a termination failure occurs, NOTIFY(Vector Key X'03') is sent to the TLU, independent of the NOTIFY specification in TERM-OTHER.

(SSCP,TLU).PRI.TERM(SESSION_KEY_CONTENT|URC)_RCV receives the TERM-OTHER request and, if it is valid, passes it to SSCP.SVC_MGR.SS, which may perform the following processing:

- Perform the same processing as described for TERM-SELF, except that which relates to the TLU and SSCP(TLU).

- Establish the authority of the end user and the TLU to request the termination of the specified session(s).

- Send a -RSP(TERM-OTHER, X'0853'--Cleanup Required), if LU1 or LU2 (but not both) is in the same domain as the TLU, the TERM-OTHER did not specify Cleanup, and the SSCP-SSCP session with the SSCP having an active SSCP-LU session with the cross-domain LU is not active.

- Resolve the uninterpreted names specified in the TERM-OTHER request to network names (performed by the SSCP(TLU)).

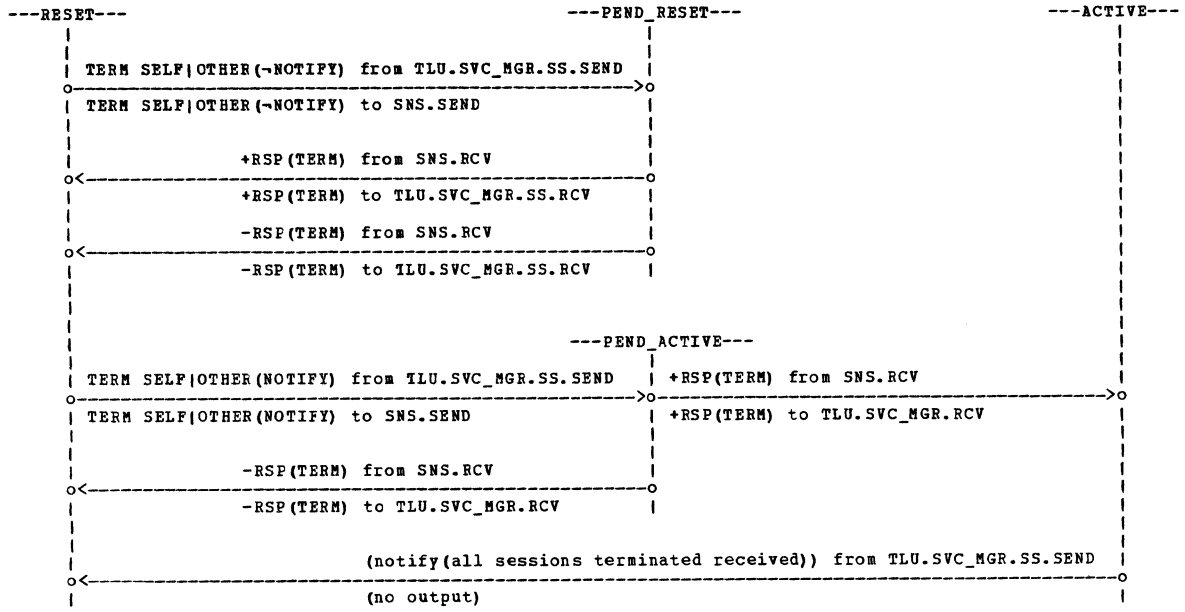
- Resolve the network name of the OLU to a network address if either LU1 or LU2 is in the same domain as the TLU.

- Retain the URC field, if one is supplied, for later use in any NOTIFY RU sent back to the TLU.

- Generate a PCID for the termination procedure if at least one of the two LUs (LU1 or LU2) is not in the same domain as the TLU.

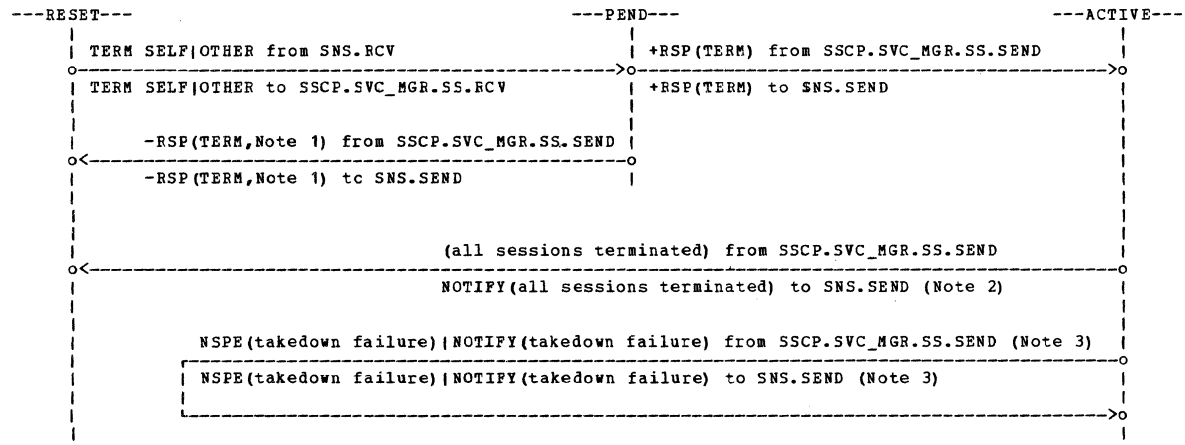
- Send a TERM-OTHER-CD if neither LU1 nor LU2 is in the same domain as the TLU. Note that the receiver of the TERM-OTHER-CD becomes the SSCP(OLU).

- Return a positive response once the TERM-OTHER has been validated (e.g., password, authorization, and receipt of a +RSP to TERM-OTHER-CD). The SSCP(OLU) sends +RSP(TERM-OTHER-CD) to the SSCP(TLU) when the setup processing portion of the initiation procedure has started and the session termination procedure has not completed; otherwise, -RSP(TERM-OTHER-CD) is sent. The deactivation of the LU-LU session is completed sometime later. If an error occurs after a positive response has been sent, then the TLU is notified via NOTIFY(Vector Key X'03').



Note: One of these FSMs exists for every SESSION_KEY_CONTENT (for a single session TERM) or URC (for multiple session TERM) that is used to correlate NOTIFY(s) to a prior TERM(Format 1).

Figure 8-14. (SSCP,TLU).SEC.TERM(SESSION_KEY_CONTENT|URC)_SEND



Notes:

1. Sense codes: 0803,0804,0806,0809,080E,080F,0810,0812,0816,081E,0835,0839, 083B,083E,083F,0842,0853,1001

   NOTIFY is sent if NOTIFY reply was specified in TERM and when all requested sessions are terminated.

   NSPE is used if TERM-SELF(Format 0) has been issued; otherwise, NOTIFY is used.

Figure 8-15. (SSCP,TLU).PRI.TERM(SESSION_KEY_CONTENT|URC)_RCV

CONTROL INITIATE (CINIT)
CONTROL TERMINATE (CTERM)
SESSION STARTED (SESSST)
SESSION ENDED (SESSEND)
BIND FAILURE (BINDF)
UNBIND FAILURE (UNBINDF)

Flow: From SSCP to PLU (Normal) for CINIT and CTERM;
from PLU to SSCP (Normal) for SESSST, BINDF, UNBINDF;
from LU to SSCP (Normal) for SESSEND

Principal FSMs:

CINIT requests the PLU to attempt to activate, via a BIND request, a session with the specified SLU. CINIT is sent to the PLU with definite response requested.

The suggested parameters for BIND (the "BIND image"), mode name, COS name, virtual route information (the type of VR required and the VR identifier list), the (PLU,SLU) network address pair, and the SLU network name are included as parameters in CINIT. The BIND parameters are selected by the SSCP.SVC_MGR.SS, based on optional implementation- and installation-specified parameters for the specific LU, and on the mode name parameter in the INIT that prompted the CINIT. The PLU uses the network address pair provided in the CINIT RU, but may modify the parameters from CINIT, except for the pacing parameters, maximum RU sizes, cryptography, URC field, SLU name and PLU name.

When an INIT (SELF or OTHER) is issued from an SLU, the SSCP(PLU) places the uninterpreted name of the PLU, as received in the INIT RU (same domain session), or as carried in the DLU Uninterpreted Name field in CDINIT (cross domain session), into the PLU Name field of the BIND image; otherwise, the SSCP(PLU) places the network name of the PLU into this field.

Mode name, COS name, and virtual route information are not included in the BIND RU. The PLU.SVC_MGR.SS passes COS name and virtual route information to PU.SVC_MGR.CSC_MGR as parameters to be used in the selection of the virtual route to be used by the subject session. The PLU.SVC_MGR.SS also retains the mode name and COS name, which it may use in a subsequent INIT-SELF or INIT-OTHER request, if it is necessary to restart the same LU-LU session with the original session characteristics.

The PLU may change the primary CPMGR's receive pacing count—but not to 0, as 0 indicates no pacing of requests to the primary CPMGR. If this count is changed, and the staging indicator specifies one-stage, the secondary CPMGR's send pacing count is made equal to the primary CPMGR's receive pacing count. The PLU may also change the maximum RU sizes that are used on the normal flows. 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 Chapter 13 for additional rules on TS Profile and TS Usage modifications that are allowed on the BIND parameters.

The ILU identification and password may be forwarded to the PLU, where they can be used to determine the authority of the initiating LU.SVC_MGR.SS or LU end user. A user field in CINIT is also passed to the PLU.

If both the PLU and SLU have cryptographic capability, the SSCP.SVC_MGR.SS inserts the session cryptography key twice into the CINIT--once enciphered under the PLU master cryptography key and once enciphered under the SLU cryptography key; the former is used at the PLU, while the latter is passed by the PLU in BIND for use at the SLU. The SSCP.SVC_MGR.SS also sets the cryptography option flags to the highest level of cryptography (see BIND in Chapter 13) as requested by the INIT (via the mode name field designation of the BIND parameters) or by implementation- and installation-dependent descriptions of the LUs known to the SSCP.SVC_MGR.SS. The session cryptography key is a pseudo random number that the SSCP.SVC_MGR.SS obtains from a UPM.

If a URC is supplied to the SSCP(PLU), it is carried in the BIND image of CINIT, as described for INIT earlier in this chapter.

CTERM requests that the PLU attempt to deactivate a session with the specified (PLU,SLU) network address pair. CTERM is sent to the PLU with definite response requested. The CTERM may be designated Orderly, Forced, or Cleanup.

CTERM(Orderly) allows the PLU to delay deactivating the session.

CTERM(Forced) requires an unconditional attempt to deactivate the session via UNBIND (optionally preceded by CLEAR).

CTERM(Cleanup) is equivalent to CTERM(Forced), except that the UNBIND resulting from this CTERM internally triggers a +RSP(UNBIND), since cleanup termination is used in instances when an LU needs to unilaterally deactivate a session,

without waiting for synchronization with the other half-session. (The association of the HSCB for the (PLU,SLU).PRI half-session with a VRCB is broken when PU.SVC_MGR.CSC_MGR processes this RSP(UNBIND).)

The PLU may send UNBIND without receiving a CTERM request from its SSCP to deactivate one of its own active sessions.

SESSST is sent, with no-response requested, by the PLU to notify the SSCP that the session between the specified LUs has been successfully activated.

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; the reason for the failure is indicated by a parameter of the request.

SESSEND is sent, with no-response requested, by the PLU, the SLU (in a subarea node only), or the BF.LU.SVC_MGR on behalf of the SLU to notify the SSCP that the session between the specified LUs has been successfully deactivated.

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 (e.g., because of a path failure).

```
FSM_SSCP_PLU_PRI_CSESS_SEND: FSM_DEFINITION;
```

STATE NAMES-->	RESET	PEND ACTIVE CINIT	PEND ACTIVE SESSST	ACTIVE
STATE NUMBERS-->	1	2	3	4
INPUT				
S,RQ,CINIT	2(A)	>(S)	>(S)	>(S)
R,+RSP,CINIT	/	3(B)	/	/
R,-RSP,CINIT /* NOTE */	/	1(B1)	/	/
R,RQ,SESSST	>	>	4(B)	>
R,RQ,BINDF	>	>	1(B1)	>
R,RQ,SESSEND	>	>	1(B1)	1(B1)
R,RQ,UNBINDF	>	>	1(B1)	1(B1)
'RESET'	—	1	1	1

OUTPUT CODE	FUNCTION
A	SEND MU TO SNS.SEND;
B	SEND MU TO SSCP.SVC_MGR.SS.RCV;
B1	SEND MU TO SSCP.SVC_MGR.SS.RCV; CALL FSM_SSCP_PLU_PRI_CTERM_SEND('RESET');
S	SEND SEND_CHECK TO SSCP.SVC_MGR.SS.SEND;

```
/*
NOTE:  SENSE CODES FOR -RSP(CINIT): 0801, 0803, 0804, 0805,
       080A, 080E, 080F, 0810, 0812, 0821, 0832, 0835, 0848
                                                          */
END FSM_SSCP_PLU_PRI_CSESS_SEND;
```

```
FSM_SSCP_PLU_SEC_CSESS_RCV: FSM_DEFINITION;
```

STATE NAMES-->	RESET	PEND ACTIVE CINIT	PEND ACTIVE SESSST	ACTIVE
STATE NUMBERS-->	1	2	3	4
INPUT				
R,RQ,CINIT	2(A)	>	>	>
S,+RSP,CINIT	>(S)	3(B)	>(S)	>(S)
S,-RSP,CINIT /* NOTE */	>(S)	1(B1)	>(S)	>(S)
S,RQ,SESSST	>(S)	>(S)	4(B)	>(S)
S,RQ,BINDF	>(S)	>(S)	1(B1)	>(S)
S,RQ,SESSEND	>(S)	>(S)	1(B1)	1(B1)
S,RQ,UNBINDF	>(S)	>(S)	1(B1)	1(B1)
'RESET'	–	1	1	1

OUTPUT CODE	FUNCTION
A	SEND MU TO PLU.SVC_MGR.SS.RCV;
B	SEND MU TO SNS.SEND;
B1	SEND MU TO SNS.SEND; CALL FSM_SSCP_PLU_SEC_CTERM_RCV('RESET');
S	SEND SEND_CHECK TO PLU.SVC_MGR.SS.SEND;

```
                                                                    /*
NOTE:  SENSE CODES FOR -RSP(CINIT): 0801, 0803, 0804, 0805,
       080A, 080E, 080F, 0810, 0812, 0821, 0832, 0835, 0848
                                                                    */
END FSM_SSCP_PLU_SEC_CSESS_RCV;
```

```
FSM_SSCP_PLU_PRI_CTERM_SEND: FSM_DEFINITION;
```

STATE NAMES-->	RESET	PEND CTERM ORDERLY FORCED	PEND CTERM CLEANUP
STATE NUMBERS-->	1	2	3
INPUT			
S,RQ,CTERM,ORDERLY|FORCED	2(A)	-(A)	>(S)
S,RQ,CTERM,CLEANUP	3(A)	3(A)	-(A)
R,+RSP,CTERM, LAST	-(C)	1(B)	1(B1)
R,-RSP,CTERM, LAST /* NOTE */	-(C)	1(B)	1(B1)
R,±RSP,CTERM,¬LAST /* NOTE */	-(C)	-(C)	-(C)
'RESET'	-	1	1

OUTPUT CODE	FUNCTION
A	SEND MU TO SNS.SEND;
B	SEND MU TO SSCP.SVC_MGR.SS.RCV;
B1	SEND MU TO SSCP.SVC_MGR.SS.RCV; CALL FSM_SSCP_PLU_PRI_CSESS_SEND('RESET');
C	DISCARD MU;
S	SEND SEND_CHECK TO SSCP.SVC_MGR.SS.SEND;

```
                                                                    /*
NOTE:  SENSE CODES FOR -RSP(CTERM): 0803, 0804, 080A, 080E,
       080F, 0810, 0816
                                                                    */
END FSM_SSCP_PLU_PRI_CTERM_SEND;
```

```
FSM_SSCP_PLU_SEC_CTERM_RCV: FSM_DEFINITION;
```

STATE NAMES-->	RESET	PEND CTERM ORDERLY FORCED	PEND CTERM CLEANUP
STATE NUMBERS-->	1	2	3
INPUT			
R,RQ,CTERM,ORDERLY|FORCED	2(A)	>	>
R,RQ,CTERM,CLEANUP	3(A)	>	>
S,+RSP,CTERM,	-(B)	1(B)	1(B1)
S,-RSP,CTERM, /* NOTE */	-(B)	1(B)	1(B1)
'RESET'	-	1	1

OUTPUT CODE	FUNCTION
A	SEND MU TO PLU.SVC_MGR.SS.RCV;
B	SEND MU TO SNS.SEND;
B1	SEND MU TO SNS.SEND; CALL FSM_SSCP_PLU_SEC_CSESS_RCV('RESET');

```
                                                                    /*
NOTE:  SENSE CODES FOR -RSP(CTERM): 0803, 0804, 080A, 080E,
       080F, 0810, 0816
                                                                    */
END FSM_SSCP_PLU_SEC_CTERM_RCV;
```

CLEANUP

Flow: SSCP(SLU) to SLU (Normal)

Principal FSMs:
(SSCP,SLU).PRI.CLEANUP_SEND (Page 8-42)
(SSCP,SLU).SEC.CLEANUP_RCV (Page 8-42)

CLEANUP is sent by the SSCP to the SLU (in a subarea node only) requesting that the SLU attempt to deactivate the session for the specified (PLU,SLU) network address pair. The UNBIND resulting from CLEANUP internally triggers a +RSP(UNBIND), since cleanup termination is used in instances when an LU needs to unilaterally deactivate a session, without waiting for synchronization with the other half-session.

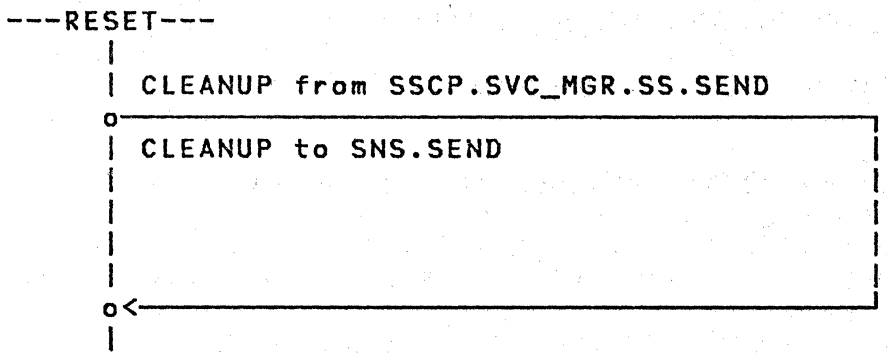


Figure 8-16. (SSCP,SLU).PRI.CLEANUP_SEND

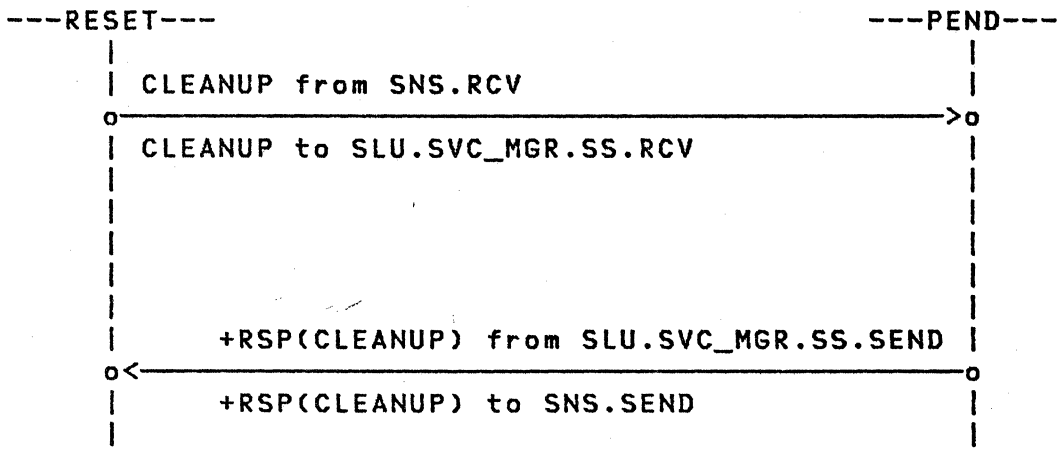


Figure 8-17. (SSCP,SLU).SEC.CLEANUP_RCV

NETWORK SERVICES PROCEDURE ERROR (NSPE)

Flow: From SSCP to ILU or TLU (Normal)

Principal FSMs:

(SSCP,ILU).PRI.INIT((OLU,DLU)|(LU1,LU2))_RCV
(Page 8-27)

(SSCP,TLU).PRI.TERM(SESSION_KEY_CONTENT|URC)_RCV
(Page 8-33)

NSPE is used by the SSCP to inform an ILU or TLU that a session initiation or termination attempt has failed after a positive response has been sent to the corresponding initiation or termination request. NSPE is used only if format 0 of INIT-SELF or TERM-SELF was issued; otherwise, NOTIFY(Vector Key X'03') is used.

An NSPE is also sent to the TLU that issued a TERM-SELF(Format 0) requesting multiple session terminations to identify each session that has failed the termination process (one NSPE per termination failure) after a positive response has been sent to the TERM-SELF. (A negative response to CDTERM is an example of a failure causing NSPE to be sent.)

NOTIFY (NOTIFY)

Flow: From SSCP to SSCP or LU and from LU to SSCP (Normal)

Principal FSMs:

(SSCP,ILU).SEC.INIT ((OLU,DLU)|(LU1,LU2))_SEND
(Page 8-27)
(SSCP,ILU).PRI.INIT((OLU,DLU)|(LU1,LU2))_RCV
(Page 8-27)
(SSCP,TLU).SEC.TERM(SESSION_KEY_CONTENT|URC)_SEND
(Page 8-33)
(SSCP,TLU).PRI.TERM(SESSION_KEY_CONTENT|URC)_RCV
(Page 8-33)
((SSCP,SSCP').SSCP)|((SSCP,LU).PRI)|((SSCP,LU).SEC).
NOTIFY_SEND (Page 8-47)
((SSCP,SSCP').SSCP)|((SSCP,LU).SEC)|((SSCP,LU).PRI).
NOTIFY_RCV (Page 8-47)
(SSCP(OLU),SSCP(ILU)).SSCP(ILU).INIT-OTHER-CD
(OLU,DLU,PCID)_SEND (Page 8-54)
(SSCP(OLU),SSCP(ILU)).SSCP(OLU).INIT-OTHER-CD
(OLU,DLU,PCID)_RCV (Page 8-54)
(SSCP(OLU),SSCP(TLU)).SSCP(TLU).TERM-OTHER-CD
(SESSION_KEY_CONTENT,PCID)_SEND (Page 8-64)
(SSCP(OLU),SSCP(TLU)).SSCP(OLU).TERM-OTHER-CD
(SESSION_KEY_CONTENT,PCID)_RCV (Page 8-64)

NOTIFY is used to send information from an SSCP to another SSCP or to an LU, or from an LU to an SSCP. NOTIFY carries information in the form of a (vector key, vector data) pair:

- Vector key X'01'—resource requested: Sent in NOTIFY from an SSCP to the current users (LUs) of a resource (LU) to inform them that another LU wishes to use the resource. The current user(s) may be in the same domain as the SSCP, or in a different domain; in the latter case, the NOTIFY flows from SSCP to SSCP to LU.

- Vector key X'03'—ILU|TLU notification or third-party SSCP notification: For ILU|TLU notification, it is sent in NOTIFY from the SSCP(ILU) to the ILU or from the SSCP(TLU) to the TLU in order to provide session initiation or termination status, if requested by the NOTIFY specification in INIT or TERM. If a session initiation or termination attempt has failed after a positive response has been sent to the INIT or TERM, NOTIFY is sent independent of the NOTIFY specification in the INIT or TERM request. NOTIFY is also sent to the TLU that issued a TERM to terminate multiple sessions, to identify each session that has failed the termination process (one NOTIFY per termination failure) after a positive response has been sent to the TERM. If the ILU sends INIT-OTHER and a requested parallel session is initiated, the NOTIFY session key parameter includes the network address pair that can be

used by the third-party TLU (=ILU) to terminate the parallel session. For ILU notification, NOTIFY is sent only if INIT-SELF(Format 1 or 2) or INIT-OTHER was issued; likewise, for TLU notification, it is sent only if TERM-SELF(Format 1) or TERM-OTHER was issued.

For third-party SSCP notification, the vector key X'03' is sent in NOTIFY from the SSCP(OLU) to a third-party SSCP that issued an INIT-OTHER-CD in order to provide session initiation status, as requested by the NOTIFY specification in the INIT-OTHER-CD. NOTIFY is also sent from the SSCP(OLU) to a third-party SSCP that issued a TERM-OTHER-CD in order to provide session termination status. Additionally, NOTIFY is sent to the third-party SSCP that issued a TERM-OTHER-CD to terminate multiple sessions, to identify each session that has failed the termination process (one NOTIFY per termination failure) after a positive response has been sent to the TERM-OTHER-CD. If an INIT-OTHER-CD results in the initiation of a parallel session, the NOTIFY session key parameter includes the network address pair that can be used by the third-party SSCP(TLU=ILU) to terminate the parallel session.

When TERM-SELF or TERM-OTHER specifies session key X'0A' (URC session key), or TERM-OTHER-CD specifies session key X'05' (PCID session key), NOTIFY returns the same session key.

- Vector key X'04'—LU notification: Sent in NOTIFY from an SSCP to an LU informing the LU of the completed termination of the identified LU-LU session, the cause of the termination, and the action, if any, to be taken by the LU to reinitiate the session.

- Vector key X'0C'—LU-LU session services capabilities: Sent in NOTIFY from an LU to its 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. Whenever an event occurs during an active SSCP-LU session causing one or more of these parameters to change, the LU sends the NOTIFY to its SSCP to convey its new session services capabilities. (At SSCP-LU session activation time, the LU's session services capabilities are conveyed to the SSCP via control vector X'0C' carried in the LU's +RSP(ACTLU)).

The SSCP uses these parameters to determine whether an LU is available for activation of an LU-LU session. In terms of these parameters, an LU is available when all of the following conditions are met:

-- Its session count is less than its session limit.

-- It can act as a PLU or SLU, as requested in the INIT (or CDINIT) request.

-- It supports parallel sessions (if at least one session between the designated LUs is already active).

Otherwise, the LU is unavailable for activation of an LU-LU session.

The SSCP also uses these parameters, other than the parallel-session support, to determine whether to queue an INIT (or CDINIT) request, provided queuing is specified in the request and supported by the SSCP(s):

-- When the SSCP receives an initiation request for a session with an LU that is currently unavailable, because either its session count equals its session limit or it cannot comply with the PLU|SLU specification, and queuing is specified and supported, the SSCP queues the initiation request.

-- When the SSCP receives an initiation request for a session with an LU that is currently unavailable and either queuing is not specified or not supported, or the LU does not support parallel sessions and a session between the designated LUs is already active, the initiation request is rejected (a negative response is returned).

-- When the SSCP receives a NOTIFY indicating the LU has become available, the SSCP dequeues initiation requests (up to the session limit) for that LU, resuming the session-initiation process.

-- When the SSCP receives an initiation request for a session with an LU 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.

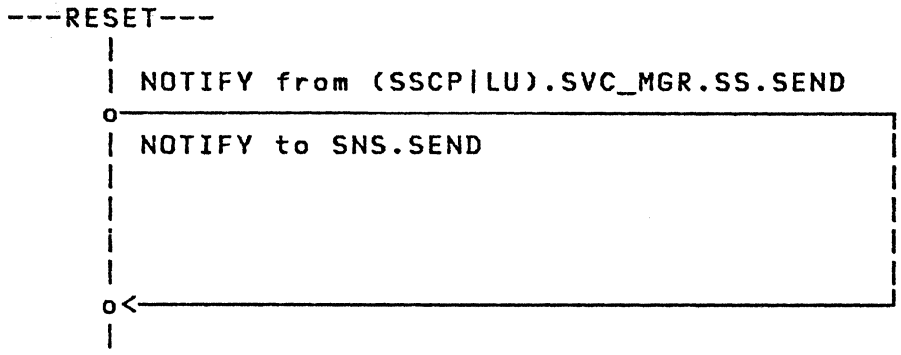


Figure 8-18. ((SSCP,SSCP').SSCP)|((SSCP,LU).PRI)|((SSCP,LU).SEC). NOTIFY_SEND

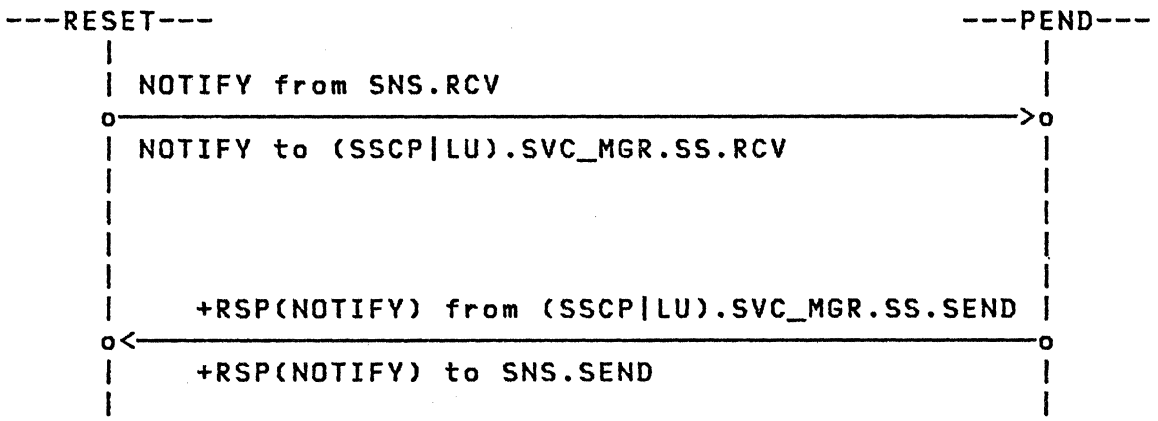


Figure 8-19. ((SSCP,SSCP').SSCP')|((SSCP,LU).SEC)|((SSCP,LU).PRI). NOTIFY_RCV

CROSS-DOMAIN INITIATE (CDINIT)

Flow: From SSCP(OLU) to SSCP(DLU) (Normal)

Principal FSMs:
(SSCP(DLU),SSCP(OLU)).SSCP(OLU).CDINIT(OLU,DLU,PCID)_SEND
(Page 8-51)
(SSCP(DLU),SSCP(OLU)).SSCP(DLU).CDINIT(OLU,DLU,PCID)_RCV
(Page 8-52)

CDINIT from the SSCP(OLU) requests that the SSCP(DLU) assist in initiating an LU-LU session for the specified (OLU,DLU) pair.

CDINIT has three formats: 0, 1, and 2.

- Format 0 is used when first attempting to set up the session (Type = initiate only, initiate or queue, or queue only).

- Format 1 (Type = dequeue) is used to retry session setup when an LU becomes available and a previous Format 0 or 2 CDINIT was queued. See INIT-SELF for further description of queuing.

- Format 2 is identical to format 0, except that it adds COS name initialization indicators and COS name.

(SSCP(DLU),SSCP(OLU)).SSCP(DLU).CDINIT(OLU,DLU,PCID)_RCV receives the CDINIT request and, if it is valid, passes it to SSCP(DLU).SVC_MGR.SS, which may perform the following processing:

- Resolve the network name of the DLU to a network address.

- Establish the availability of the requested LU (e.g., complies with the PLU|SLU specification, not yet at session limit)

- Determine that a path exists between the DLU and SSCP(DLU); this may require configuration services to establish a connection via a switched link.

- Establish the authority of the requester (an end user) and the OLU to access the DLU. The password may be used to verify the identity of the requester.

- Determine which LU is to be the primary LU (PLU) for the session, as specified in the CDINIT request.

- Assign a network address to the DLU, if required. The SSCP(DLU) issues RNAA to the PU of the DLU if DLU=PLU and if the DLU supports parallel sessions. If a network address cannot be assigned by the PU, a negative response—Insufficient Resource (X'0812')—is returned to CDINIT.

- Select session parameters for the BIND image (if SSCP(DLU) = SSCP(SLU)) based on the mode name parameter in the request, and on optional implementation- and installation-specific parameters for the specific LUs.

- Verify (when DLU=PLU) that the COS name is a valid entry in the "COS name to VR identifier list" table. If not valid, it sends a -RSP(CDINIT, X'08610000'), thereby indicating the invalid COS name.

- Derive a COS name (when DLU=SLU and the ILU did not specify a COS name) from the mode table and place it in the RSP(CDINIT).

- Specify in RSP(CDINIT) whether the SLU or BF supports sending UNBIND and SESSEND. The specification is in the LU status byte, byte 7, bit 5, of RSP(CDINIT).

A positive response is returned to the CDINIT request once the LU availability, mode name, COS name, password, and requester ID are verified and, if applicable, the CDINIT request is queued. Information about the DLU is returned in the response to the CDINIT request. At the completion of the processing of CDINIT and its response, both SSCPs have:

- The network names and network addresses of both LUs (OLU and DLU). (Format 0 or 2 of CDINIT carries the DLU uninterpreted name as specified in INIT-SELF, or in INIT-OTHER when ILU = OLU; otherwise, the DLU uninterpreted name is omitted.)

- The uninterpreted LU name used in the original session initiation request (if INIT-SELF originated the request).

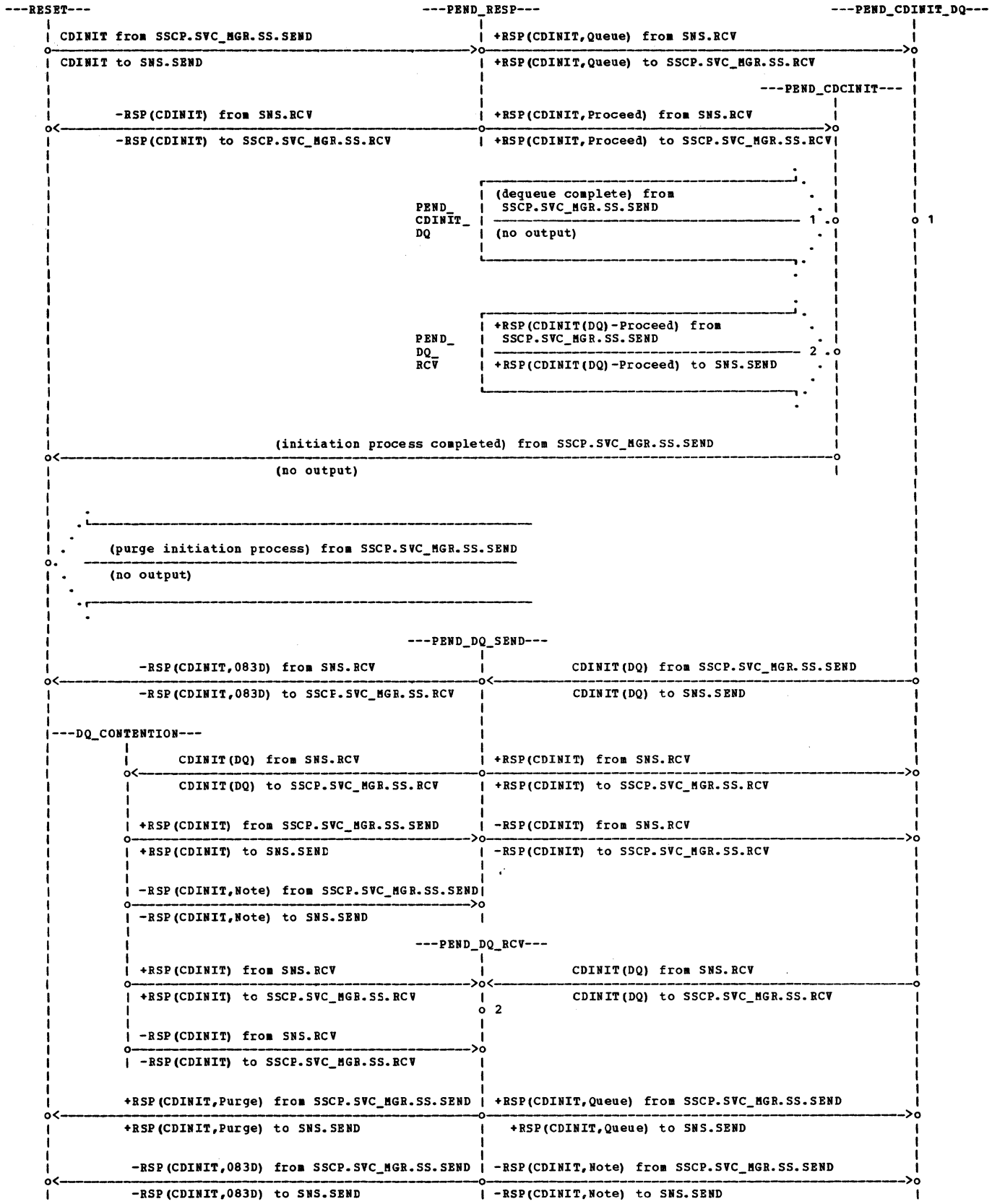
- The PCID used to correlate the initiation procedure (the PCID is generated by the SSCP(ILU)).

- The status of the LUs (e.g., available) and of the CDINIT procedure (e.g., initiated successfully, queued).

- The mode name, COS name, requester ID, password, and user field.

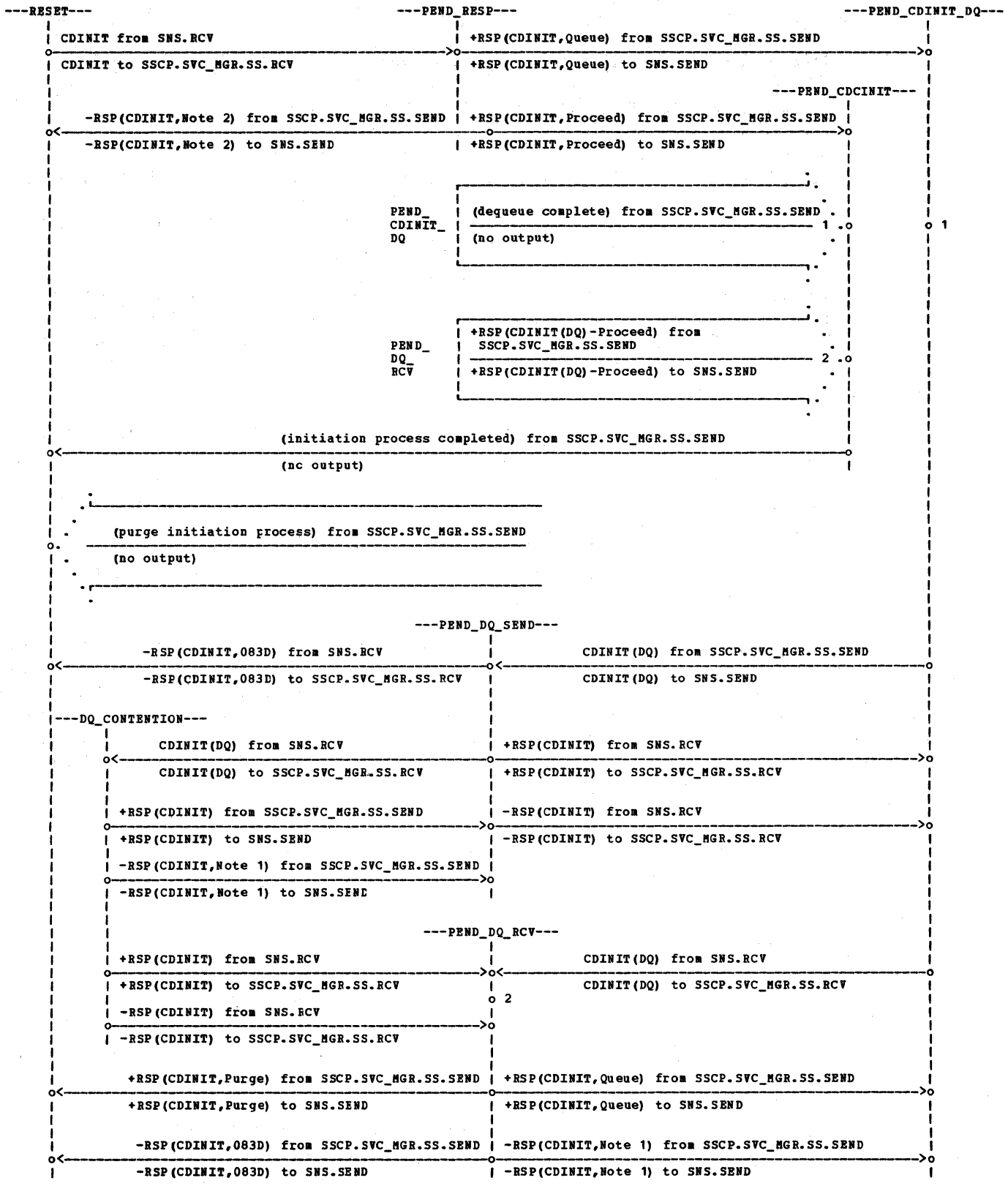
If queuing is specified (and necessary) and supported by the SSCPs, then processing of the CDINIT and its response consists of both SSCPs queuing the CDINIT request until a later event (e.g., receipt of NOTIFY(Vector Key X'0C') indicating the LU is now available for activation of an LU-LU session) causes dequeuing.

When a positive response to CDINIT has been returned and both LUs are available for an active session, the SSCP(SLU) issues a CDCINIT to the SSCP(PLU).



Note: Sense Codes: 0844,1003

Figure 8-20. (SSCP(DLU),SSCP(OLU)).SSCP(OLU).CDINIT(OLU,DLU,PCID)_SEND



Notes:

1. Sense Codes: 0844,1003

2. Sense Codes: 0801,0803,0804,0805,0806,080E,080F,0810,0812,0818,0836,0837,0838,0839,083A,083B,0841

Figure 8-21. (SSCP(DLU),SSCP(OLU)).SSCP(DLU).CDINIT(OLU,DLU,PCID)_RCV

INITIATE-OTHER CROSS-DOMAIN (INIT-OTHER-CD)

Flow: From SSCP(ILU) to SSCP(OLU) (Normal)

Principal FSMs:
(SSCP(OLU),SSCP(ILU)).SSCP(ILU).INIT-OTHER-CD(OLU,DLU,PCID)_
SEND (Page 8-54)
(SSCP(OLU),SSCP(ILU)).SSCP(OLU).INIT-OTHER-CD(OLU,DLU,PCID)_
RCV (Page 8-54)

INIT-OTHER-CD from the SSCP(ILU) requests that a session be initiated between the two LUs named in the RU. The INIT-OTHER-CD request simply transports an INIT-OTHER from the SSCP(ILU) (a third-party SSCP in this case) to the SSCP(OLU).

(SSCP(OLU),SSCP(ILU)).SSCP(OLU).INIT-OTHER-CD(OLU,DLU,PCID)_RCV receives the INIT-OTHER-CD request and, if it is valid, passes it to SSCP(OLU).SVC_MGR.SS, which performs the same basic processing described for the SSCP(OLU) for INIT, in addition to the following:

- Retain the PCID for later use in sending NOTIFY RUs back to the SSCP(ILU).

A positive response is returned to the INIT-OTHER-CD request once the LU availability, mode name, COS name, password, and requester ID are verified for both LUs; a +RSP(CDINIT) is received (for a cross-domain session); and, if applicable, the initiation request is queued. If errors occur after a positive response has been sent, then the SSCP(ILU) is notified via NOTIFY(Vector Key X'03').

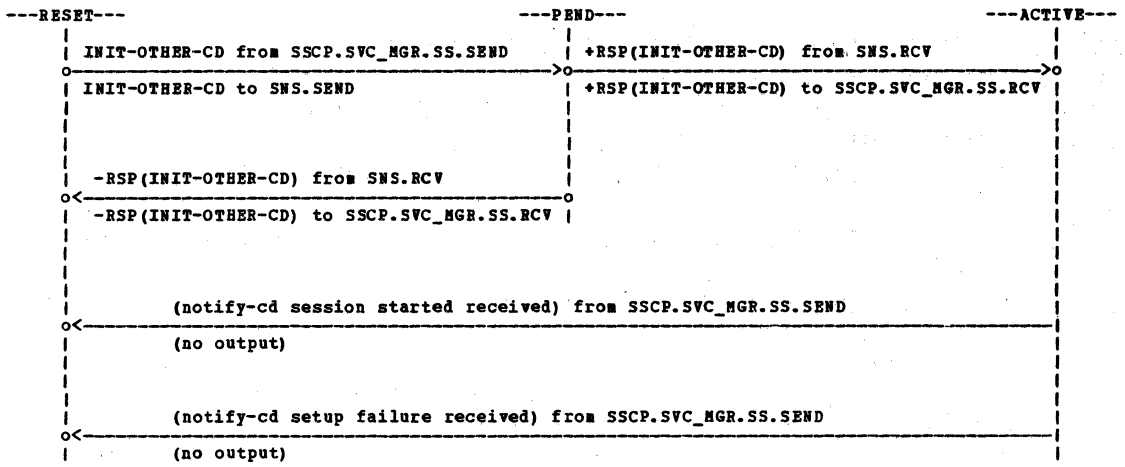
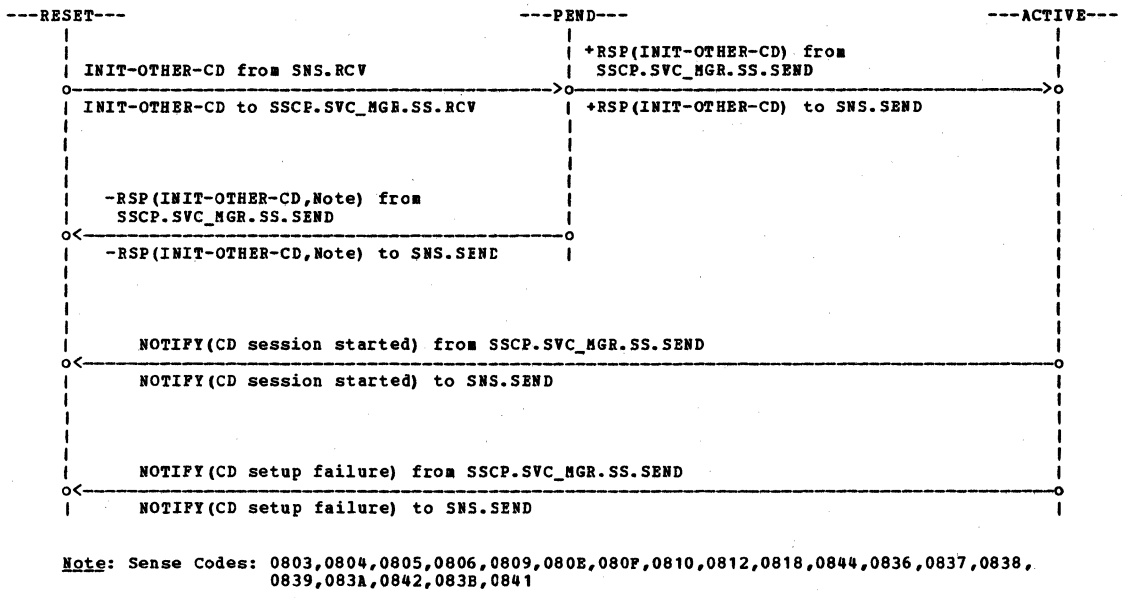


Figure 8-22. (SSCP(OLU),SSCP(ILU)).SSCP(ILU).INIT-OTHER-CD(OLU,DLU,PCID)_SEND



Note: Sense Codes: 0803,0804,0805,0806,0809,080E,080F,0810,0812,0818,0844,0836,0837,0838, 0839,083A,0842,083B,0841

Figure 8-23. (SSCP(OLU),SSCP(ILU)).SSCP(OLU).INIT-OTHER-CD (OLU,DLU,PCID)_RCV

CROSS-DOMAIN CONTROL INITIATE (CDCINIT)
CROSS-DOMAIN SESSION STARTED (CDSESSST)
CROSS-DOMAIN SESSION SETUP FAILURE (CDSESSSF)
CROSS-DOMAIN SESSION ENDED (CDSESSEND)
CROSS-DOMAIN SESSION TAKEDOWN FAILURE (CDSESSTF)

Flow: From SSCP(SLU) to SSCP(PLU) (Normal) for CDCINIT;
from SSCP to SSCP (Normal) for CDSESSEND;
from SSCP(PLU) to SSCP(SLU) (Normal) for the others

Principal FSMs:
(SSCP(PLU),SSCP(SLU)).SSCP(SLU).CDCSESS(PLU,SLU,PCID)
(Page 8-58)
(SSCP(PLU),SSCP(SLU)).SSCP(PLU).CDCSESS(PLU,SLU,PCID)
(Page 8-59)

CDCINIT passes information about the SLU from the SSCP(SLU) to the SSCP(PLU) and requests that the SSCP(PLU) send CINIT to the PLU. The information passed by CDCINIT includes the BIND image selected by the mode name parameter (in the preceding INIT), along with a cryptography key and LU or device characteristics. The PCID and the (PLU,SLU) network addresses are also passed in the RU. If an INIT is issued by the SLU (SLU=ILU) and a URC is supplied, the SSCP(SLU) places the URC in the BIND image of CDCINIT, as described for INIT earlier in this chapter.

The BIND image passed by CDCINIT contains an uninitialized PLU name field (i.e., its contents are to be ignored); the SSCP(PLU) inserts the PLU name into the BIND image when it builds a CINIT RU. The SSCP(PLU) may modify the parameters from CDCINIT (for use in CINIT), except for pacing parameters, maximum RU sizes, URC field, cryptography options, SLU name, and PLU name. The primary CPMGR receive pacing count may be decreased by the SSCP(PLU)--but not to 0, as 0 indicates no pacing of requests to the primary CPMGR. If this count is changed, and the staging indicator for secondary-to-primary pacing is set for one stage, the secondary CPMGR send pacing is set equal to the primary CPMGR receive pacing count. The SSCP(PLU) may also decrease the normal-flow maximum RU sizes.

The changing of any of the pacing parameters and maximum RU sizes for one session may affect the performance characteristics of that session and of concurrently active sessions that share network resources with it. The nonallowable and allowable changes to the BIND image are summarized in "BIND Image and BIND RU Modification Table" in Chapter 13, under the description of BIND.

After the SSCP(PLU) successfully processes the CDCINIT request, it returns a positive response to the SSCP(SLU) and sends a CINIT to the PLU. When building CINIT, the SSCP performs the following functions related to COS processing:

- Derives the VR identifier list from the COS name and includes it in the Mode/Class of Service/VR Identifier List control vector (key X'0D'), which is used in the CINIT RU.

- If the SSCP(SLU) does not support virtual route protocols, the SSCP(PLU) initializes the type of virtual route required (byte 20 of the above control vector) to X'00', indicating that virtual routes mapping to ER0 must be used for the reverse ERN. Otherwise, it indicates that virtual routes mapping to any reverse ERN may be used for the subject LU-LU session.

  The requirement for using ER0 for the reverse ERN (i.e., the ERN for the SLU to PLU direction) stems from the fact that if the SSCP(SLU) does not support receiving SESSEND from the SLU or the BF for the SLU, it receives NS_LSA in order to clean up its state information for the session. The PU_T4|5 sends an NS_LSA only if ER0 fails.

- Includes the mode name and COS name in the above vector.

CDSESSST notifies the SSCP(SLU) that the LU-LU session identified by the Session Key Content field and the specified PCID for the initiation procedure has been successfully activated.

CDSESSSF notifies the SSCP(SLU) that the LU-LU session initiation identified by the Session Key Content field and the specified PCID for the initiation procedure has failed. The request contains the reason for the failure and associated sense data.

CDSESSEND notifies the SSCP that the LU-LU session identified by the Session Key Content field and the specified PCID for the termination procedure has been successfully deactivated.

CDSESSEND may flow from the SSCP(PLU) to the SSCP(SLU), from the SSCP(SLU) to the SSCP(PLU), or both. The SSCP.SVC_MGR.SS uses the following algorithm to send CDSESSEND and +RSP(CDSESSEND):

- CDSESSEND is sent, following the receipt of SESSEND, if and only if a CDSESSEND has not been received.

- +RSP(CDSESSEND) is sent:

  1. Immediately following the receipt of CDSESSEND:

     (a) if the receiver has already sent a CDSESSEND,
     (b) if the indicated LU-LU session is not known, or
     (c) if the CDSESSEND receiver does not have an active session with the LU in its domain.

  2. When a SESSEND is received after receipt of a CDSESSEND.

  3. When the SSCP-LU session is lost after receipt of a CDSESSEND.

CDSESSEND(Format 2) and RSP(CDSESSEND, Format 2) also specify the cause of the deactivation of the identified LU-LU session and indicate (via an Action field) if either the primary half-session or the secondary half-session will restart the session (via an INIT request). The cause and action values in CDSESSEND are equal to the respective values in the corresponding SESSEND. The cause and action values in RSP(CDSESSEND, Format 2) are determined as follows.

- If the RSP(CDSESSEND) is sent by the SSCP(SLU), it echoes back the contents of the Cause and Action fields from the CDSESSEND that it received, unless the PLU indicated normal action (no automatic restart), while the SLU indicated that the secondary half-session will restart; in this case, the SSCP(SLU) indicates that the secondary half-session will restart.

- If the RSP(CDSESSEND) is sent by the SSCP(PLU), and each side wants to restart, it overrides the contents of the CDSESSEND that it received from the SSCP(SLU) with the contents of the SESSEND that it received from the PLU.

CDSESSTF notifies the SSCP(SLU) that the LU-LU session termination identified by the Session Key Content field and the specified PCID for the termination procedure has failed. The request contains the reason for the failure and associated sense data.

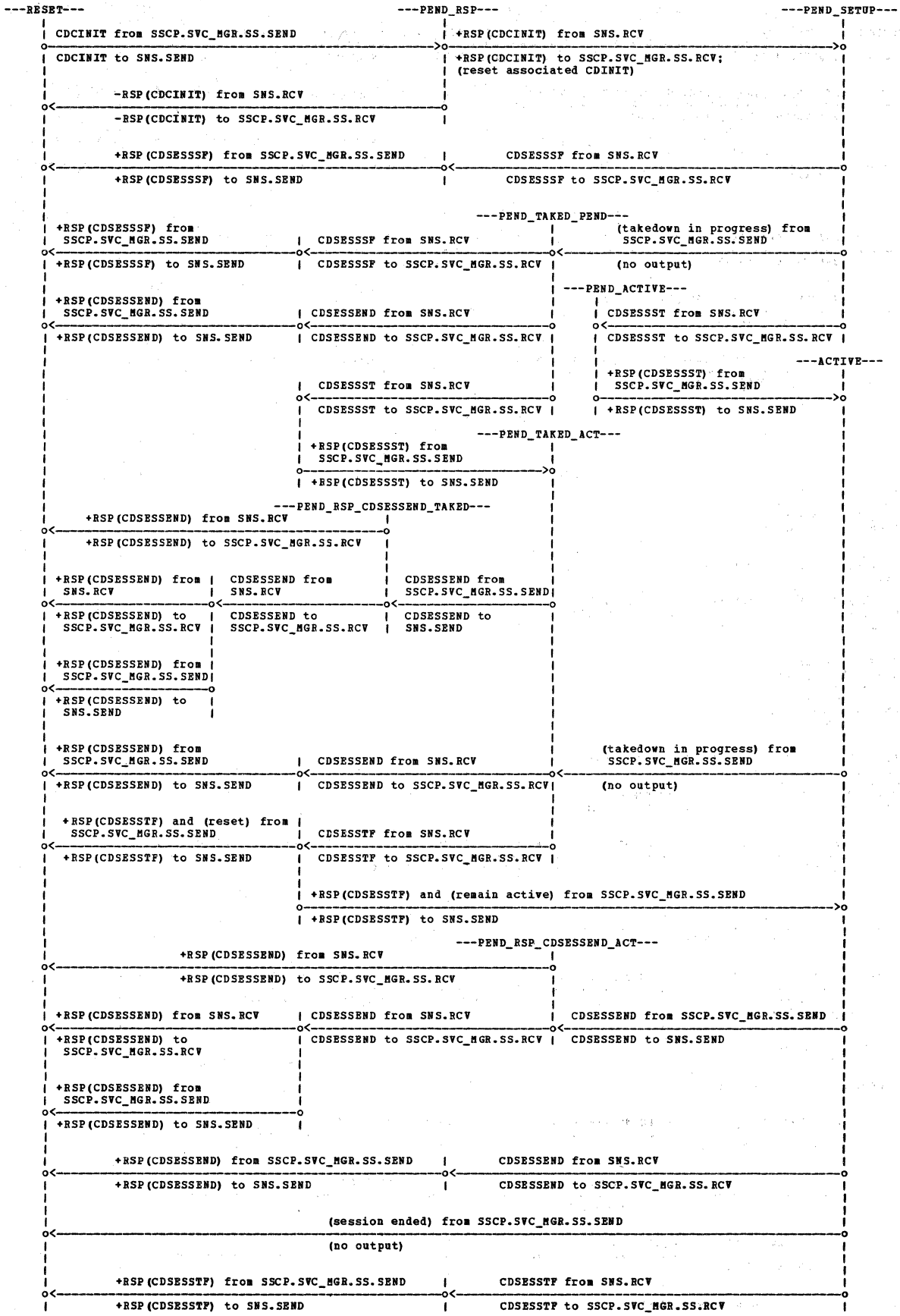
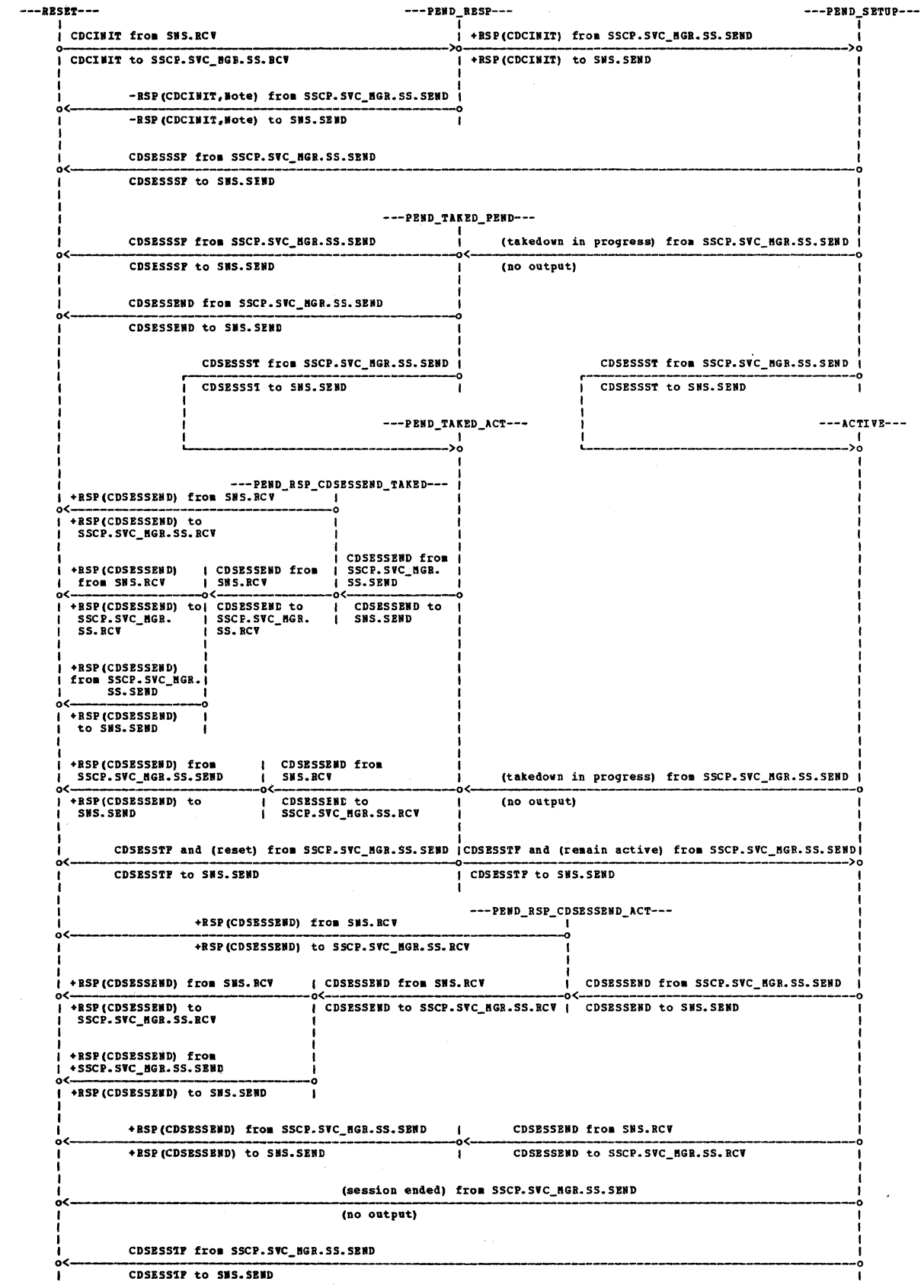


Figure 8-24. (SSCP(PLU),SSCP(SLU)).SSCP(SLU).CDCSESS(PLU,SLU,PCID)



Note: Sense Codes: 0812,0821,083B,1005

Figure 8-25. (SSCP(PLU),SSCP(SLU)).SSCP(PLU).CDCSESS(PLU,SLU,PCID)

CROSS-DOMAIN TERMINATE(CDTERM)

Flow: SSCP(OLU) to SSCP(DLU) (Normal)

Principal FSM:
(SSCP(DLU),SSCP(OLU)).SSCP(OLU|DLU).CDTERM(SESSION_KEY_CONTENT,PCID)_SEND-RCV (Page 8-62)

CDTERM from the SSCP(OLU) requests that the SSCP(DLU) assist in the termination of the cross-domain LU-LU session identified by the Session Key Content field and the Type byte of the RU. Each SSCP executes that portion of terminate processing that relates to the LU in its domain. The Type byte specifies whether the request applies to:

- Active and pending-active sessions
- Active, pending-active, and queued sessions
- Queued sessions, only

The Type byte specifies also if the termination is to be Forced, Orderly, or Cleanup. Forced, Orderly, and Cleanup terminations are described under TERM-SELF.

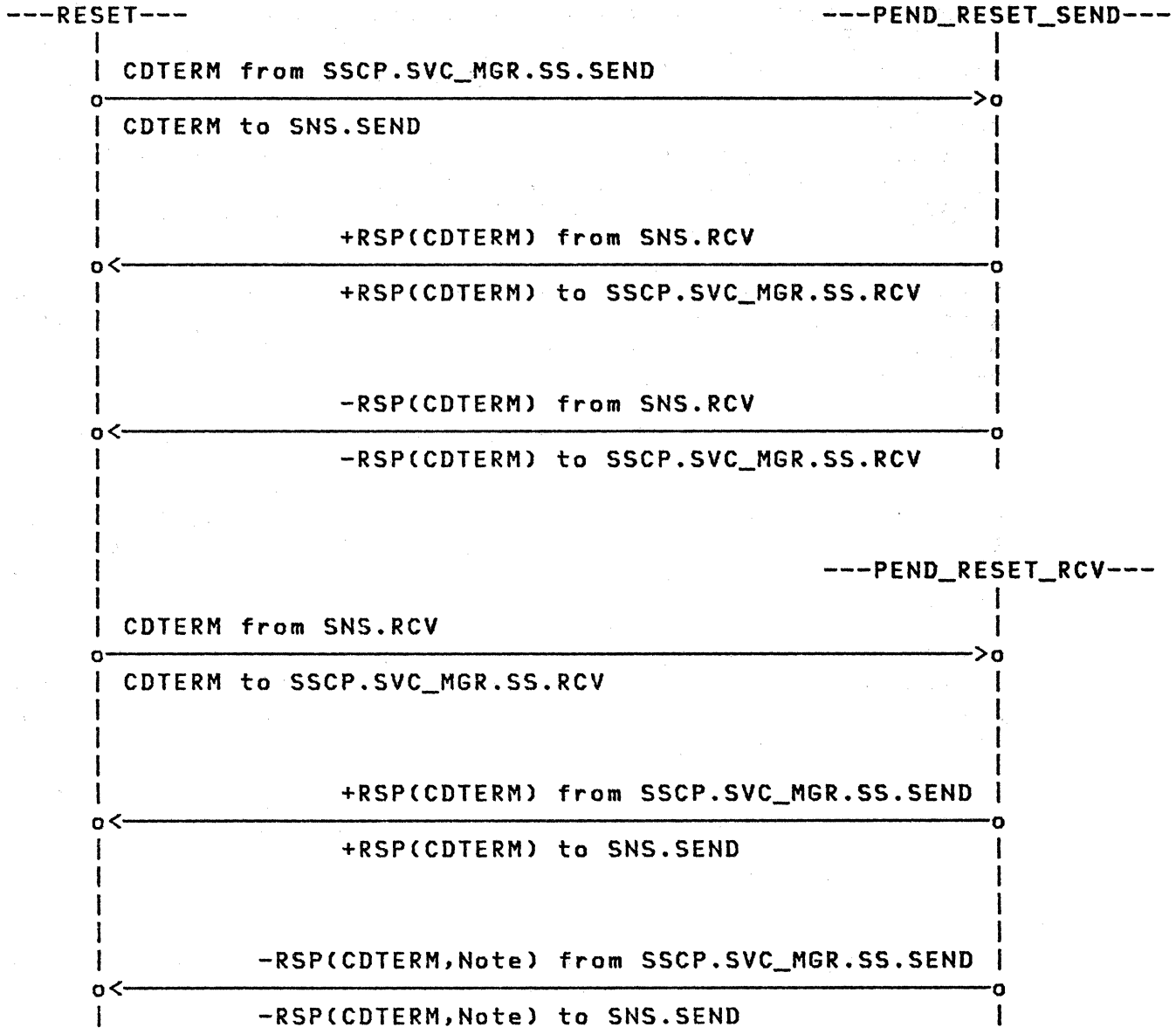
CDTERM identifies the session to be terminated via a network-name-pair, network-address-pair, or PCID session key. When the CDTERM is sent to terminate a session, prior to receipt of the CDINIT response carrying the assigned DLU network address, the CDTERM carries the PCID session key to identify the session to be terminated; the PCID is that of the CDINIT.

(SSCP(DLU),SSCP(OLU)).SSCP(DLU).CDTERM(SESSION_KEY_CONTENT, PCID).RCV receives the CDTERM request and, if it is valid, passes it to the SSCP(DLU).SVC_MGR.SS, which may perform the following processing (based upon Type and Reason fields defining the protocol requested):

- Establish the authority of the end user to request the termination of the specified session.
- Retain the PCID for later use within any appropriate NOTIFY, CDSESSEND, and CDSESSTF RUs.
- Determine the PLU and SLU for the session, based on information retained from the session initiation.
- Resolve the network name of the DLU into a network address to be sent in the response to CDTERM.
- If SSCP(DLU) = SSCP(PLU): Send a CTERM (Orderly|Forced|Cleanup), as specified in CDTERM.

- If SSCP(DLU) = SSCP(SLU) and the CDTERM specified Cleanup: Send a CLEANUP to the SLU (in a subarea node) or either DACTLU or ACTLU(Cold) to the SLU (in a peripheral node).

A positive response is returned once the CDTERM is accepted. The deactivation of the LU-LU session is completed sometime later.



Note: Sense Codes: 0803,0804,0806,080E,080F,0810,0836,083E

Figure 8-26. (SSCP(DLU),SSCP(OLU)).SSCP(OLU|DLU).CDTERM (SESSION_KEY_CONTENT,PCID)_SEND-RCV

TERMINATE-OTHER CROSS-DOMAIN (TERM-OTHER-CD)

Flow: SSCP(TLU) to SSCP(OLU) (Normal)

Principal FSMs:
(SSCP(OLU),SSCP(TLU)).SSCP(TLU).TERM-OTHER-CD(SESSION_KEY_CONTENT,PCID)_SEND (Page 8-64)
(SSCP(OLU),SSCP(TLU)).SSCP(OLU).TERM-OTHER-CD(SESSION_KEY_CONTENT,PCID)_RCV (Page 8-64)

TERM-OTHER-CD transports a TERM-OTHER request from the SSCP(TLU) where it was received, to the SSCP(OLU), which manages at least one of the (LU1,LU2) pair participating in the session(s) to be terminated.

TERM-OTHER-CD identifies the session to be terminated via a network-name-pair, network-address-pair, or PCID session key. When the TERM-OTHER-CD is sent to terminate a session, prior to receipt of the NOTIFY(Vector Key X'03') carrying the assigned network address pair, the TERM-OTHER-CD carries the PCID session key to identify the session to be terminated.

(SSCP(OLU),SSCP(TLU)).SSCP(OLU).TERM-OTHER-CD(SESSION_KEY_CONTENT,PCID)_RCV receives the TERM-OTHER-CD after it has been validated. The SSCP(OLU).SVC_MGR.SS may perform the following processing:

- Perform the same processing as described for TERM-SELF, except that which relates to the TLU and the SSCP(TLU).
- Resolve the network name of the OLU to a network address.
- Establish the authority of the end user to request the termination of the specified session.
- Retain the PCID for later use within any NOTIFY RU sent to the SSCP(TLU).
- Send a -RSP(TERM-OTHER-CD, X'0853'--Cleanup Required), if the TERM-OTHER-CD did not specify Cleanup and the SSCP-SSCP session with the SSCP having an active SSCP-LU session with the cross-domain LU is not active.

A positive response is returned once the TERM-OTHER-CD is accepted. The deactivation of the LU-LU session(s) is completed sometime later and the SSCP(TLU) is notified via NOTIFY(Vector Key X'03') from the SSCP(OLU).

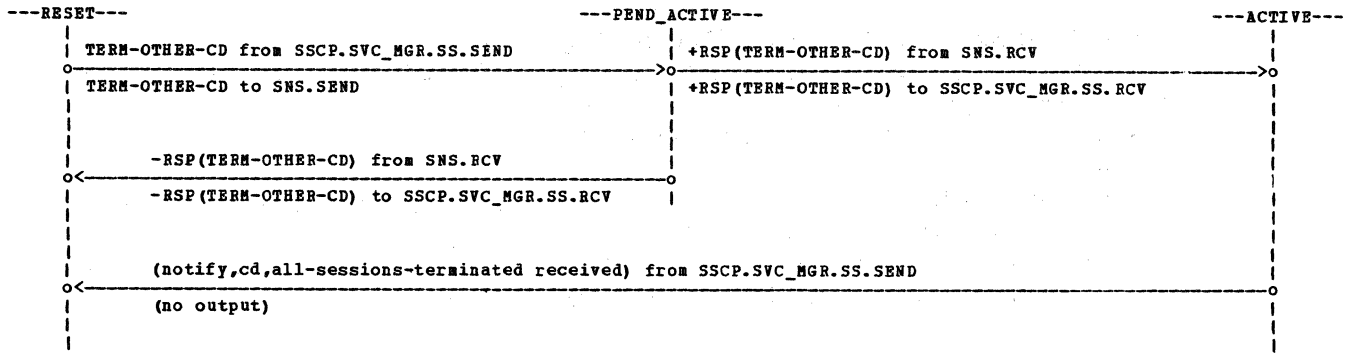
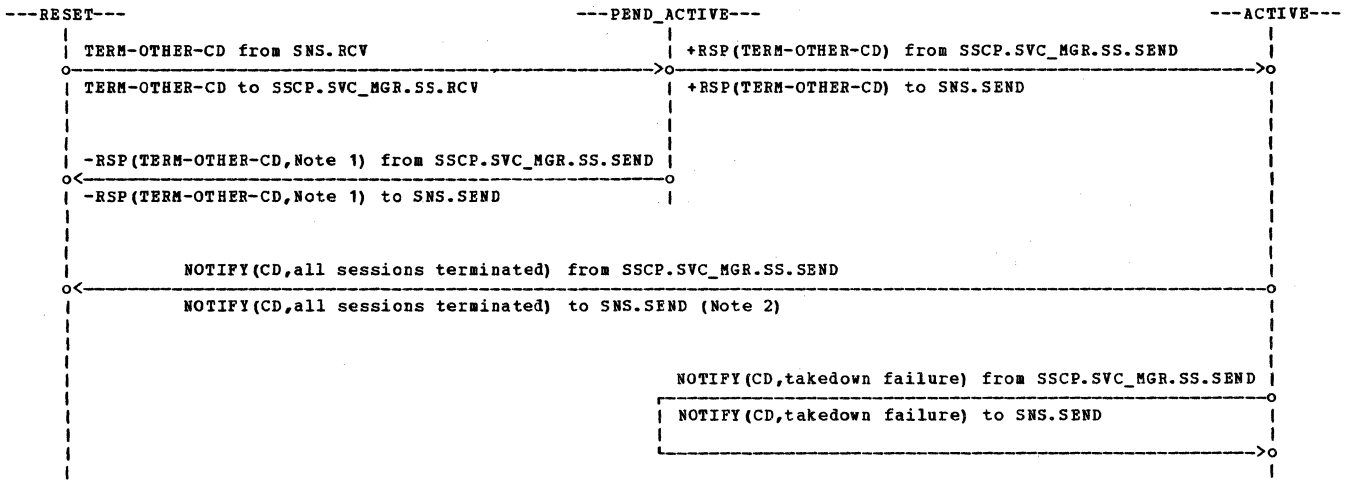


Figure 8-27. (SSCP(OLU),SSCP(TLU)).SSCP(TLU).TERM-OTHER-CD (SESSION_KEY_CONTENT,PCID)_SEND



Notes:

1. Sense Codes: 0803,0804,0806,0809,080E,080F,0810,0812,0816,081E,0839,083B,0842,0853

2. NOTIFY is sent when all requested sessions are terminated.

Figure 8-28. (SSCP(OLU),SSCP(TLU)).SSCP(OLU).TERM-OTHER-CD (SESSION_KEY_CONTENT,PCID)_RCV

CROSS-DOMAIN TAKEDOWN (CDTAKED)
CROSS-DOMAIN TAKEDOWN COMPLETE (CDTAKEDC)

Flow: From SSCP to SSCP (Normal)

Principal FSMs:
(SSCP,SSCP').SSCP.CDTAKED(Type,PCID)_SEND-RCV
(Page 8-68)
(SSCP,SSCP').SSCP.CDTAKED(CU)_SEND-RCV
(Page 8-69)

CDTAKED initiates a procedure to cause the takedown of all cross-domain LU-LU sessions (active, pending-active, and queued) involving the domains of both the sending and receiving SSCP. It also prevents the initiation of new LU-LU sessions between these domains; i.e., neither SSCP is allowed to send CDINIT to the other. In the case of contention (not involving the cleanup option) for domain takedown, the primary SSCP responds negatively to the CDTAKED request from the secondary SSCP, and continues its processing of the CDTAKED it sent; the secondary SSCP processes the received CDTAKED. Each session termination is reported individually via CDSESSEND or CDSESSTF for Quiesce, Orderly, or Forced takedown procedure. Takedown using cleanup is mutual, both SSCPs participating, but no CDSESSEND or CDSESSTF is sent.

The CDTAKED Type byte coding and the resulting SSCP procedures are as follows:

- Quiesce with queued-only: sessions end normally; queues are purged.
- Quiesce with active and pending-active: sessions end normally; queues go on hold (no queued sessions may be started).
- Quiesce with active, pending-active, and queued: sessions end normally; queues are purged.
- Orderly with queued-only: same as quiesce with queued-only.
- Orderly with active and pending-active: sessions end by CTERM(Orderly) being sent; queues go on hold.
- Orderly with active, pending-active, and queued: sessions end by CTERM (Orderly) being sent; queues are purged.
- Forced with queued-only: same as quiesce with queued-only.

- Forced with active and pending-active: sessions end by CTERM(Forced) being sent; queues go on hold.

- Forced with active, pending-active, and queued: sessions end by CTERM(Forced) being sent; queues are purged.

- Cleanup with queued-only: same as quiesce with queued-only.

- Cleanup with active and pending-active: sessions end when CTERM(Cleanup), CLEANUP, or DACTLU followed by ACTLU are sent to the applicable LUs; queues go on hold.

- Cleanup with active, pending-active, and queued: sessions end when CTERM(Cleanup), CLEANUP, or DACTLU followed by ACTLU are sent to the applicable LUs; queues are purged.

Except when the Cleanup option was specified, the SSCP that received CDTAKED (and positively responded to it) sends CDTAKEDC upon completion of its domain takedown procedure. The other SSCP, after completing its domain takedown procedure and receiving a CDTAKEDC, also sends a CDTAKEDC. If multiple CDTAKED's are sent, CDTAKEDC will contain the PCID of the highest level CDTAKED.

In processing the different types of CDTAKED, the following precedence rules apply for major levels of precedence (Quiesce, Orderly, Forced, Cleanup):

- Orderly takes precedence over Quiesce.

- Forced takes precedence over Orderly and Quiesce.

- Cleanup takes precedence over Forced, Orderly and Quiesce.

A CDTAKED may affect just the queued sessions, just the active/pending-active sessions, or both. If multiple CDTAKEDs are sent, a takedown procedure can progress with queued sessions terminating at a different major level (quiesce, orderly, forced, cleanup) than active/pending-active sessions. Also, queued sessions may be terminating at the same major level as active/pending-active sessions but as a result of two separate CDTAKEDs. CDTAKEDC is sent after both termination functions are complete.

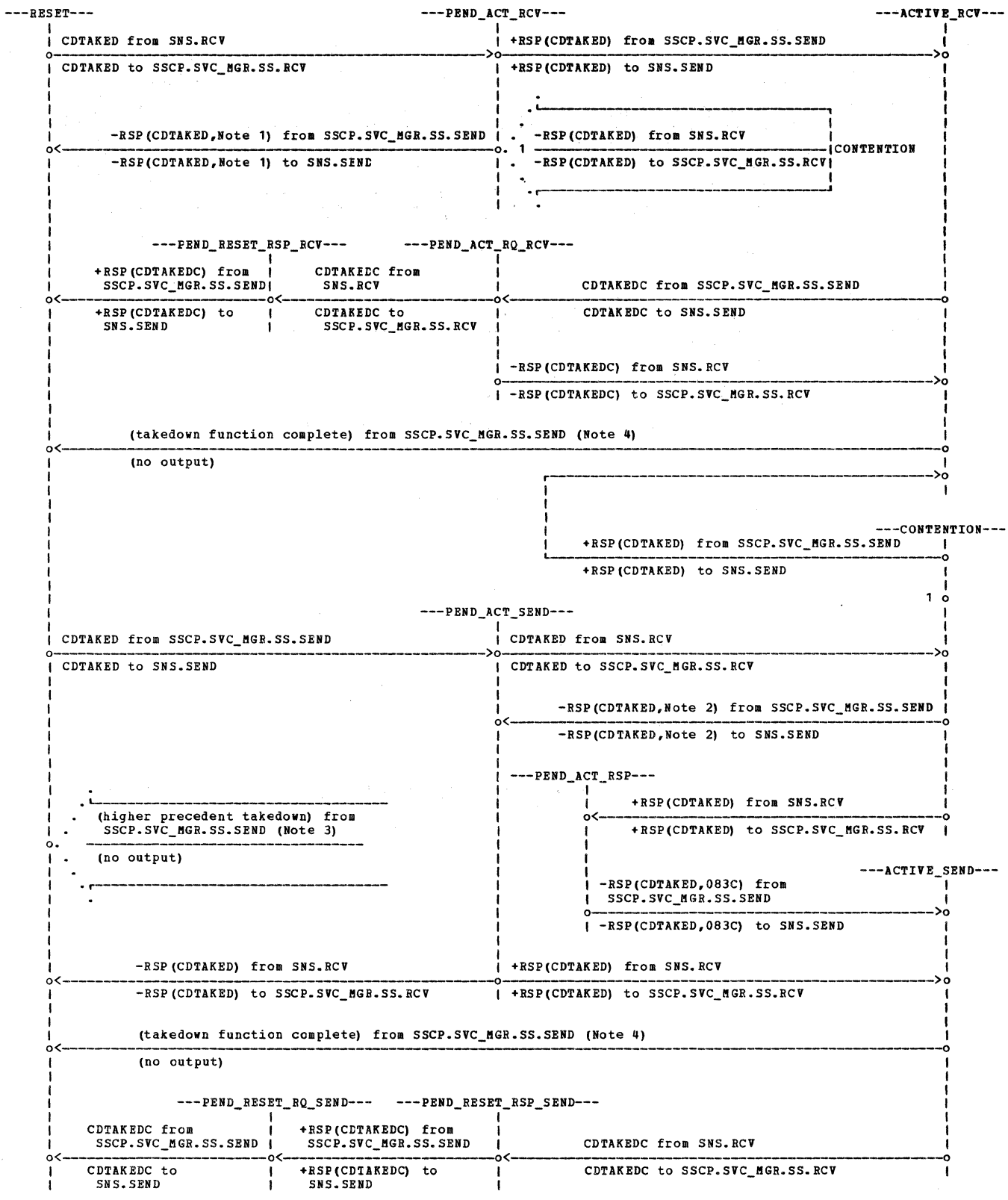
To determine which CDTAKED PCID to use in CDTAKEDC, three minor levels of precedence are defined—queued, active/pending-active (A/PA), and both queued and active/pending-active (queued-and-A/PA)—within each major level. For minor levels of precedence:

- A/PA takes precedence over queued.
- Queued-and-A/PA takes precedence over A/PA and over queued.

The following order of precedence combines major and minor levels of precedence; each level in the list takes precedence over all levels preceding it in the list:

Major Level	Minor Level
1. Quiesce	Queued
2. Quiesce	A/PA
3. Quiesce	Queued-and-A/PA
4. Orderly	Queued
5. Orderly	A/PA
6. Orderly	Queued-and-A/PA
7. Forced	Queued
8. Forced	A/PA
9. Forced	Queued-and-A/PA
10. Cleanup	Queued
11. Cleanup	A/PA
12. Cleanup	Queued-and-A/PA

Contention between CDTAKEDs (received CDTAKED before receiving response to previous CDTAKED) occurs only if both major and minor levels of precedence are the same.



Notes:

1. Sense codes: 080A,1003
2. Sense codes: 080A,083C,1003
3. Higher precedent takedown supersedes this FSM and the takedown function.
4. Higher precedent takedown occurred, and did not supersede this takedown function.

Figure 8-29. (SSCP,SSCP').SSCP.CDTAKED(Type,PCID)_SEND-RCV

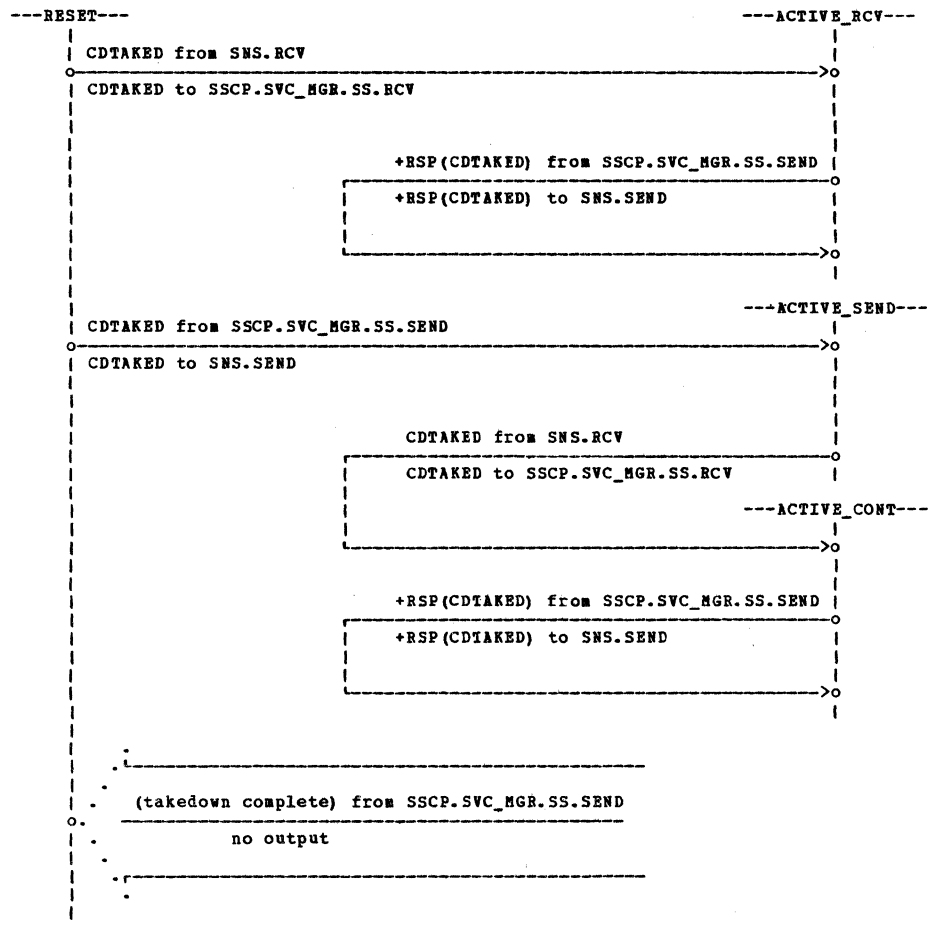


Figure 8-30. (SSCP,SSCP').SSCP.CDTAKED(CU)_SEND-RCV

DIRECT SEARCH LIST (DSRLST)

Flow: From SSCP to SSCP (Normal)

Principal FSMs:
(SSCP,SSCP').SSCP.DSRLST_SEND (Page 8-71)
(SSCP,SSCP').SSCP'.DSRLST_RCV (Page 8-71)

DSRLST identifies a control list type and specifies a list search argument to be used at the receiving SSCP. The receiving SSCP searches the control list accordingly, and returns the appropriate control list entry data in RSP(DSRLST).

The Control List Type field in DSRLST specifies an LU status control list; the Control List Search Argument field contains the network name of the LU in question. The control list entry returned in RSP(DSRLST) provides status information related to the LU's availability for LU-LU session initiation and to the LU's location (whether it resides in a PU_T5 node).

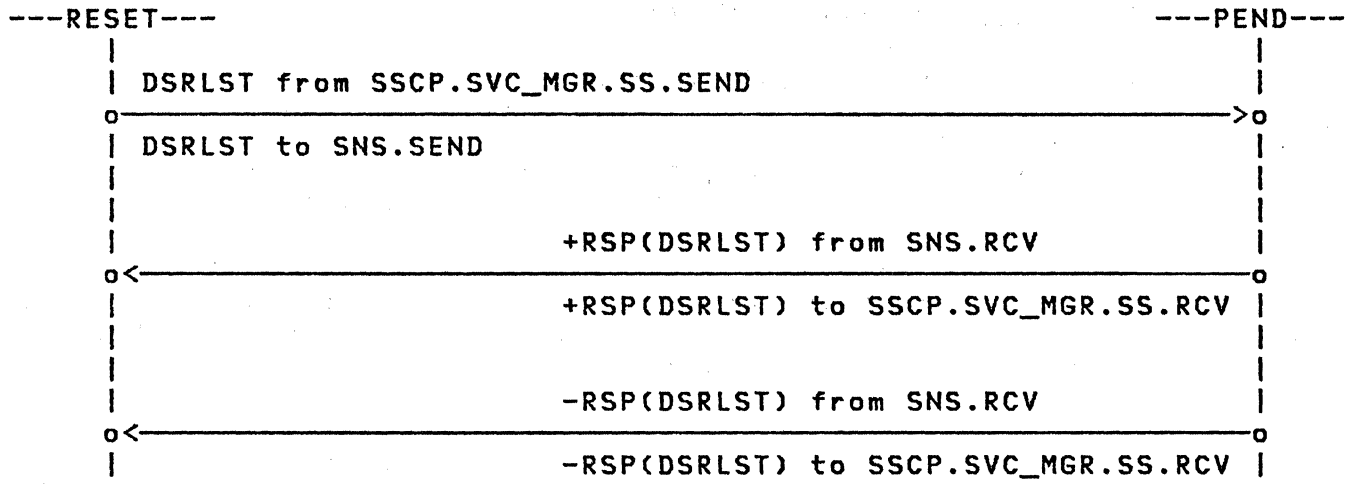
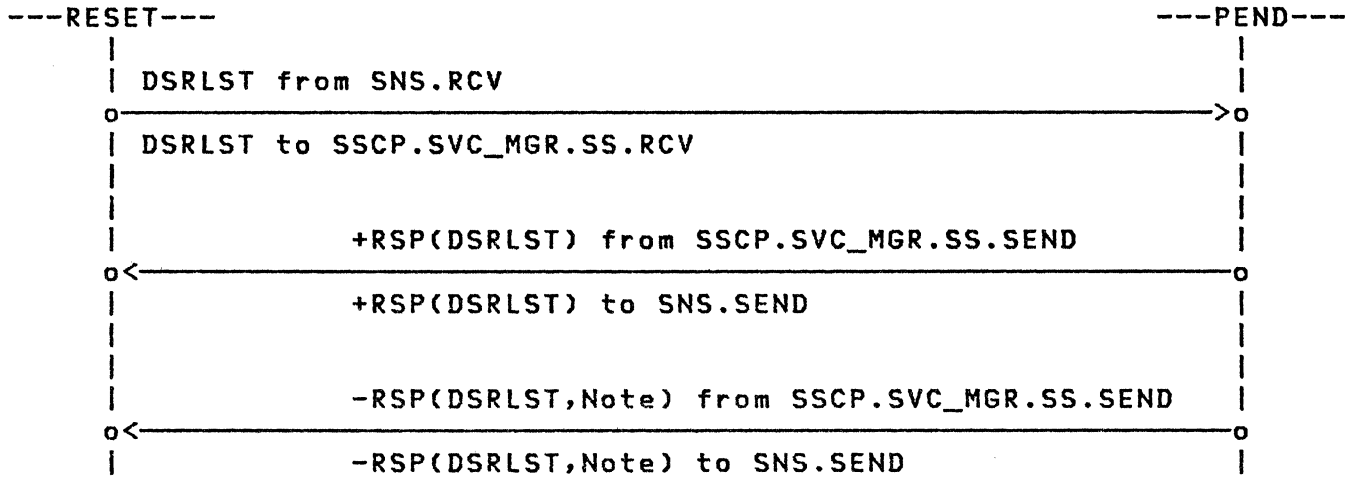


Figure 8-31. (SSCP,SSCP').SSCP.DSRLST_SEND



Note: Sense codes: 0820,0823

Figure 8-32. (SSCP,SSCP').SSCP'.DSRLST_RCV

```
FSM_INPUT_DEFINITION:

   BINDF       NS_RQ_CODE = BINDF;
   CINIT       NS_RQ_CODE = CINIT;
   CLEANUP     NS_RQ_CODE = CTERM & CTERM_RQ.TYPE = CLEANUP;
   CTERM       NS_RQ_CODE = CTERM;
   FORCED      NS_RQ_CODE = CTERM & CTERM_RQ.TYPE = FORCED;
   LAST        IF UPM_RSP_TO_LAST_RQ = OK; /* TO DETERMINE WHETHER A RESPONSE IS THE        */
                                           /* RESPONSE TO THE LAST REQUEST SENT, THE ID OF  */
                                           /* THE LAST REQUEST SENT IS STORED AND COMPARED  */
                                           /* WITH THE ID ON THE RESPONSE.                  */
   ORDERLY     NS_RQ_CODE = CTERM & CTERM_RQ.TYPE = ORDERLY;
   R           MUCB.DIRECTION = RECEIVE;
  'RESET'      INPUT('RESET');
  ±RSP         RRI = RSP;
  +RSP         RRI = RSP & RTI=POS;
  -RSP         RRI = RSP & RTI = NEG;
   RQ          RRI = RQ;
   S           MUCB.DIRECTION = SEND;
   SESSEND     NS_RQ_CODE = SESSEND;
   SESSST      NS_RQ_CODE = SESSST;
   UNBINDF     NS_RQ_CODE = UNBINDF;

END FSM_INPUT_DEFINITION;
```

CHAPTER 9. MANAGEMENT AND MAINTENANCE SERVICES

Every SNA node contains an SSCP (in PU_T5 nodes) or a PUCP (in PU_T1|2|4 nodes), a PU, and (optionally) one or more LUs. These are collectively called NAUs. Every NAU, in turn, contains a NAU services layer, designated SSCP.SVC, PUCP.SVC, PU.SVC, and LU.SVC, respectively. Distributed among the NAU services layers within a network are service and control components. These components control the network operation by exchanging RUs with one another. Additional information about the NAU services layer is contained in Chapters 1 and 6, and illustrations depicting the structure of the NAU services within a node are contained in Chapter 6.

Distributed among each SSCP.SVC, PU.SVC, and LU.SVC are maintenance services, which coordinate the testing of various network resources and the reporting of the status of network resources. This coordination is accomplished by exchanging maintenance services RUs on SSCP-LU and SSCP-PU sessions. Maintenance services requests are used to support link level traces, the testing by a PU of network resources (such as the PU, or LUs and links supported by the PU), and the reporting by LUs and PUs of the status of network resources and the results of test requests.

Distributed among each SSCP.SVC and LU.SVC are management services, which support communications network management applications. (Note that there are no management services in PU.SVC.) Management services allow the communications network management application to use the existing LU-SSCP session and, indirectly, the SSCP-PU sessions to access the communications network management services component associated with a specific node, as described in the section "Communication Network Management." This CNM component access is accomplished using network names; the SSCP translates network names to network addresses.

As shown in Figures 9-1 and 9-2, the management and maintenance services for each SSCP.SVC and LU.SVC consists of a services manager component, and one or more half-session components (one per half-session). The services manager component for the SSCP.SVC is designated SSCP.SVC_MGR.(MN&MA); for the LU.SVC, it is designated LU.SVC_MGR.(MN&MA). The half-session components for both the SSCP.SVC and LU.SVC are designated SNS.(MN&MA). The SSCP.SVC_MGR.(MN&MA), LU.SVC_MGR.(MN&MA), and SNS.(MN&MA) components are each made up of two main subcomponents: a send subcomponent (*.(MN&MA).SEND) and a receive subcomponent (*.(MN&MA).RCV).

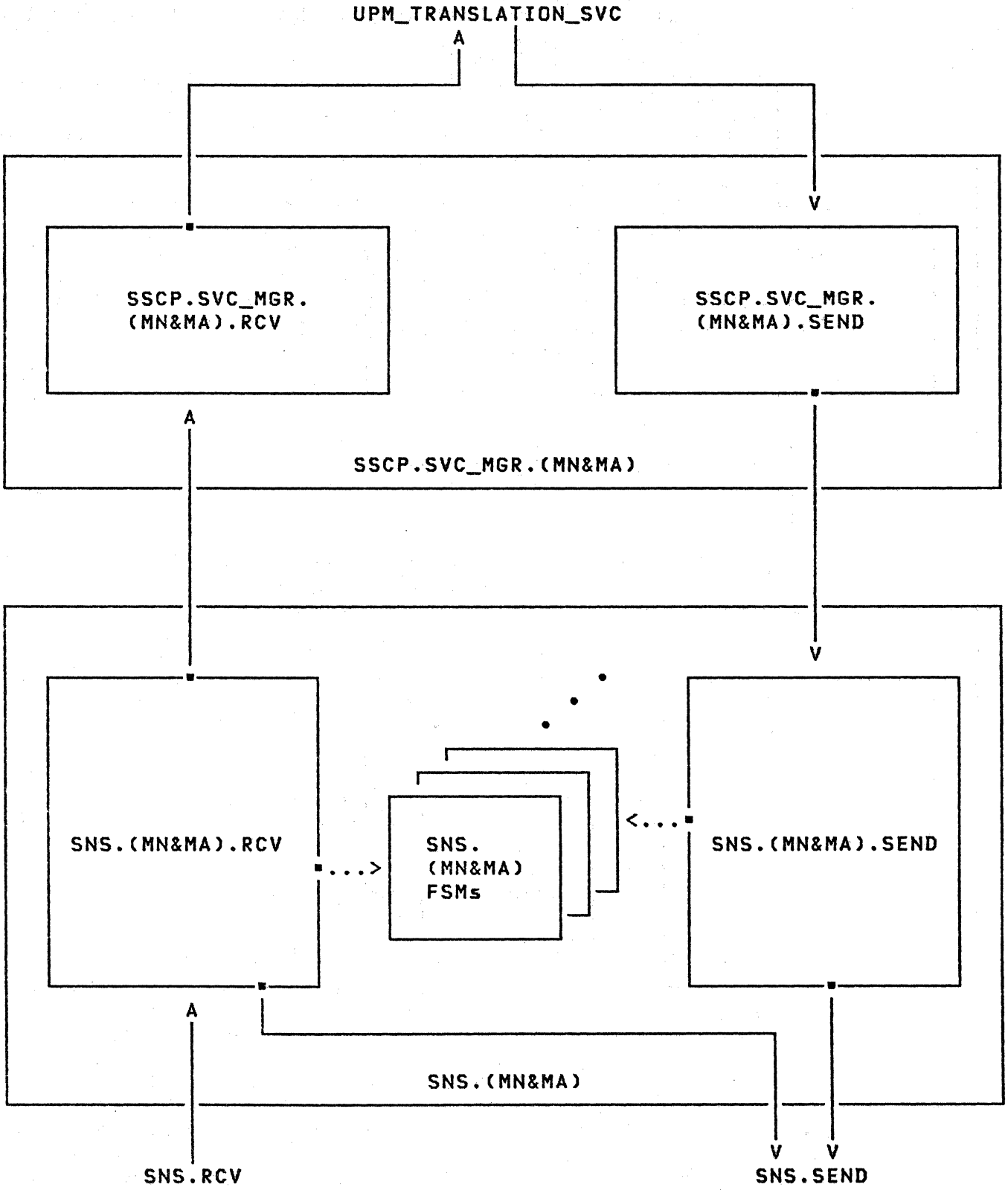


Figure 9-1. Structure of SSCP Management and Maintenance Services

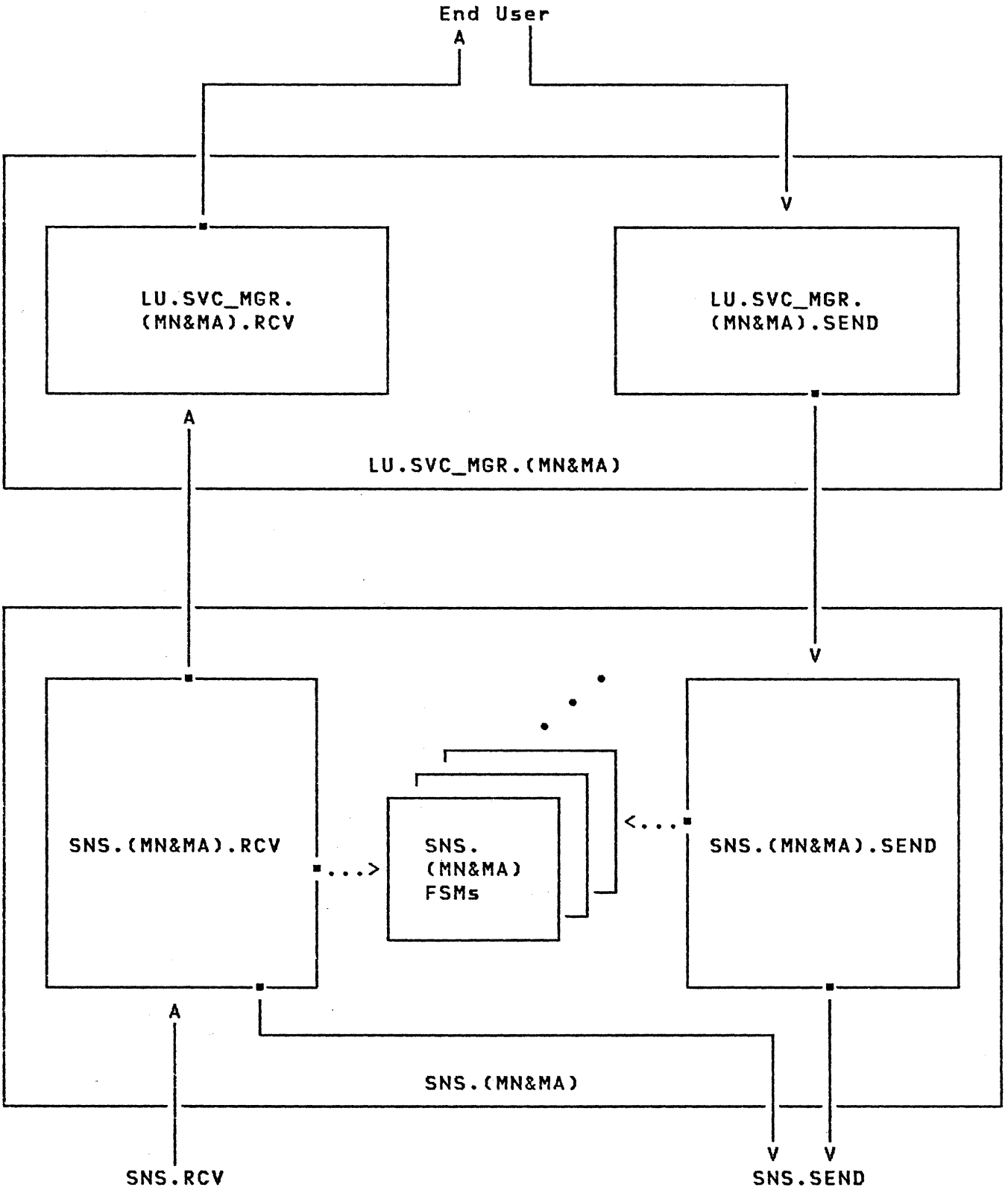


Figure 9-2. Structure of LU Management and Maintenance Services

COMMUNICATION NETWORK MANAGEMENT

INTRODUCTION

Communication network management (CNM) consists of two types of components: a CNM application (CNMA) component, providing CNM functions such as problem determination for a collection of network resources in the domain of the SSCP to which the CNMA is connected; and a CNM services (CNMS) component associated with each PU, providing CNM functions such as threshold monitoring and statistics gathering. The CNMA components may communicate with each other using (cross-domain) LU-LU sessions. The CNMA and CNMS components communicate with each other via the SSCP using LU-SSCP and SSCP-PU sessions. Figure 9-3 shows the communication network management connection alternatives. Only problem determination related CNM requests are defined in this book.

The relationship of the CNMA to the CNMS is illustrated in Figure 9-3. The CNMA is coupled to the SSCP via an LU-SSCP session. Management category RUs are used to send CNMA requests from the CNMA to the SSCP and to send CNMS requests from the SSCP to the CNMA. The management category RUs allow the CNMAs and CNMSs to communicate with each other using the existing SSCP-PU session.

The CNMS is associated with a PU (see Chapter 11), which uses maintenance services to send or receive CNM requests to or from the SSCP.

CNM request flows are illustrated in Figure 9-4. CNMA to CNMS flow requests are called simply CNM requests, while the CNMS to CNMA flow contains both solicited and unsolicited CNM requests. The solicited CNM requests are called CNM replies.

The network name of the PU with which a CNMS is associated is the destination name for requests from CNMA to CNMS and is the origin name for requests from CNMS to CNMA. The network name of the resource (PU, LU, link, or adjacent link station) controlled and monitored by the CNMS is the target name. The type of resource that may be used as a target in a specific CNM request is specified in the definition of that request.

The two management services RUs, FORWARD and DELIVER, include fields that contain the CNM request and the target name, as well as either the destination name (in FORWARD) or the origin name (in DELIVER). The CNM request field is referred to as the embedded request. See the descriptions of FORWARD and DELIVER for the RUs that may be embedded. Some embedded requests have a CNM header, in which case it is partially initialized, and the SSCP completes the

initialization by filling in the CNM target ID field. The embedded request appears as a maintenance services request on the SSCP-PU session.

The CNM header contains a target address and information that denotes whether a request from CNMS to CNMA has been solicited by a prior CNMA to CNMS request, or is an unsolicited request. It also contains a parameter, the procedure-related identifier (PRID), to allow CNMS to CNMA reply requests to be correlated to a prior CNMA to CNMS request. When the CNMS sends a reply to a CNMA request, the CNMS echoes the PRID field of the associated prior request. There may be multiple reply requests associated with a single CNM request from the CNMA, and the CNMS echoes the same PRID field in each reply; multiple replies to the same request are in a single series, i.e., all but the last reply have the Not Last Request indicator set.

An SSCP may use the PRID field for SSCP routing. PRIDs generated by an SSCP provide the SSCP with a means to interleave requests from multiple SSCP-PU|LU sessions onto a single SSCP-PU|LU session. The SSCP PRID-based routing procedure is the following: prior to sending the embedded NS RU to the PU|LU destination, the SSCP saves a copy of the CNMA-generated PRID along with the network address of the LU associated with the sending CNMA, and overlays the PRID field with an SSCP-generated PRID. As mentioned earlier, the PRID is echoed in the reply request from the CNMS. The echoed PRID allows the SSCP to determine which CNMA is to receive the reply request. Following this determination, the SSCP restores the saved CNMA-generated PRID into the reply request, embeds the reply request into a DELIVER request, and routes it to the LU associated with the CNMA.

SSCPs not doing PRID-based routing pass on, but do not use, the PRID in requests and reply requests.

The SSCP transforms a management services request received from an LU (CMNA) into a maintenance services request to be sent to a PU (CNMS). The PU and its associated CNMS do not know their own PU network name or the network names of the LUs, links, or adjacent link stations that they control and monitor; therefore the SSCP translates the network name for the target used in the management services requests into the proper form of target address to be used with the maintenance services requests sent to the CNMS; and selects the appropriate SSCP-PU session based upon the destination name. This translation is performed without the need to determine the specific CNM request being sent.

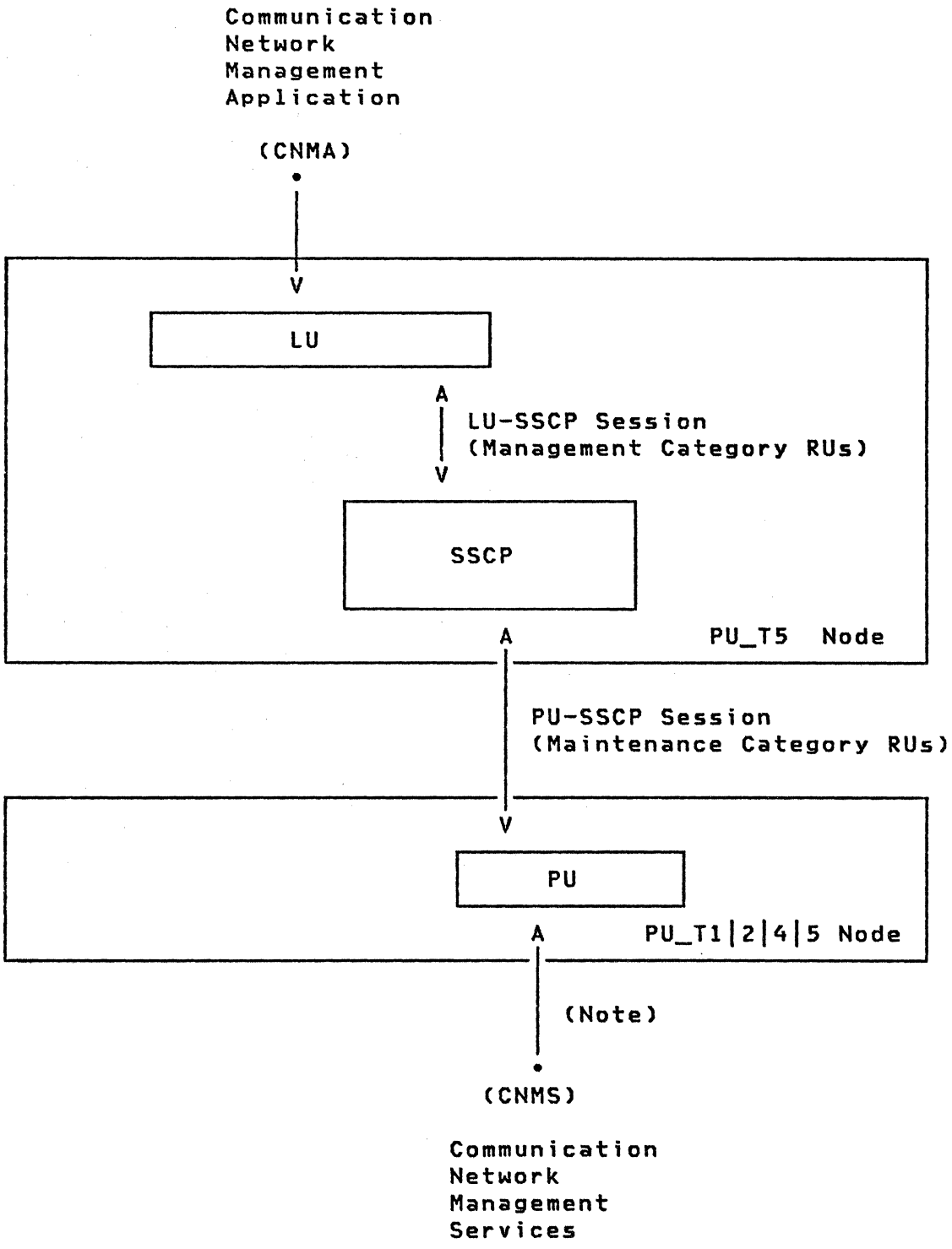
On request flows from a PU to an SSCP, the SSCP determines which maintenance services requests are to be processed by the SSCP and which are to be sent to the CNMA. The latter are embedded in a DELIVER RU. The DELIVER RU contains the

network name for both origin and target, and the configuration hierarchy information (see appendix E for details) associated with the target. An SSCP routes CNM requests by using an SSCP request routing table (implementation- and installation-defined) and the PRID parameter. Embedded requests contained within a DELIVER request on the flow to an LU are processed by CNMA

The processing of CNM maintenance services RUs is done by the CNMS. The PU.SVC_MGR.NS provides routing to or from the CNMS for those RUs.

The amount of data that can be transferred between the CNMA and CNMS is limited by the request-reply protocol used on the SSCP-PU session. In addition, each RU is limited to a total length of 256 bytes. These sessions (LU-SSCP, SSCP-PU) are not used for moving large amounts of data, e.g., files. When this is required, LU-LU sessions are used.

Responses to CNMA-originated management services requests are returned to CNMA when the embedded request within the management services request is delivered (i.e., the response is received at the SSCP) to the destination CNMS. Unlike management services responses to the CNMA, responses to CNMS originated maintenance services requests are returned to CNMS when the request is delivered to the SSCP. The responses to management services requests need not be received by CNMA in the same order as the corresponding requests were sent. The CNMA-related SSCP-LU session uses FM profile 6 and TS profile 1. The CNMS-related SSCP-PU session uses existing profiles for SSCP-PU sessions.



Note: The protocol boundary between the PU and CNMS is defined in Chapter 11.

Figure 9-3. CNM Connection Alternatives

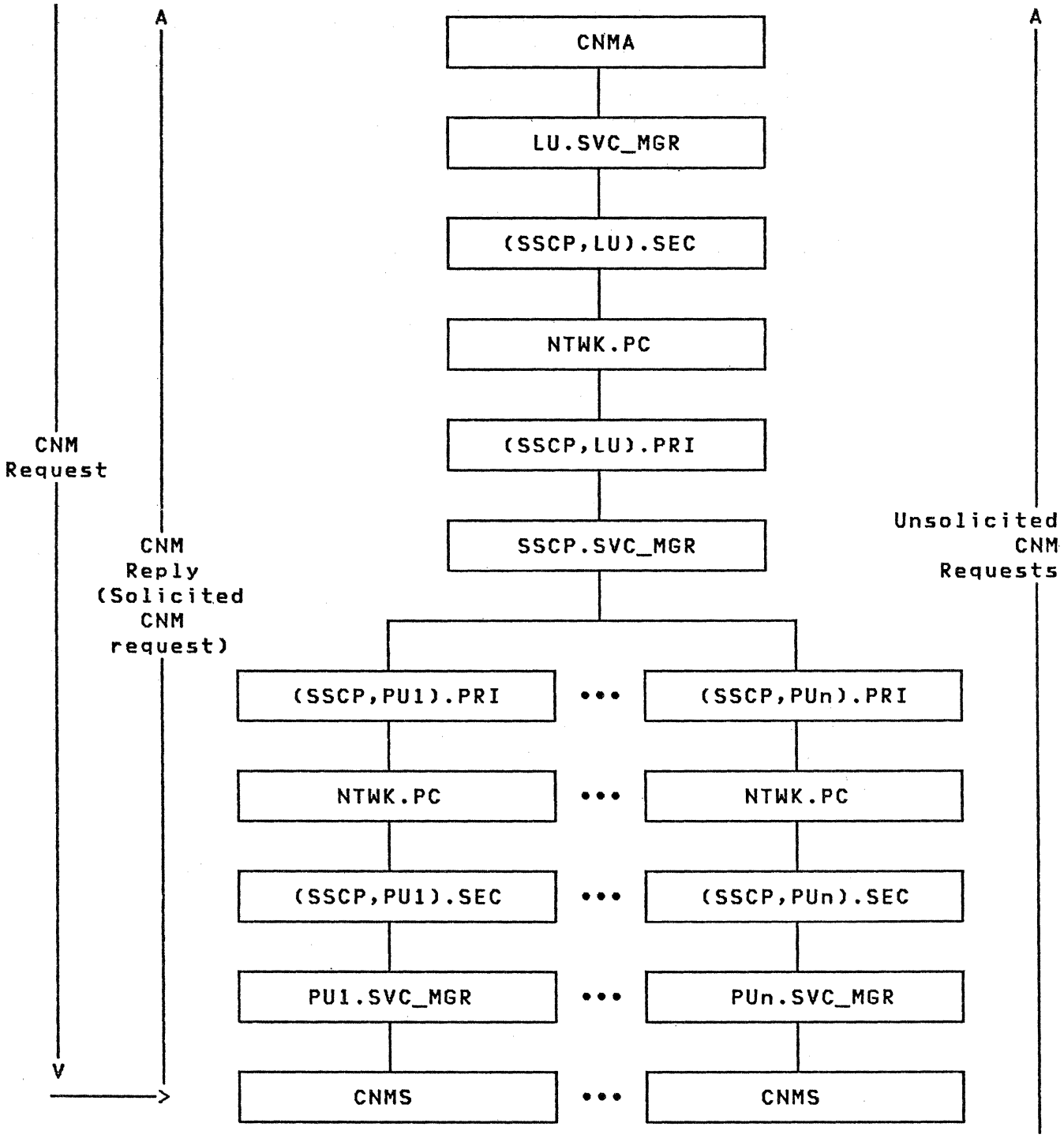


Figure 9-4. CNM Flows

COMMUNICATION NETWORK MANAGEMENT HEADER

The communication network management header is a five-byte header used in certain NS requests that can be embedded in FORWARD or DELIVER requests. When this header is used, it immediately follows the NS header in the embedded maintenance services RU. The CNM header formats are shown below.

- CNM header for requests sent by the SSCP:

Bytes	Description
0-4	CNM Header
0-1	CNM target ID, as specified in bytes 2-3, bits 2-3 Note: The target is the resource to which the requested statistics or other information pertain
2-3	bits 0-1, reserved bits 2-3, CNM target ID descriptor: 00 byte 1 contains a local address for a PU or LU in a PU_T2 node or an LSID for a PU or LU in a PU_T1 node; byte 0 is reserved 01 bytes 0-1 contain a network address identifying a link, adjacent link station, PU, or LU in the destination subarea bits 4-15, procedure related identifier (PRID): a CNM application program generated value for CNM application program correlation, or an SSCP generated value for SSCP routing
4	Request-Specific Information bit 0, request-specific indicator bit 1, reserved bits 2-7, request-specific type: specifies different functions and data presentations for the general function specified by the NS request code.

Note: For reply (i.e., solicited) requests, bytes 0-3 and byte 4, bits 2-7, echo the corresponding fields in the CNM header received in the request that solicited the reply request(s).

- CNM header for reply requests and unsolicited requests sent to the SSCP:

Bytes	Description
0-4	CNM Header
0-1	CNM target ID, as specified in bytes 2-3, bits 2-3 Note: The target is the resource to which the sent statistics or other information pertain
2-3	bits 0-1, reserved bits 2-3, CNM target ID descriptor: 00 byte 1 contains a local address for a PU or LU in a PU_T2 node or an LSID for a PU or LU in a PU_T1 node; byte 0 is reserved 01 bytes 0-1 contain a network address identifying a link, adjacent link station, PU, or LU in the origin subarea bits 4-15, procedure related identifier (PRID): a CNM application program generated value for CNM application program correlation, or an SSCP generated value for SSCP routing
4	Request-Specific Information bit 0, solicitation indicator: 0 unsolicited request 1 reply request bit 1, not last request indicator: 0 last request in a series of related unsolicited or reply requests, e.g., last reply request in a series corresponding to a single soliciting request 1 not last request bits 2-7, request-specific type: specifies different functions and data presentations for the general function specified by the NS request code.

Note: For reply (i.e., solicited) requests, bytes 0-3 and byte 4, bits 2-7, echo the corresponding fields in the CNM header received in the request that solicited the reply request(s).

For unsolicited requests, these fields--the CNM target ID descriptor, the CNM target ID, the PRID, and the request-specific information--are generated by the request sender. For unsolicited requests, the PRID field contains X'000'.

MAINTENANCE SERVICES RUS

ACTIVATE TRACE (ACTTRACE)
DEACTIVATE TRACE (DACTTRACE)

Flow: From SSCP to PU_T4|5 (Normal)

Principal FSMs:
(SSCP,PU).PRI.TRACE(na,n)_SEND (Figure 9-5)

ACTTRACE requests the PU to activate the specified type of resource trace (n) related to the specified network address (na).

DACTTRACE requests that the specified trace be deactivated.

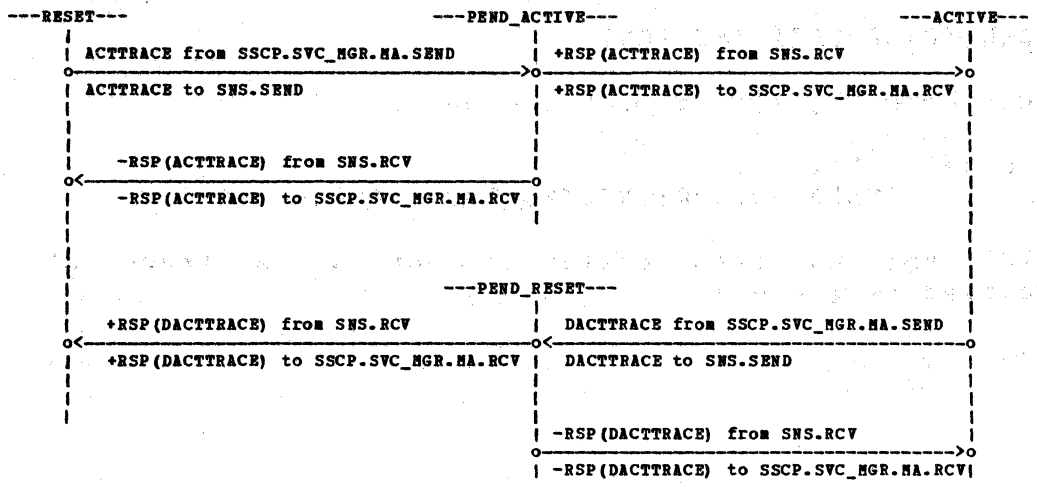


Figure 9-5. (SSCP,PU).PRI.TRACE(na,n)_SEND

RECORD TRACE DATA (RECTRD)

Flow: From PU_T4|5 to SSCP (Normal)

Principal FSMs:
(SSCP,PU).PRI.RECTRD_RCV (Figure 9-6)

RECTRD returns data collected during a trace of the specified resource.

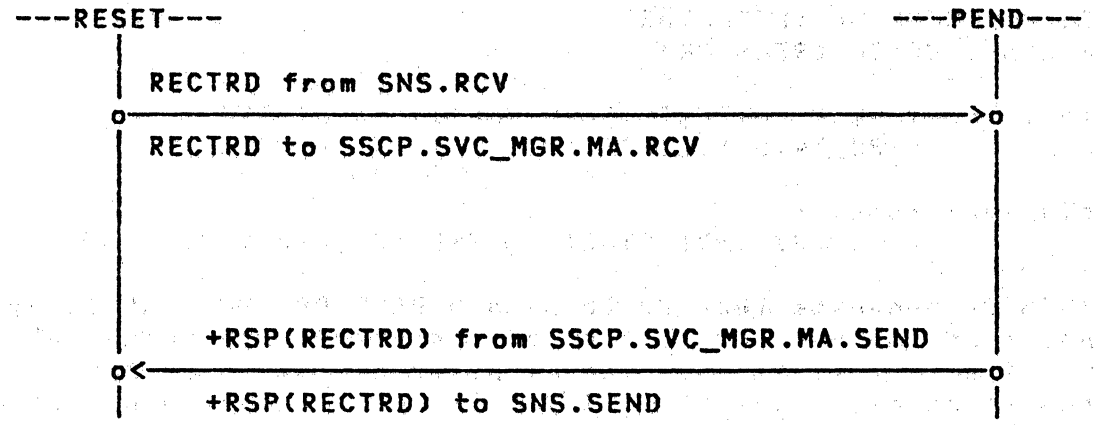


Figure 9-6. (SSCP,PU).PRI.RECTRD_RCV

DISPLAY STORAGE (DISPSTOR)
RECORD STORAGE (RECSTOR)

Flow: From SSCP to PU_T4|5 (Normal) for DISPSTOR
From PU_T4|5 to SSCP (Normal) for RECSTOR

Principal FSMs:
(SSCP,PU).PRI.STORAGE_SEND (Figure 9-7)

DISPSTOR requests the PU to send a RECSTOR RU containing a specified number of bytes of storage beginning at a specified location. If the Type byte specifies nonstatic program storage, the RECSTOR RU is constructed in real time and the PU may be changing the storage contents while the display bytes are being set up in the RU. If the Type byte specifies static snapshot storage, the RECSTOR RU is built with the assurance that storage contents are not being changed while the RECSTOR RU is being prepared.

RECSTOR carries the storage dump as requested in the above command.

DISPSTOR may be embedded in a FORWARD request; RECSTOR may be embedded in a DELIVER request. See the descriptions of FORWARD and DELIVER for details.

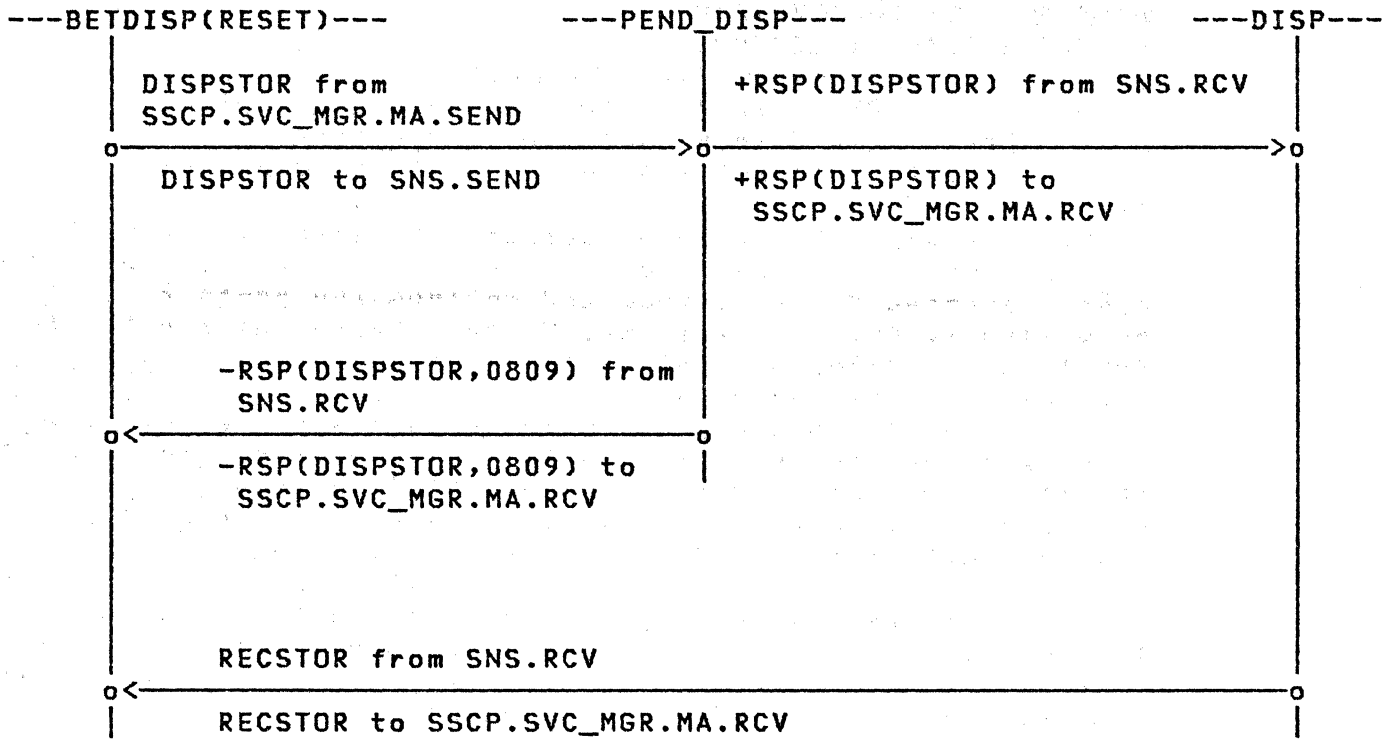


Figure 9-7. (SSCP,PU).PRI.STORAGE_SEND

EXECUTE TEST (EXECTEST)

Flow: From SSCP to PU_T4|5 (Normal)

Principal FSMs:
(SSCP,PU).PRI.TEST(na,n)_SEND (Figure 9-8)

EXECTEST requests the PU to activate the specified test type (n) related to the specified network address (na). The test code specifies the test type and defines the contents of the test data field. The test may be for the PU, or for the LUs or links supported by the PU.

A Link-Level 0 test or Link-Level 1 test can be specified. These are identical to the test described for Link-Level 2 (see TESTMODE for details), except for the level of dedication of resources while the test is being performed. Link-Level 0 requires a dedicated PU_T4|5 node, a dedicated link, and a dedicated secondary link station while the test is performed. Link-Level 1 allows sharing the PU_T4|5 node but dedicating the link and secondary link station. Link-Level 2 allows sharing both the PU_T4|5 node and link and dedicating only the link station.

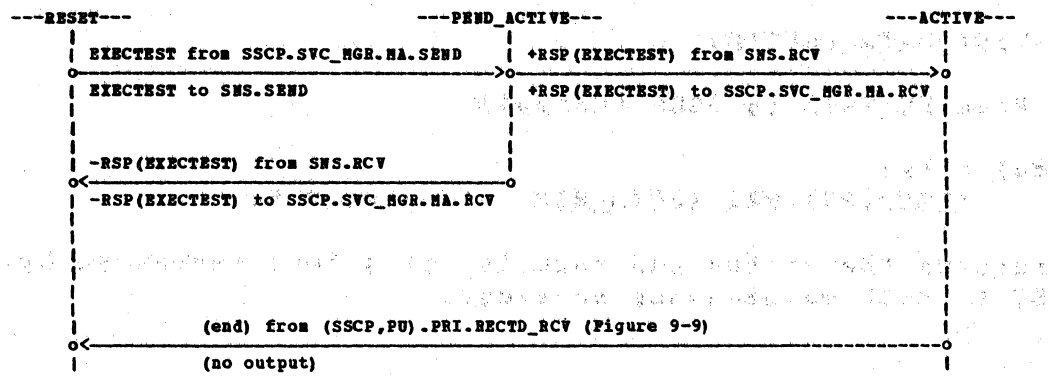


Figure 9-8. (SSCP,PU).PRI.TEST(na,n)_SEND

RECORD TEST DATA (RECTD)

Flow: From PU_T4|5 to SSCP (Normal)

Principal FSMs:
(SSCP,PU).PRI.RECTD_RCV (Figure 9-9)

RECTD returns the status and results of a test requested by EXECTEST to SSCP maintenance services.

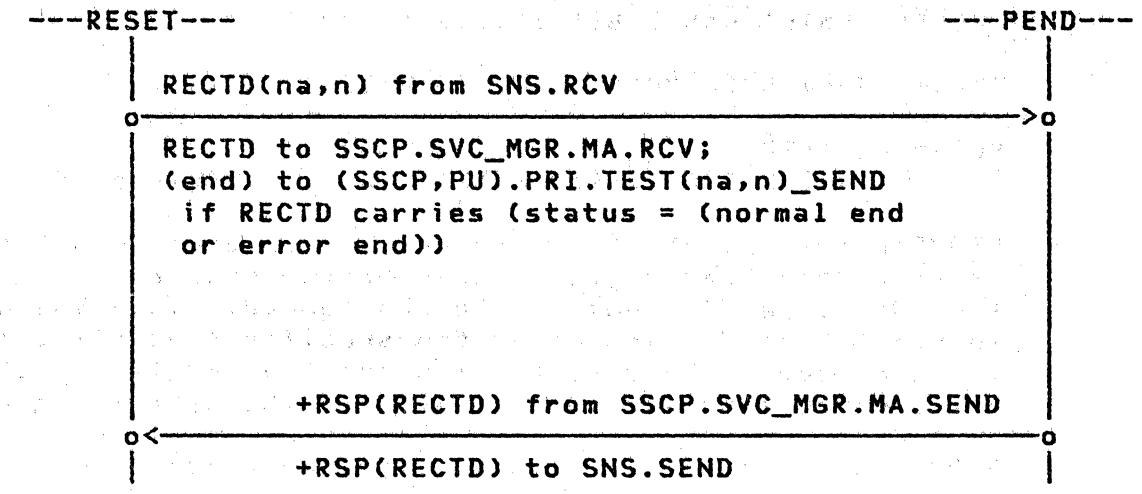


Figure 9-9. (SSCP.PU).PRI.RECTD_RCV

REQUEST MAINTENANCE STATISTICS (REQMS)

Flow: From SSCP|PUCP to PU (Normal)

Principal FSMs:
(SSCP,PU).PRI.REQMS(na,n)_SEND (Figure 9-10)

REQMS requests the CNM services associated with the PU to provide maintenance statistics for the resource indicated by the CNM target ID (na) in the CNM header. The Type code (n) in the CNM header indicates the specific statistics that are to be passed. These statistics are transmitted by the PU to maintenance services at the SSCP via the RECFMS request.

REQMS requests have the following Type codes:

000001 requests SDLC Test command/response statistics

000010 requests summary error data.

000011 requests error statistics from a peripheral PU.

000100 requests PU|LU dependent data.

000101 requests engineering change levels.

000110 requests link-connection subsystem (e.g., modem) data.

A reset indicator is on if the counters requested by REQMS are to be reset when RECFMS is sent. Refer to the description of RECFMS for details on the Type codes.

This request may be sent as an embedded NS RU in a FORWARD request. Refer to the description of FORWARD for details. REQMS cannot exceed 256 bytes.

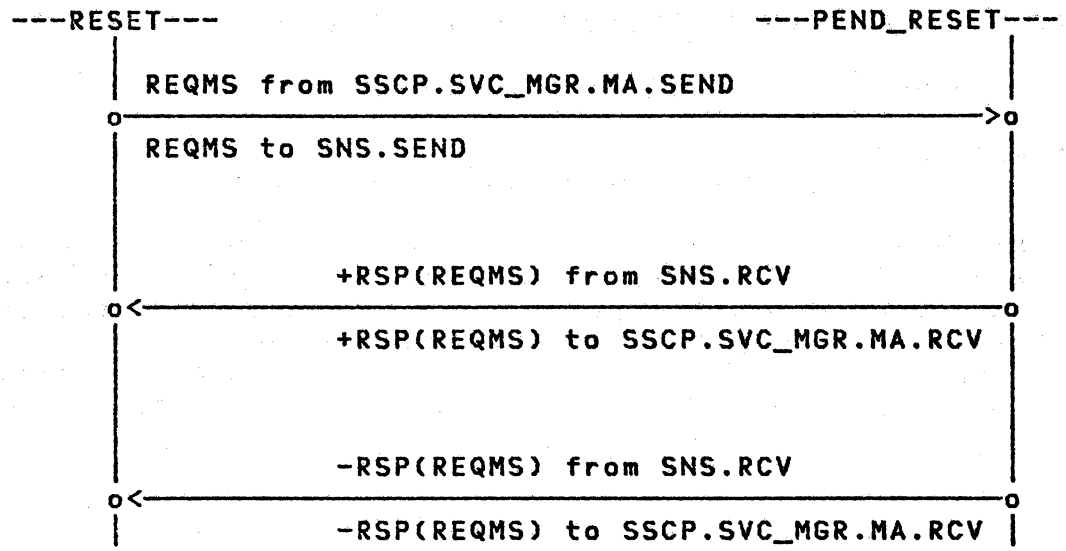


Figure 9-10. (SSCP,PU).PRI.REQMS(na,n)_SEND

RECORD FORMATTED MAINTENANCE STATISTICS (RECFMS)

Flow: From PU to SSCP|PUCP (Normal)

Principal FSMs:
(SSCP,PU).PRI.RECFMS(na,n)_RCV (Figure 9-11)

RECFMS permits the passing of maintenance related information from a PU to maintenance services at the SSCP. The information is generated by the CNM services associated with the PU. The CNM target ID (na) in the CNM header in the RECFMS indicates whether the statistics are for the PU or an LU in the node, or for a link or adjacent link station.

The Type code (n) in the CNM header indicates the specific statistics that are generated by the CNM services.

RECFMS requests have the following Type codes:

000000 reports an alert event and may convey information about the conditions that caused the event to be initiated.

000001 reports SDLC Test command/response statistics.

000010 reports summary error data.

000011 reports error statistics from a peripheral PU.

000100 reports PU|LU dependent data.

000101 reports engineering change levels.

000110 reports link connection subsystem (e.g., modem) data.

Counters are reset if the RECFMS is solicited by a REQMS with the reset indicator on. Counters are also reset if a RECFMS with any type code other than 000000 is sent unsolicited. If counters are to be reset, they are reset by CNMS at the time RECFMS is sent, i.e., before a +RSP(RECFMS) is received. Only counters of the type reported in that RECFMS are reset. Counters that reach their maximum value are not wrapped (reset) until reported via an unsolicited RECFMS, or until solicited via a REQMS with the counter reset indicator on.

RECFMS type 000000 is sent by the CNMS associated with a PU|LU to notify its CNMA(s) of an event that affects the PU|LU's ability to perform its intended functions and may require intervention and/or action (e.g., to correct a failure of its associated hardware). RECFMS type 000000

carries information from other RECFMS types to provide classification of the alert event. It may also provide the currently stored information relevant to the initiation of the alert event in RECFMS vectors appended to the RECFMS type 000000 request (see the RECFMS definition in Appendix E).

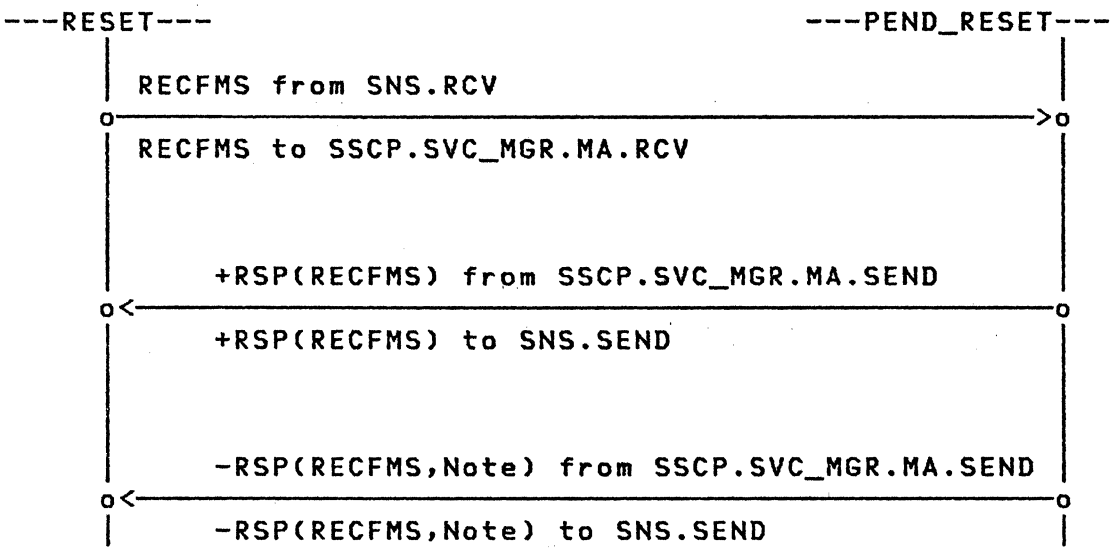
RECFMS type 000011 reports error statistics from a peripheral PU. These statistics are counts of selected errors, useful for problem determination, that have been supplied by the communication adapter. For these error statistics the RECFMS type 000010 communication adapter error counter is always incremented. The RECFMS type 000010 product error counter is also incremented for those communication adapter errors classified as internal errors by the product identified by the block number.

Multiple sets of counters are defined in order to accommodate different product implementations of communication adapter function.

The counters in RECFMS types 000010 and 000011 are positional. Counters that are not used are indicated in the validity mask field by a bit setting of 0 and their contents are to be ignored. Counters that are used are indicated in the validity mask field by a bit setting of 1.

All RECFMS types may be sent as unsolicited requests. RECFMS types other than 000000 may be solicited by a REQMS request. Multiple RECFMS reply requests may be sent in reply to a single REQMS. RECFMS cannot exceed 256 bytes.

This request may be embedded in a DELIVER request. See the description of DELIVER for details.



Note: Sense codes: 080C, 0812, 0815.

Figure 9-11. (SSCP,PU).PRI.RECFMS(na,n)_RCV

RECORD MAINTENANCE STATISTICS (RECMS)

Flow: From PU_T4|5 to SSCP (Normal)

Principal FSMs:
(SSCP,PU).PRI.RECMS_RCV (Figure 9-12)

RECMS permits the passing of maintenance statistics from a PU to a centralized recording facility at the SSCP. A PU may send statistics for itself, for its node, for supported links, or for adjacent link stations, as indicated by the network address in the request.

This request may be embedded in a DELIVER request. See the description of DELIVER for details.

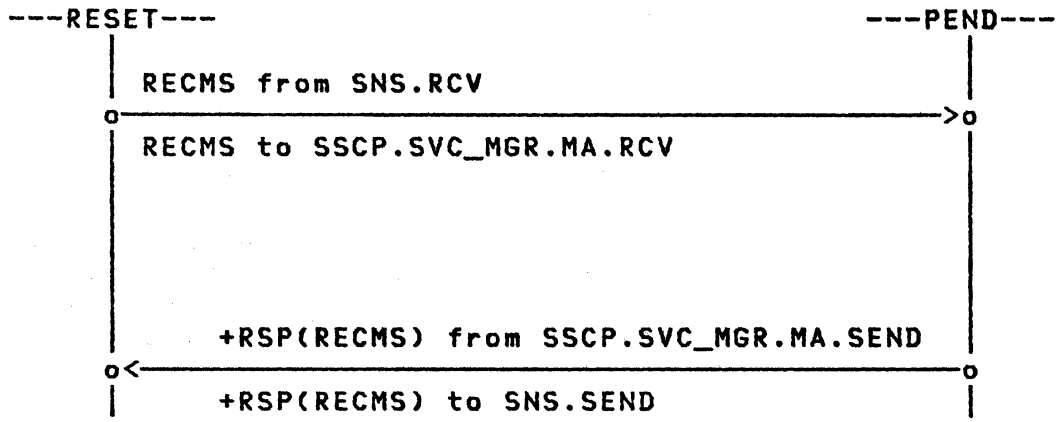


Figure 9-12. (SSCP,PU).PRI.RECMS_RCV

REQUEST TEST PROCEDURE (REQTEST)

Flow: From PU_T4|5 or LU to SSCP (Normal)

Principal FSMs:

(SSCP,LU).SEC.REQTEST(nn2,n)_SEND (Figure 9-13)
(SSCP,PU|LU).PRI.REQTEST(nn2,n)_RCV (Figure 9-14)

REQTEST requests that the specified test procedure (n) be executed for network name 2 (nn2) and be controlled by network name 1 (see Appendix E).

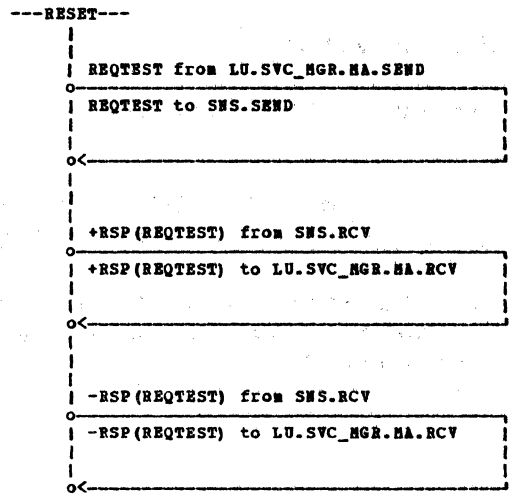
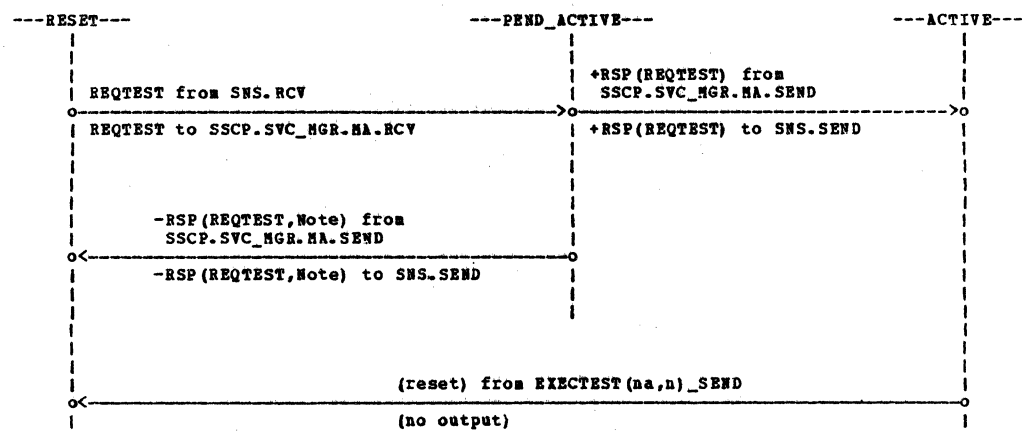


Figure 9-13. (SSCP,LU).SEC.REQTEST(nn2,n)_SEND



Note: Sense codes: 080C, 0812

Figure 9-14. (SSCP,PU|LU).PRI.REQTEST(nn2,n)_RCV

TEST MODE (TESTMODE)
RECORD TEST RESULTS (RECTR)

Flow: From SSCP to PU_T4|5 (Normal) for TESTMODE;
from PU_T4|5 to SSCP (Normal) for RECTR

Principal FSMs:
(SSCP,PU).PRI.TESTMODE_SEND (Figure 9-15)
(SSCP,PU).PRI.RECTR_RCV (Figure 9-16)

TESTMODE requests the CNM services associated with the PU to manage a test procedure. The test procedure begins with the TESTMODE request that initiates a test and ends when the test results and status are returned in a RECTR reply request corresponding to the initializing TESTMODE request. The test portion of the procedure is terminated under any of the following conditions:

- The test runs to completion; i.e., the test is self-terminating.
- A subsequent TESTMODE specifying test termination is received by CNM services.
- An error occurs.

The TESTMODE request denotes (1) test initiation and whether the test is self-terminating or continuous, or (2) test termination.

TESTMODE contains a CNM header. The target ID in the header indicates the resource that is to be tested. The Type code in the header specifies the test.

TESTMODE may be sent as an embedded NS RU in a FORWARD request. Refer to the description of FORWARD for details.

RECTR is the reply request corresponding to a TESTMODE request. It returns the results and status for the test. Multiple reply requests may be sent in answer to a single soliciting TESTMODE request. When TESTMODE initiates a continuous test, the RECTR(s) is sent in reply to the TESTMODE request that terminates the test. However, the PRID that is echoed in the CNM header of the replying RECTR is the PRID received in the TESTMODE that initiated the test.

RECTR may be embedded in a DELIVER request. Refer to the description of DELIVER for details.

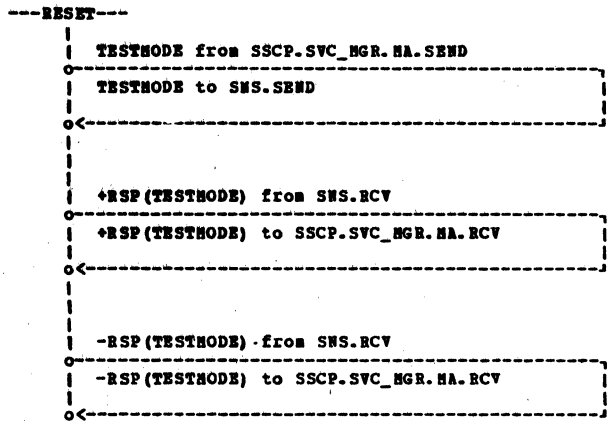


Figure 9-15. (SSCP,PU).PRI.TESTMODE_SEND

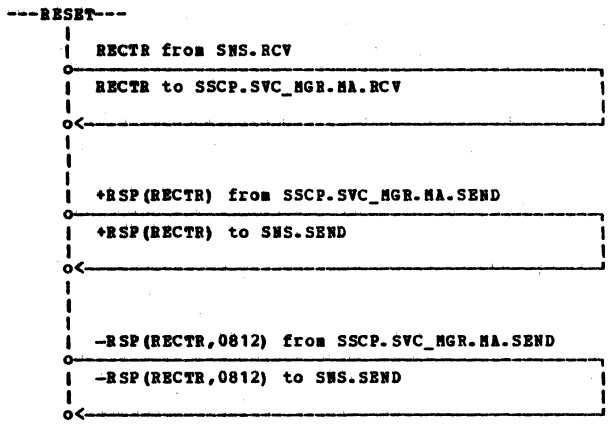


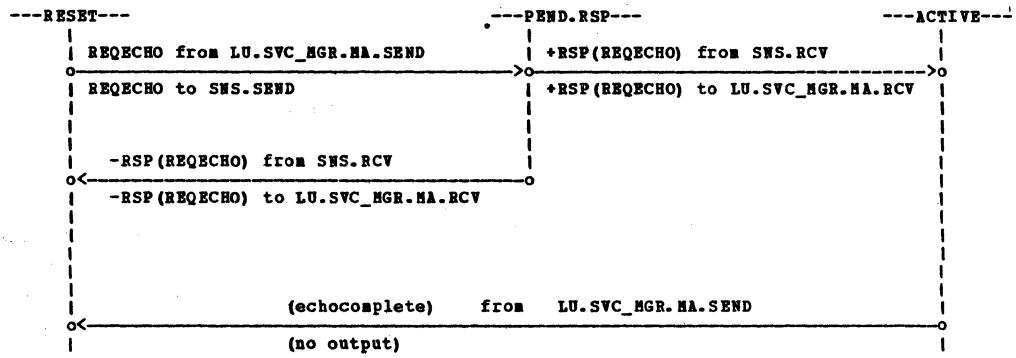
Figure 9-16. (SSCP,PU).PRI.RECTR_RCV

REQUEST ECHO TEST (REQECHO)

Flow: From LU to SSCP (Normal)

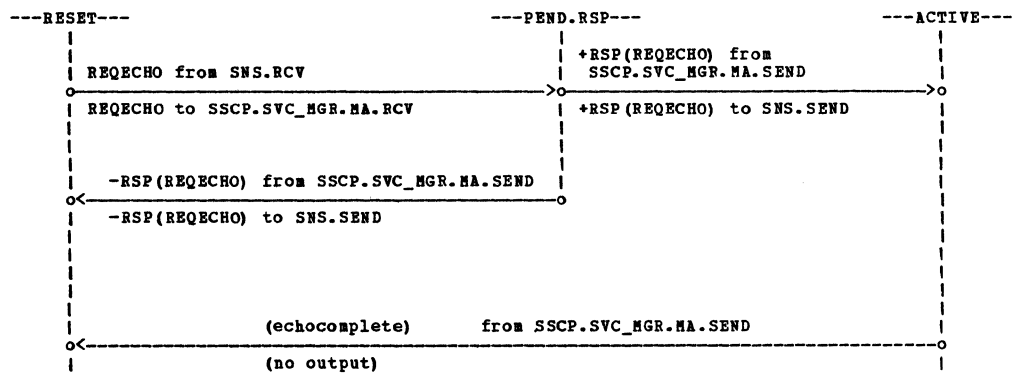
Principal FSMs:
(SSCP,LU).SEC.REQECHO_SEND (Figure 9-17)
(SSCP,LU).PRI.REQECHO_RCV (Figure 9-18)

REQECHO requests that the SSCP send to the LU the data included in REQECHO. The target LU for the test data is the LU that generated the REQECHO. The SSCP is to send the test data, via the ECHOTEST RU, to the target LU the number of times specified by the repetition factor in the REQECHO.



Note: Sense codes (in addition to those defined on the primary side): 8001, 8002, 8004, 8005

Figure 9-17. (SSCP,LU).SEC.REQECHO_SEND



Note: Sense codes: 1003, 0803, 0804, 0806, 0809, 0812, 0826, 0832 0835, 0842, 0857, 0859

Figure 9-18. (SSCP,LU).PRI.REQECHO_RCV

ECHOTEST (ECHOTEST)

Flow: From SSCP to LU (Normal)

Principal FSM: None

ECHOTEST carries test data to the target LU; the test data is the same as that carried in the corresponding REQECHO. The target LU is the LU that generated the REQECHO. The number of ECHOTESTs sent is specified by the repetition factor in the REQECHO, except that any -RSP(ECHOTEST) ends the sequence.

SET CONTROL VECTOR (SETCV)

Flow: From SSCP to PU_T4|5 (Normal)

Principal FSMs:
(SSCP,PU).PRI.SETCV_SEND (Figure 9-19)

This SETCV sets the intensive mode (X'08') control vector that is maintained by the PU receiving the request and that is associated with the network address specified in the RU.

For SETCV(configuration services) see Chapter 7.

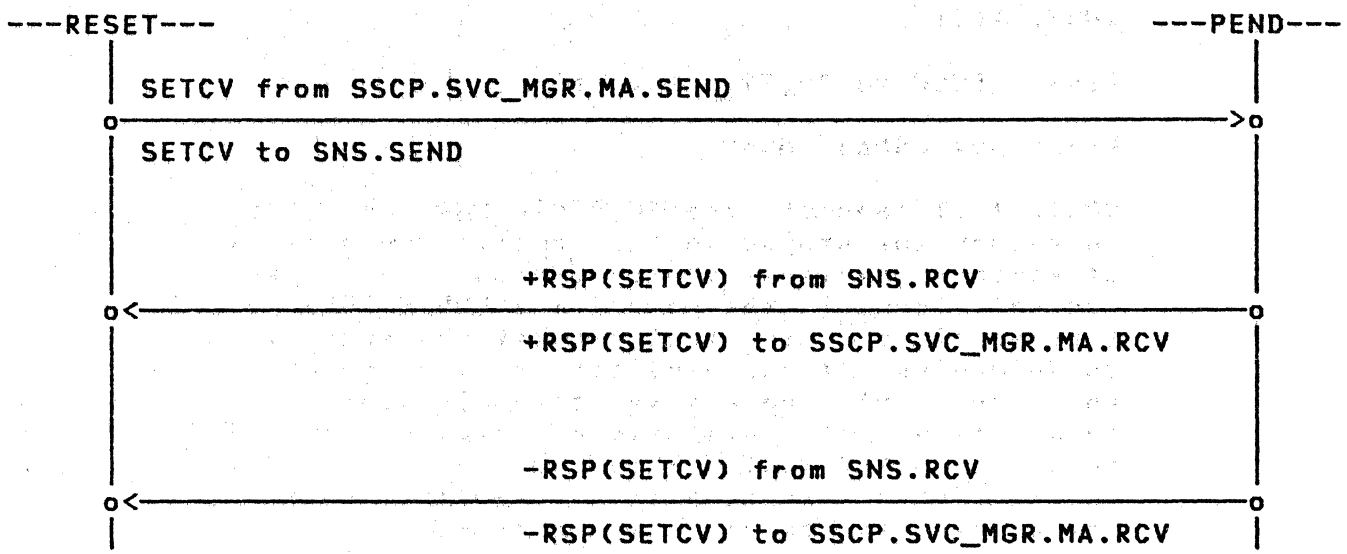


Figure 9-19. (SSCP,PU).PRI.SETCV_SEND

ROUTE_TEST

Flow: SSCP to PU_T4|5 (Normal)

Principal FSMs: None

ROUTE_TEST requests the PC_ROUTE_MGR component of PU.SVC_MGR to return the status (e.g., active, operative, not defined) of various explicit and/or virtual routes as known in the control blocks in the node. The ROUTE_TEST may also specify that explicit route test requests be sent into the network to determine if the explicit routes are operative, and if they are not operative to determine where they are inoperative. The statuses of explicit and virtual routes returned in RSP(ROUTE_TEST) and in ER_TESTED (which conveys to the SSCP the results of the explicit route test requests) may be sent to the network operator.

EXPLICIT ROUTE TESTED (ER_TESTED)

Flow: PU_T4|5 to SSCP (Normal)

Principal FSMs: None

ER_TESTED is sent by a subarea node to one or more SSCPs to provide the the status of an ER as determined by explicit route test procedures. When an NC_ER_TEST fails, the ER manager detecting the failure sends an ER_TESTED for each SSCP-PU session that has received SDT. When an NC_ER_TEST succeeds and the corresponding NC_ER_TEST_REPLY reaches its destination (i.e., the originator of the NC_ER_TEST), the ER manager at that node sends an ER_TESTED to the SSCP that originated the ROUTE_TEST that requested the explicit route be tested. In both cases the information obtained from the ER_TESTED may be sent to the network operator.

DELIVER (DELIVER)

Flow: From SSCP to LU (Normal)

Principal FSMs:
(SSCP,LU).PRI.DELIVER_SEND (Figure 9-20)
(SSCP,LU).SEC.DELIVER_RCV (Figure 9-21)

DELIVER contains an embedded NS RU. A flag in the DELIVER RU indicates whether the NS RU contains a CNM header. An embedded NS RU is either a reply request corresponding to an NS RU embedded in a FORWARD request, or it is an unsolicited request. The procedure-related identifier (PRID) is the only information that the SSCP may modify in an embedded NS RU. The SSCP changes PRIDs when doing its own PRID-based routing.

All names sent in a DELIVER request are network names.

The name in the Origin field is derived from the half-session from which the SSCP receives the NS RU to be embedded in the DELIVER.

When the NS RU to be embedded has a CNM header, the name in the Target field is derived from the CNM target ID descriptor and the CNM target ID. When the NS RU to be embedded has no CNM header, the name in the Target field is derived from the network address in the NS RU. When the NS RU to be embedded is originated by an LU, the resource identified in the NS RU may only be an LU.

A configuration hierarchy list provides configuration information about a target. See appendix E for detailed description of the hierarchy list.

NS RUs to be embedded in a DELIVER request are processed by management services at the SSCP.SVC_MGR. The SSCP SNS processing for embedded NS RUs consists solely of a pass-through routing function; FSMs are not maintained.

NS RUs that may be embedded in a DELIVER are RECFMS, RECMS, RECSTOR, and RECTR.

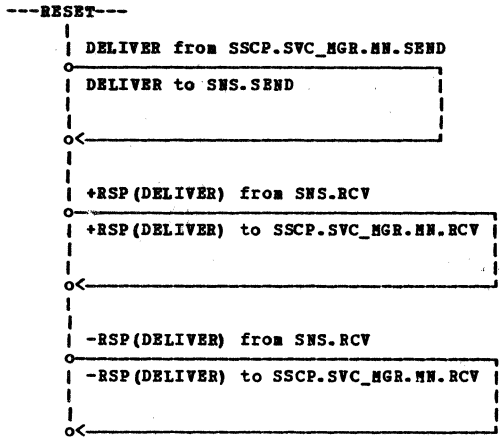


Figure 9-20. (SSCP,LU).PRI.DELIVER_SEND

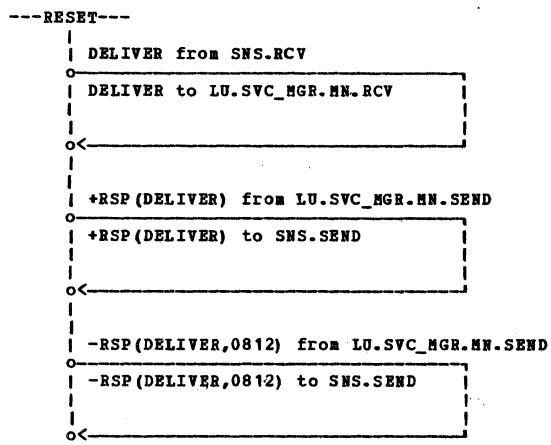


Figure 9-21. (SSCP,LU).SEC.DELIVER_RCV

FORWARD (FORWARD)

Flow: From LU to SSCP (Normal)

Principal FSMs:
(SSCP,LU).SEC.FORWARD_SEND (Figure 9-22)
(SSCP,LU).PRI.FORWARD_RCV (Figure 9-23)

FORWARD requests the SSCP to send the embedded NS RU to the named destination PU|LU, using the corresponding SSCP-PU|LU session. The FORWARD RU contains a flag that specifies whether the embedded NS RU contains a partially initialized CNM header or no CNM header at all.

All names sent in a FORWARD request are network names.

FORWARD always contains a destination name and a target name. The SSCP selects the appropriate SSCP-PU session to forward the embedded NS RU based upon the destination network name.

When the destination is a PU|LU in a subarea node, the SSCP resolves the target name to a network address. When the destination is a PU|LU in a peripheral node, the SSCP resolves the target name to the appropriate one-byte local address form. In any case, the SSCP completes the CNM header initialization by entering the CNM target ID and CNM target descriptor in the CNM header based on the address derived from the target name and on the PU type at the destination.

Embedded NS RUs without CNM headers are sent only to a subarea PU. All embedded NS RUs, whether or not they contain a CNM header, have the network address of the target at a common location in the FORWARD RU.

An SSCP optionally queues or rejects a FORWARD request when the SSCP half-session for the embedded NS RU destination operates in immediate request mode and definite response protocol, and a prior normal-flow RQD request is outstanding on the half-session.

The NS header category field in an embedded NS RU always denotes same-domain.

NS RUs embedded in FORWARD requests are processed by management services at the SSCP.SVC_MGR. The SSCP SNS processing for an embedded NS RU consists only of a send usage check on the destination PU request receive capability and of a pass-through routing by SNS.SEND; FSMs are not maintained.

NS RUs that may be embedded in FORWARD are: SETCV, DISPSTOR, TESTMODE, and REQMS.

The response to a FORWARD request is not sent until the response to the embedded NS RU is received at the SSCP. If the SSCP intends to respond negatively to a FORWARD request, it does so and the embedded NS RU is not forwarded.

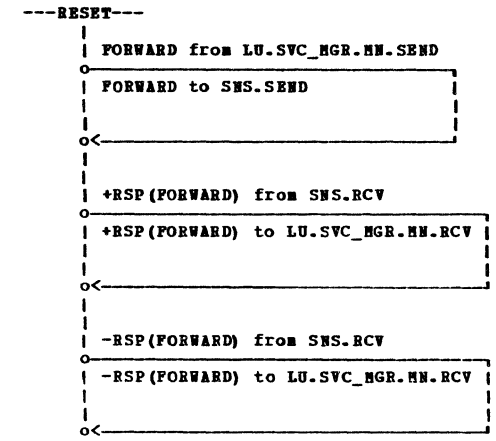
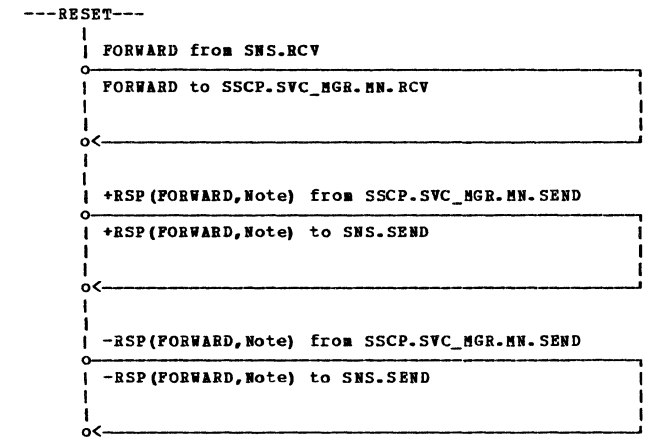


Figure 9-22. (SSCP,LU).SEC.FORWARD_SEND



Note: The SSCP may respond negatively to a FORWARD request using one of the following sense codes: 0806, 080E, 0812, 0851, 1002, 1003 or 1007. A negative response is also sent if the SSCP receives a negative response to the NS RU that was imbedded in the FORWARD and forwarded. In this case, the sense code received by the SSCP to the forwarded NS RU is used on the negative response to FORWARD. Note that the CNM application cannot necessarily distinguish the sense code originator.

Figure 9-23. (SSCP,LU).PRI.FORWARD_RCV

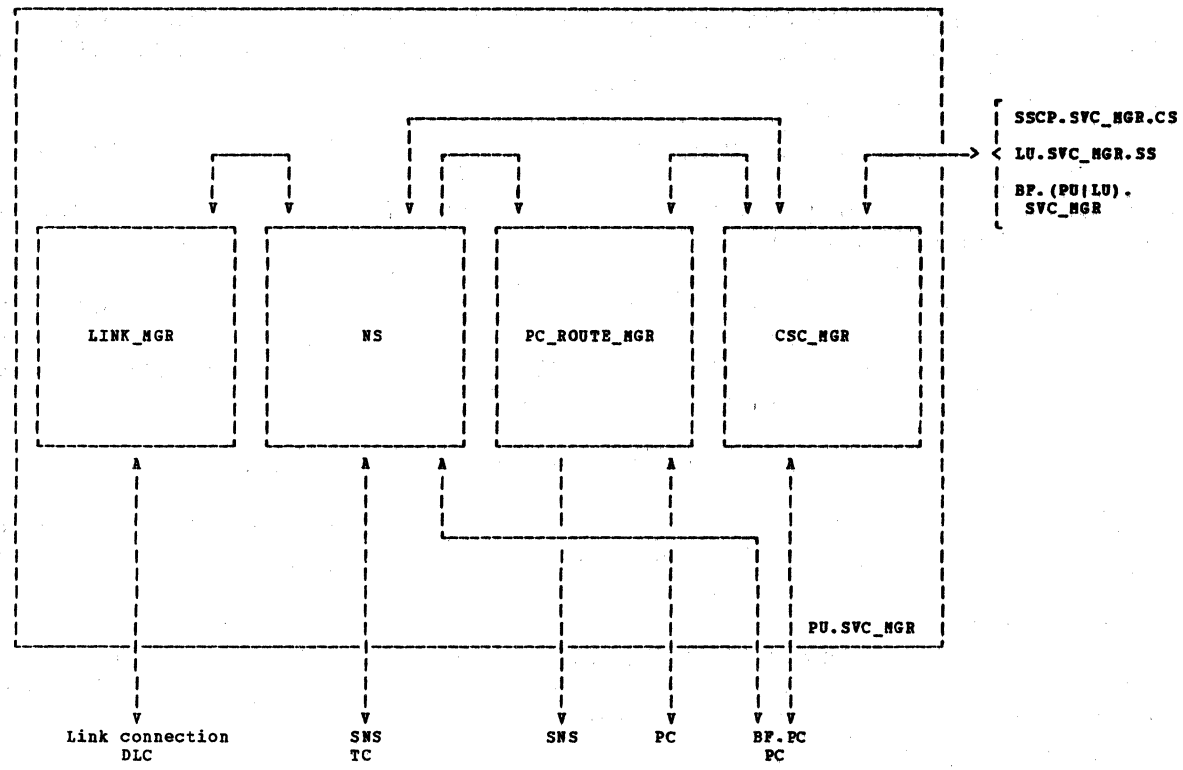
This page intentionally left blank

PU.SVC_MGR STRUCTURE

The PU.SVC_MGR (Figure 10-1) is composed of the following components:

- Network services (PU.SVC_MGR.NS or NS)
- Common session control manager (PU.SVC_MGR.CSC_MGR or CSC_MGR)
- Path control route manager (PU.SVC_MGR.PC_ROUTE_MGR or PC_ROUTE_MGR)
- Link manager (PU.SVC_MGR.LINK_MGR or LINK_MGR)

The NS component and resource finite-state machines are defined in Chapter 11; the VR and ER managers and route finite-state machines are defined in Chapter 12; the CSC manager component and session finite-state machines are defined in Chapter 13; the data structures that the PU.SVC_MGR components use are defined in Appendix A. The link manager is not defined in this book.



Note:
The components of PU.SVC_MGR are described in the following chapters:

Component	Chapter
PU.SVC_MGR.NS	11
PU.SVC_MGR.PC_ROUTE_MGR	12
PU.SVC_MGR.CSC_MGR	13
PU.SVC_MGR.LINK_MGR	*

* The PU.SVC_MGR.LINK_MGR is not described in this book.

Figure 10-1. Structure of PU.SVC_MGR

PU.SVC_MGR.NS COMPONENT

The NS component processes requests that are received on PUCP-PU or SSCP-PU half-sessions. The following functions are contained in this component:

- Receive all CP-PU half-session requests and responses.
- Session control processing of ACTPU, DACTPU, and, for a subarea node, SDT.
- Link and link station management including functions common to all DLCs and protocol verification are done in this component. (The DLC and link connection unique functions are handled by the LINK_MGR.)
- Management of subarea element address space.
- Management of the loading or dumping of an adjacent subarea node, or the loading of an adjacent type 2 node.
- Provision of a protocol boundary with communication network management services (see Chapter 9) for maintenance requests and responses.
- Provision of a routing function between the CP-PU half-sessions and path control route manager.

PU.SVC_MGR.PC_ROUTE_MGR COMPONENT

The PC route manager component of the PU.SVC_MGR exists only in subarea nodes and manages virtual and explicit routes. It consists of two major sections: an explicit route manager (ER_MGR) and a virtual route manager (VR_MGR). These two managers interact to set up routing for sessions. The PC route manager component has a single protocol boundary routine called PU.SVC_MGR.PC_ROUTE_MGR.RCV that receives input from other components of the PU.SVC_MGR as well as from the explicit route control component of path control (See Chapter 3). The PC route manager component sends output to transmission group control, explicit route control, CSC manager, and to SSCP-PU half-sessions.

PU.SVC_MGR.CSC_MGR COMPONENT

The CSC manager handles the activation and deactivation of both locally supported and boundary function supported half-sessions. Path control and the services managers direct all session activation and deactivation requests and responses to CSC manager for processing. CSC manager maintains session FSMs to remember the status of the half-sessions.

In addition, CSC manager controls session outage notification. When CSC manager is informed by the PC route manager of conditions that disrupt traffic flow between half-sessions, it identifies the affected sessions and, depending upon the specific cause of the outage, sends session deactivation requests to each affected half-session.

PU.SVC_MGR.LINK_MGR COMPONENT

The link manager is a generic component that manages the unique aspects of DLCs (e.g., SDLC and S/370 channel) and of link connections. The aspects of data link control common to all DLCs are managed by the network services component; however, the transformation of the generic requests into the protocol required for a specific type of data link control is managed by the link manager.

The link manager component provides management function not only for the DLC layer (e.g., interpreting CONTACT and sending and receiving XID), but also for the link connection by interpreting commands such as ACTLINK and DACTLINK as well as functions associated with switched links (e.g., auto dial and auto answer).

The link manager is not documented in this book.

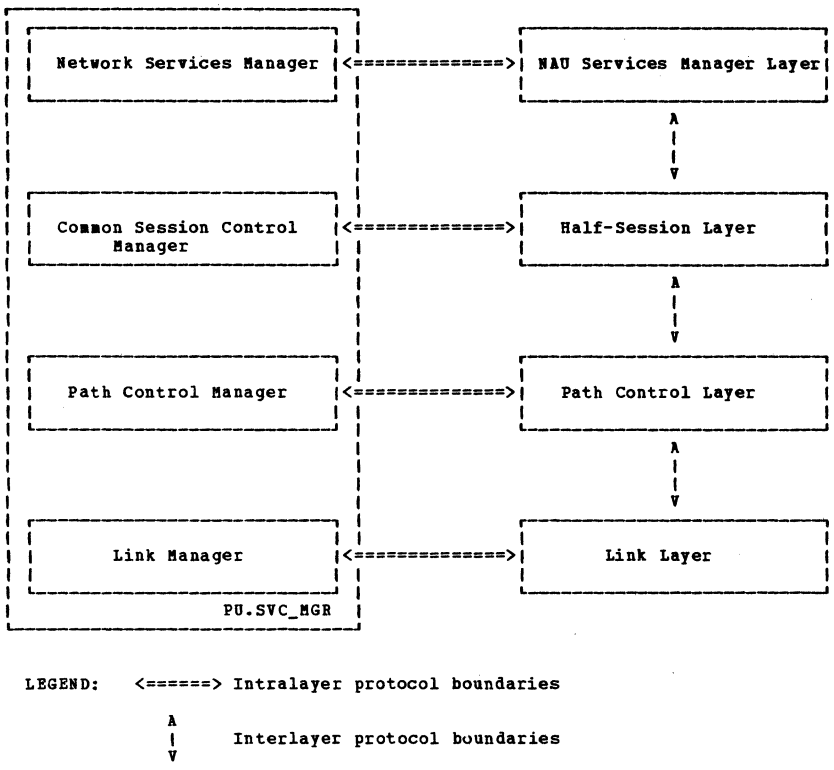


Figure 10-2. PU.SVC_MGR as a manager of SNA layers

NODE LAYER MANAGEMENT

The PU.SVC_MGR is the collection of function that manages and controls the functional layers of a node (Figure 10-2). The functional layers consist of links, path control, half-sessions, and NAU services. The PU.SVC_MGR manages the resources used by half-sessions, path control, and links. Nodes may attach to other resources (e.g., files and devices); these are managed by the other service managers (SSCP, LU, BF.LU, and BF.PU) and may involve common service functions provided by the operating environment of the node (e.g., buffer allocation and sharing of objects).

Each node contains one and only one PU.SVC_MGR. A node may be defined by the resources that are controlled by a PU.SVC_MGR. Each layer of the architecture exchanges RUs and control information with the PU.SVC_MGR.

MANAGING THE LINK LAYER

The DLC layer (SDLC or System/370 Channel) exchanges control information with the PU.SVC_MGR. This information is in the form of signals, except for XID (Exchange identification), which is a DLC-level message unit (see Appendix E). Some of this control information controls the link connection (e.g., answering an incoming call or providing dial digits), while other information is translated into unique protocols on the physical medium (e.g., CONTACT into either an SDLC SNRM command or UA response).

The PU.SVC_MGR.NS component does validation on requests received from a control point, but does not translate these requests into specific actions to be taken with respect to the link. The PU.SVC_MGR.LINK_MGR does this translation for the specific link control type involved.

MANAGING THE PATH CONTROL LAYER

The functions of path control are to provide a signaling path for requests and responses between two half-sessions, and to provide intermediate node routing for routes that use an intermediate node. Path control has three basic types: a simple form in a peripheral node; the corresponding function in a subarea node to support connections to peripheral nodes; and a form that connects subarea nodes to other subarea nodes, which involves transmission groups, explicit routes, and virtual routes.

The management of this layer requires the creation of a correspondence between address fields in a TH (any FID type) and half-sessions identified by session control blocks. The creation of this correspondence is part of the PU.SVC_MGR.CSC_MGR function. The management of subarea path control requires additional support from the PU.SVC_MGR to

manage the three sublayers (i.e., transmission group control, virtual route control, and explicit route control).

Managing the Transmission Group Control Sublayer

Transmission groups are managed by the PU.SVC_MGR.NS component. A transmission group is controlled by managing the individual links that comprise the group. A TG is operational when the first link becomes usable for data exchange and remains functional until no links are functional. Information necessary for transmission group bring up by exchanging XID (format 2) commands and responses between the subarea nodes. This exchange is managed by PU.SVC_MGR.NS. One additional function, TG trace, is also managed by PU.SVC_MGR.NS. This function is enabled by the receipt of an ACTTRACE request specifying the TG trace option and is terminated by DACTTRACE.

Managing the Explicit Route Control Sublayer

Explicit routes are managed by the PU.SVC_MGR.PC_ROUTE_MGR component. Explicit routes are defined at system definition time and may be dynamically added. The activation of these routes is managed by the PC route manager. Deactivation of the route occurs only because of physical path failure and is not initiated by control point session requests, or by the PU services manager. The primary function of explicit route control is to determine if the RU is destined for this subarea or another subarea. If it is destined for another subarea, the explicit route number and destination subarea address are used to access the SUBAREA_ROUTING list to determine the outbound transmission group. If the RU is destined for this subarea it is forwarded to virtual route control. The explicit route component manages the entries in the SUBAREA_ROUTING list.

Managing the Virtual Route Control Sublayer

Virtual routes are also managed by the PC route manager. Virtual routes are activated when the first session using the route is activated, and are deactivated when all half-sessions using the route have been deactivated. The VR manager's primary function is to assign a half-session with a virtual route. The VR manager creates a VRCB for each virtual route that is activated and when it is informed that the session count has gone to 0, deactivates the virtual route and discards the VRCB.

MANAGING THE HALF-SESSION

Half-sessions are managed by the PU.SVC_MGR.CSC_MGR component. All session activation and deactivation RUs (ACTCDRM, ACTLU, ACTPU, BIND, DACTCDRM, DACTLU, DACTPU, and UNBIND) are intercepted by path control in the destination

node or boundary function and are forwarded to PU.SVC_MGR.CSC_MGR. These RUs are used to construct a session control block for the half-session. Part of the session management function for resetting and initializing session control blocks is described as subroutines in the transmission control and data flow control chapters. (See Chapters 4 and 5). These subroutines can be considered logical extensions of the PU.SVC_MGR.CSC_MGR.

In the event of a failure along the ER used by a session, session outage notification takes place to notify the NAU services managers managing the half-sessions at the ends of the route. This notification function is part of PU.SVC_MGR.CSC_MGR, but the detection of the failure is part of other components in the node.

MANAGING THE NAU SERVICES MANAGERS

In a subarea node the PU services manager manages its own element address space and provides management support for any boundary function contained in its node.

Managing Subarea Element Addresses

Links, adjacent link stations, and LUs may be dynamically added and removed from the address space of the subarea node by RUs that flow from a control point to the PU.SVC_MGR (e.g., RNAA and FNA). The PU.SVC_MGR protects sessions using these resources from damage resulting from removing or redefining a resource.

Managing Boundary Function PUs and LUs

Management of the boundary function is limited to tailoring the boundary function support to the particular type of node attached to the boundary function. This is accomplished through the processing of SETCV type 3 (SDLC Secondary Station control vector) and type 4 (LU control vector). These control vectors specify the type of PU attached and other parameters used by boundary function path control and half-session layers.

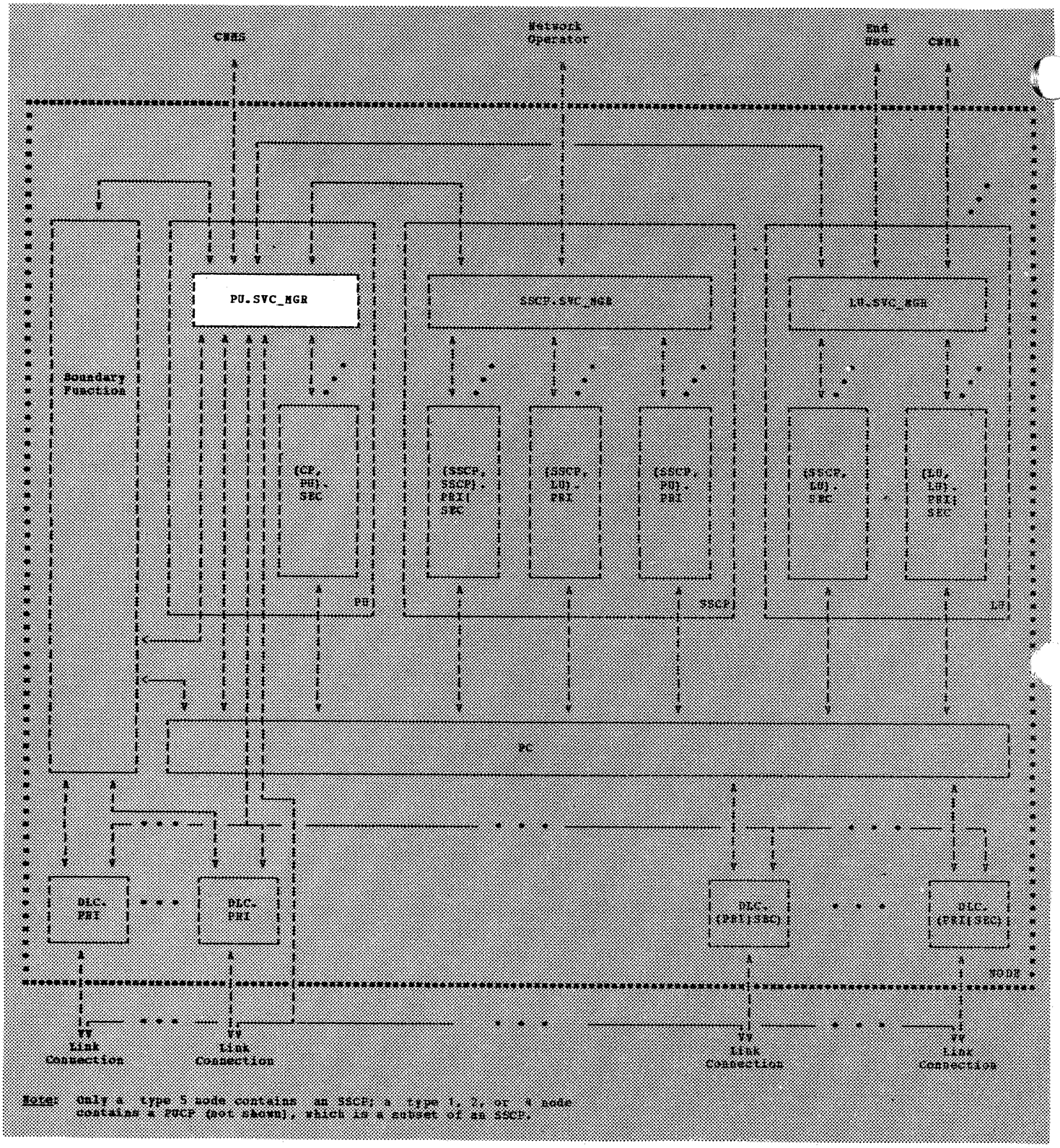
PU SERVICES MANAGER (NETWORK SERVICES) GENERAL DESCRIPTION

Every node in an SNA network contains one PU, one PU services manager, and one PU-based network services component (PU.SVC_MGR.NS). Within the node, the function of the PU.SVC_MGR (see Figure 11-1) is to control the resources belonging to the PU (e.g., links and adjacent link stations). The PU.SVC_MGR.NS component participates in providing configuration and maintenance services and network control for the network, via sessions formed with SSCPs, and with its own PU control point (PUCP).

The PUCP is a functional subset of an SSCP, being basically an SSCP substitute at a non-PU_T5 node responsible for activating the PU and its local link resources. The PUCP communicates with the PU via path control in the same way that two NAUs forming a session in the same node do so. The PUCP is addressable by the PU, and the PUCP-PU session protocols are identical to those of the SSCP-PU for the subset described; the structure of the PUCP is undefined, being a product option. Because of their similarity, the term "control point" (CP) is used, where applicable, to refer to either an SSCP or a PUCP.

A representation of each resource within, or controlled by, the node is contained in the node resource list (see Appendix A). This list is created by an implementation- and installation-dependent process. The node resource list is managed by the PU.SVC_MGR.NS, and a representation of a resource can be added to, or deleted from, the node resource list by the PU.SVC_MGR.NS.

An important task of the PU.SVC_MGR.NS is supervising the shared control of the PU and links in the node. When a PU or link station may be activated by more than one SSCP it is said to be under shared control. For example, the PU.SVC_MGR.NS requests the PU.SVC_MGR.LINK_MGR to activate a link when the first CP requests it be activated via the ACTLINK RU; the request to deactivate the link is sent to PU.SVC_MGR.LINK_MGR by the PU.SVC_MGR.NS only when all CPs that had requested activation of the link have requested the link be deactivated, or when the link has been reset by a lost control point reset or INOP processing.



Note: Only a type 5 node contains an SSCP; a type 1, 2, or 4 node contains a PUCP (not shown), which is a subset of an SSCP.

Figure 11-1. Structural Overview of a Node

If a resource is under shared control, there is a concurrency share limit. A resource's share limit is the maximum number of CPs that may have concurrently active control over the resource. The PU.SVC_MGR.NS enforces the share limit of the PU, links, and adjacent link stations.

Within the PU.SVC_MGR.NS there exist resource finite-state machines that provide information about the state (e.g., active, reset, or pending active) of the physical resource (e.g., PU or link) that is represented by the resource FSM. The finite-state machines that describe the CP's interaction with a specific resource are defined in Chapter 7. All the resource FSMs that describe the state of the PU's resource have a suffix of RES (e.g., FSM_LINK_ACT_RES).

The states of the resource FSM for a given shared resource are coupled with those of the CP's FSM, e.g., the resource FSM becomes active when an activation request is received from the first CP, and is reset only after all CPs have deactivated it. For example, if one CP's FSM_LINK_ACT_DOM_RES is in the ACTIVE state, the FSM_LINK_ACT_RES is in the ACTIVE state. If all the CP's FSM_LINK_ACT_DOM_RESs are in the RESET state, the FSM_LINK_ACT_RES is in the RESET state. The FSM_LINK_ACT_RES may temporarily be in an ACTIVE state when the CP's FSM_LINK_ACT_DOM_RES is not in the ACTIVE state; however, the CP's FSM will be in the same state as the resource FSM when the CP receives the response to the ACTLINK.

This chapter describes only the resource FSMs, not the coupled CP's FSMs, which are described in Chapter 7. Figure 11-2 presents a list of the RUs and signals that can affect the resource FSMs, the resource FSM name, and the CP FSM name. This does not imply that an RU is checked only against the state of the FSM listed; for example, a DUMPINIT is checked not only against the state of FSM_ALS_SEC_DUMP_RES as listed, but also the IPL, RPO, LOAD, CONTACT, and DISCONTACT resource FSMs (shown in the procedure, but not in the list).

Request Code	Node Resource FSM Name (Note 1)	Control Point Domain Resource FSM
ABCONN	FSM_ALS_CONNECTED_RES	FSM_ALS_CONNECTED_DOM_RES
ABCONNOUT	FSM_LINK_CONNOUT_RES	FSM_LINK_CONNOUT_DOM_RES FSM_ALS_CONNECTED_DOM_RES
ACTCONNIN	FSM_LINK_CONNIN_RES	FSM_LINK_CONNIN_DOM_RES FSM_ALS_CONNECTED_DOM_RES
ACTLINK	FSM_LINK_ACT_RES	FSM_LINK_ACT_DOM_RES
ACTPU	FSM_PU_ACT_RES	FSM_PU_ACT_DOM_RES
CONNOUT	FSM_LINK_CONNOUT_RES	FSM_LINK_CONNOUT_DOM_RES FSM_ALS_CONNECTED_DOM_RES
CONTACT	FSM_ALS_CONTACT_DISCONTACT_RES	FSM_ALS_CONTACT_DOM_RES
CONTACTED	FSM_ALS_CONTACT_DISCONTACT_RES	FSM_ALS_CONTACT_DOM_RES
DACTCONNIN	FSM_LINK_CONNIN_RES	FSM_LINK_CONNIN_DOM_RES FSM_ALS_CONNECTED_DOM_RES
DACTLINK	FSM_LINK_ACT_RES	FSM_LINK_ACT_DOM_RES
DACTPU	FSM_PU_ACT_RES	FSM_PU_ACT_DOM_RES
DISCONTACT	FSM_ALS_CONTACT_DISCONTACT_RES	FSM_ALS_CONTACT_DOM_RES
DUMPFINAL	FSM_ALS_SEC_DUMP_RES	FSM_ALS_DUMP_DOM_RES
DUMPINIT	FSM_ALS_SEC_DUMP_RES	FSM_ALS_DUMP_DOM_RES
DUMPTEXT	FSM_ALS_SEC_DUMP_RES	FSM_ALS_DUMP_DOM_RES
IPLFINAL	FSM_ALS_SEC_IPL_RES	FSM_ALS_IPL_DOM_RES
IPLINIT	FSM_ALS_SEC_IPL_RES	FSM_ALS_IPL_DOM_RES
IPLTEXT	FSM_ALS_SEC_IPL_RES	FSM_ALS_IPL_DOM_RES
RPO	FSM_ALS_SEC_RPO_RES	FSM_ALS_RPO_DOM_RES
XID_COMPLETED (Note 2)	FSM_ALS_SEC_XID_RES	None

NOTES:

1. All other requests do not have resource FSMs associated with them in the PU.SVC_MGR.NS.

2. XID_COMPLETED is not a request code but is a signal representing the SDLC XID response.

Figure 11-2. Correspondence of Node Resource FSMs to CP DomainResource FSMs

PU.SVC_MGR.NS STRUCTURE

The NS component is decomposed into the following components (see Figure 11-3):

- NS.RCV, which handles the receiving of all requests, responses, and signals from half-sessions, the common session control manager, and the data link control managers.
- NS.CS_RCV, which is called by PU.SVC_MGR.NS.RCV to handle the requests and responses for configuration services that are received from SNS.RCV.
- NS.DLC_RCV or NS.DLC_CONFIG, which are called by PU.SVC_MGR.NS.RCV to handle the responses and signals that are received from LINK_MGR (either SDLC_MGR or S370_CHAN_MGR).
- NS.SC_PROC, which is called by PU.SVC_MGR.NS.RCV to handle the requests for activation or deactivation that are received from PU.SVC_MGR.CSC_MGR.
- NS.MS_PROC, which is called by PU.SVC_MGR.NS.RCV to handle the requests and responses for maintenance services that are received from SNS.RCV.

Each of the above components is represented as a procedure whose detailed description may be found in the FAPL description section.

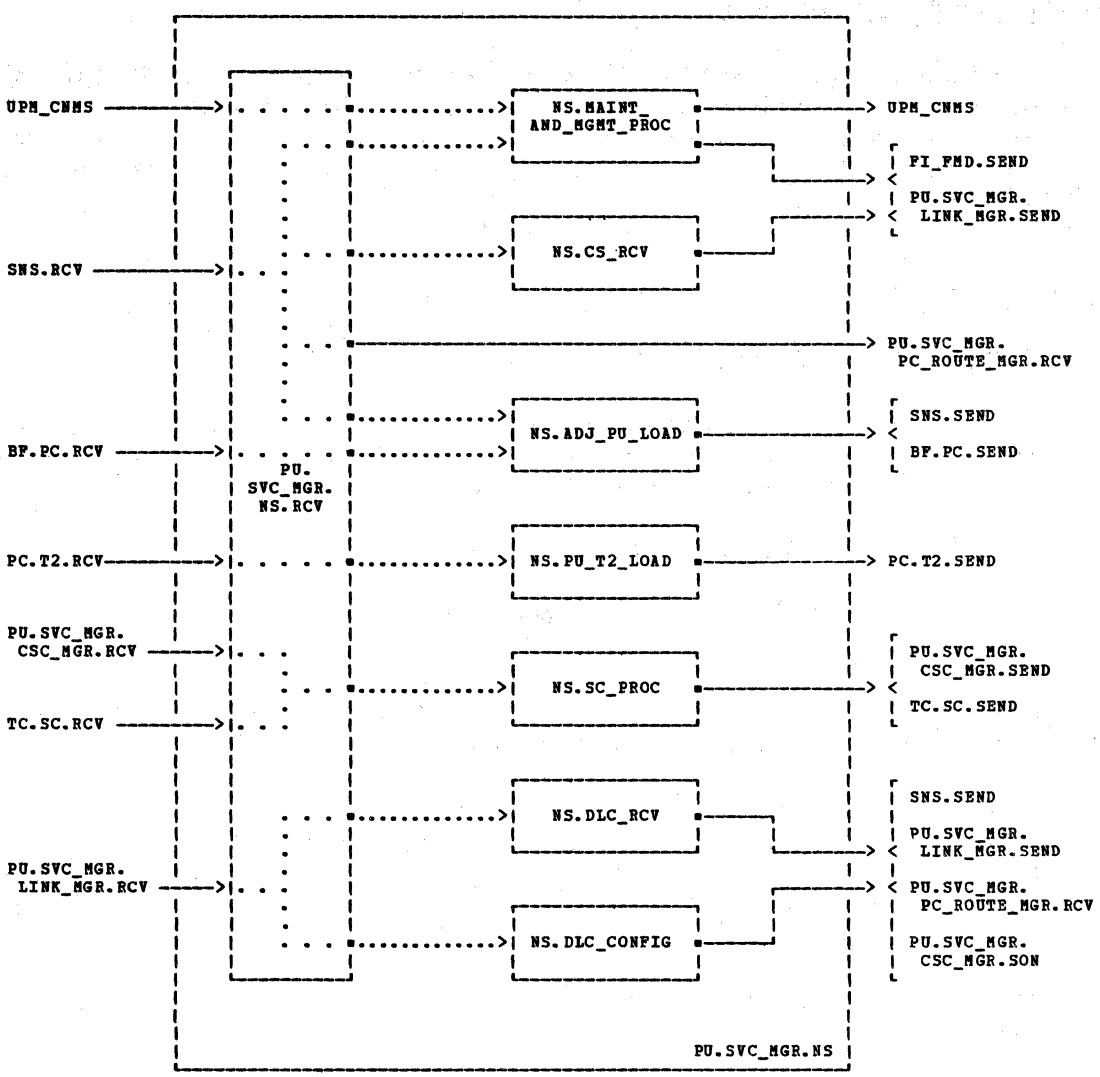


Figure 11-3. Structure of PU.SVC_MGR.NS

PU.SVC_MGR.NS PROTOCOL BOUNDARIES

The protocol boundary information for the PU.SVC_MGR.NS depends on the sender of the RU or signal; the specific protocol boundary information is contained in NS.CS_RCV (page 11-34), NS.DLC_RCV (page 11-76), and NS.SC_PROC (page 11-30) procedures.

PU.SVC_MGR.NS FUNCTIONS

PU.SVC_MGR.NS provides the following functions:

- Activating and deactivating the SSCP-PU session, which includes the checking and retaining of the parameters contained in the ACTPU request.
- Loading an adjacent PU_T2 node when capable.
- Initiating link-level procedures when requested by a CP.
- Preventing interruption of necessarily continuous functions, such as certain link-level procedures.
- Generating multiple requests or responses from a single request or response, when appropriate. For example, sending INOP to all CPs that actively control a link when LINK_MGR sends INOP to the PU.SVC_MGR.NS, notifying it that the link has become inoperative.
- Making send checks, to avoid violation of half-session protocols.
- Resetting of appropriate resource FSMs, as a result of losing a control point or of receiving INOP from LINK_MGR.
- Causing XID to be sent on successful completion of connect-in or connect-out
- Managing shared control of the resources belonging to the PU.SVC_MGR.NS.
- Enforcing the share limits of resources under shared control.

SHARE LIMITS

Each PU, LU, link, and adjacent link station has a share limit. The share limit of a resource is the maximum number of CPs that may concurrently control the resource. The PUCP

is included in the count made to test for share limit exceeded for any resource represented in the node resource list. For some of these resources, the share limit is one; thus, they may be considered to be only sequentially, not concurrently, shared. Only PU_T4s, PU_T5s, links, and certain adjacent link stations may have a share limit greater than one. For a node, the share limit of the PU, of each link, and of each adjacent link station is stored as a parameter in the node resource list. The share limit of a link cannot exceed the share limit of the PU; the share limit of an adjacent link station cannot exceed the share limit of the link.

The concurrency count of a PU, link, or adjacent link station is calculated by counting the number of CPs that are in the CP list for that resource (Appendix A); this count is always less than or equal to the share limit. The share limit of the link and adjacent link station associated with a switched link connection is always 1.

SERIALIZATION OF DLC

Link procedures other than for ACTLINK, CONTACT, and DISCONTACT are serialized, as these procedures may be conducted only one at a time. In these cases, the PUCP is considered in the serialization of the procedure. The PU.SVC_MGR.NS recognizes that the LINK_MGR requires serialization by maintaining a PEND state in the corresponding resource FSM. For example, only one SSCP or the PUCP may have a connect-in pending for a given link, and only one SSCP or the PUCP may have a connect-out procedure in progress for a given link although both a connect-in and a connect-out may be active for a given link from the same CP (but not from different CPs). All further ACTCONNINs and CONNOUTs from other CPs receive a negative response until all active connect-in and/or connect-out procedures have been terminated. Also, only one IPL or DUMP procedure may be carried out at a given time.

With respect to ACTLINK, CONTACT, or DISCONTACT, only one link-level procedure at a time occurs, but more than one CP may be allowed to use the resource serially. At the completion of the link procedure, the PU.SVC_MGR.NS generates multiple responses if necessary. For example, three CPs issue ACTLINK: the PU.SVC_MGR.NS issues the ACTLINK to LINK_MGR only once; when the response from ACTLINK is received from LINK_MGR, the PU.SVC_MGR.NS generates three responses, one to each CP.

RESET HIERARCHY

The resource FSMs contained in the PU.SVC_MGR.NS lie in a reset hierarchy shown in Figure 11-4. For example, when PU.SVC_MGR.NS receives INOP(LINK_EA) from a LINK_MGR it resets the group of FSMs shown in the LINK_RESET procedure (Figure 11-4). The procedures that perform the reset are described on pages 11-94, 11-95, and 11-96.

LOST CONTROL POINT HIERARCHICAL RESET

The hierarchical reset necessary because of a lost control point is activated by the receipt of a DACTPU RU from CSC_MGR.RCV (Chapter 13). The hierarchical reset is performed by NS.LCP_RESET_PROC (Page 11-33).

During system definition, one of two lost control point reset options for resource FSMs is selected for the PU and for each link and adjacent link station. The NS.LCP_RESET_PROC uses these selections in resetting or not changing various resource FSMs during a lost control point hierarchical reset.

The first option is RESET, which implies that this resource is to be reset if only one CP is in the resource's CP list. The other option is CONTINUE, which implies that the resource is not to be reset. Choosing option RESET for the PU implies RESET for all links and adjacent link stations as well. Choosing option RESET for a link implies RESET for all its associated adjacent link stations as well.

If option RESET is chosen for the PU, link, or adjacent link station required by a cross-domain session, the session using the reset resource fails after failure of the SSCP. Cross-domain sessions are maintained after the SSCP has failed, only if the option CONTINUE is chosen for all resources required by the session.

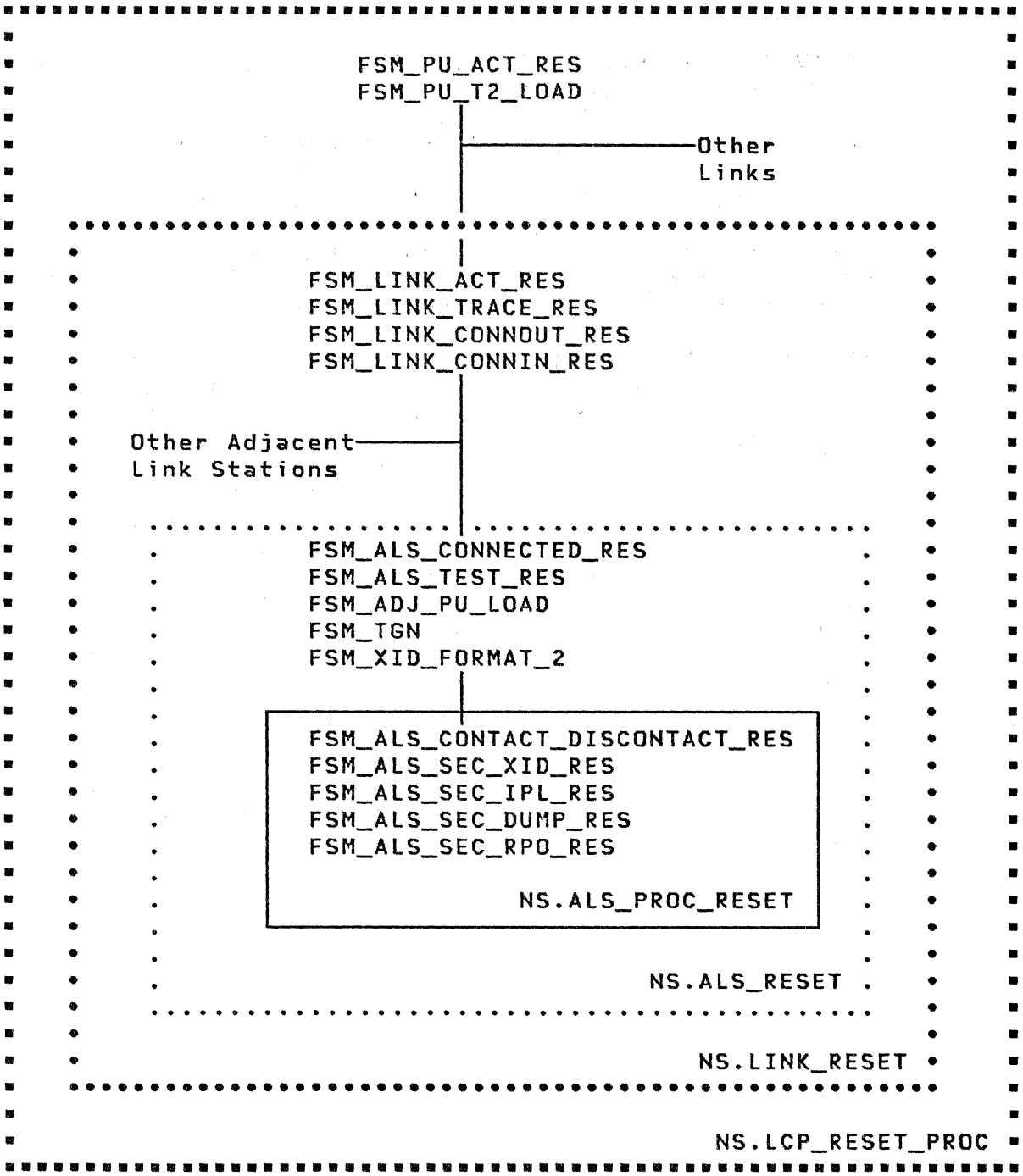


Figure 11-4. The Reset Hierarchy of Resource FSMs in a PU

PHYSICAL UNIT ACTIVATION

ACTIVATE PHYSICAL UNIT (ACTPU)
DEACTIVATE PHYSICAL UNIT (DACTPU)

Principal FSM: FSM_PU_ACT_RES (page 11-118)

Activating and deactivating a PU is different from activating and deactivating a session with the PU. The resource FSM, FSM_PU_ACT_RES, shows the state of the PU, while FSMs of the form, FSM_SESS_CP_PU_SEC (Chapter 13), show the state of a given session with the PU.

The PU.SVC_MGR.NS changes the state of FSM_PU_ACT_RES from RESET to ACTIVE when it receives a valid ACTPU. While FSM_PU_ACT_RES is ACTIVE any valid ACTPUs received are immediately positively responded to, provided the concurrency count does not exceed the share limit. No further change is made to FSM_PU_ACT_RES by an ACTPU while FSM_PU_ACT_RES is ACTIVE. If a DACTPU is received while FSM_PU_ACT_RES is ACTIVE, then PU.SVC_MGR.NS checks the CP list for the resource to see if any other CPs have sessions with the PU. If so, then the DACTPU receives a positive response, but no change is made to FSM_PU_ACT_RES. If not, PU.SVC_MGR.NS changes FSM_PU_ACT_RES from ACTIVE to RESET, along with sending a positive response to DACTPU.

When a valid ACTPU(ERP) is received, PU.SVC_MGR.NS decides whether to send a Cold or ERP response, based on the state of FSM_PU_ACT_RES. If that FSM is reset, +RSP(ACTPU,Cold) is returned; otherwise, +RSP(ACTPU,ERP) is returned. Whenever ACTPU(Cold) is requested, a +RSP(ACTPU,Cold) is returned.

LINK AND ADJACENT LINK STATION MANAGEMENT

The PU.SVC_MGR.NS performs some checking on configuration services requests and responses and also forwards them to the correct LINK_MGR. The PU.SVC_MGR.NS checks share limits and prevents conflicting requests for link procedures from being passed to the LINK_MGR at the same time. The LINK_MGR performs the actual activation, contact, or other procedure.

LINK ACTIVATION

ACTIVATE LINK (ACTLINK)
DEACTIVATE LINK (DACTLINK)

Principal FSM: FSM_LINK_ACT_RES (page 11-119)

For every link attaching to the node, the PU.SVC_MGR.NS contains a resource FSM, FSM_LINK_ACT_RES. The origin of the requests that cause state changes in this FSM differs depending on the primary or secondary SDLC characteristic of the link station and on the PU type of the node. For primary link stations, or secondary link stations in a PU_T4|5 node, ACTLINK and DACTLINK originate from either an SSCP or the local PUCP. For secondary stations in a PU_T1|2 node, ACTLINK and DACTLINK originate only from the PUCP local to the PU_T1 or PU_T2 node. (The PUCP may function such that ACTLINK is sent once, and DACTLINK is never sent. This leaves the FSM_LINK_ACT_RES in the active state after the PUCP has sent ACTLINK.)

At a given node, links and adjacent link stations, like the PU, are resources that may be under shared control. The PU.SVC_MGR.NS manages the actual activation and deactivation of the link similarly to the management of PU activation and deactivation, as described earlier. The state of the FSM_LINK_ACT_DOM_RES reflects a particular SSCP's view of the status of the link. A particular SSCP's view may differ from the state of the physical link.

THe PU.SVC_MGR.NS passes only one ACTLINK or DACTLINK at a time to LINK_MGR for a given link. Subsequent requests for the same procedure received from different CPs, while the original procedure is being processed by DLC, are discarded by the PU.SVC_MGR.NS (but are represented by an entry in the CP list). When LINK_MGR responds, the PU.SVC_MGR.NS forwards a response to each CP half-session that requested the function.

SWITCHED LINK CONNECTION

ACTIVATE CONNECT IN (ACTCONNIN)
DEACTIVATE CONNECT IN (DACTCONNIN)
CONNECT OUT (CONNOUT)
ABANDON CONNECT OUT (ABCONNOUT)
REQUEST CONTACT (REQCONT)
ABANDON CONNECTION (ABCONN)

Principal FSMs:

FSM_LINK_CONNIN_RES	(page 11-120)
FSM_LINK_CONNOUT_RES	(page 11-121)
FSM_ALS_SEC_XID_RES	(page 11-124)
FSM_ALS_CONNECTED_RES	(page 11-121)

For every switched link attachable to a node, the PU.SVC_MGR.NS contains three resource FSMs--FSM_ALS_CONNECTED_RES, FSM_LINK_CONNIN_RES, and FSM_LINK_CONNOUT_RES. For the secondary link station accessed via a switched link whose primary link station is in the node, there is an FSM_ALS_SEC_XID_RES in the PU.SVC_MGR.NS.

The FSM_LINK_CONNIN_RES, FSM_LINK_CONNOUT_RES, and FSM_ALS_CONNECTED_RES FSMs are functionally independent of the primary or secondary characteristic of the link station. Similarly to FSM_LINK_ACT_RES, the origins of the requests that affect these machines may be SSCPs or the PUCP for any primary link station or a secondary link station in a PU_T4|5 node, but can be only a PUCP for a secondary link station in a PU_T1|2 node.

The PU.SVC_MGR.NS allows only one CP at a time to have an active connect-in procedure on a given link or a connect-out procedure to a given adjacent link station. This means that the share limit for a switched link and its corresponding adjacent link station is always one and the PUCP is counted in the concurrency count. The PU.SVC_MGR.NS allows a connect-out procedure to be initiated even if the link has been enabled to accept incoming connections, and allows a connect-in procedure to be initiated even if the link has been connected.

The PU.SVC_MGR.NS exchanges XIDs via DLC independent of any CP, as part of a connect-in or connect-out procedure.

If a connect-out procedure or an XID exchange fails, a LINK_MGR generates an INOP(LINK_EA), which in turn causes a reset of both FSM_ALS_SEC_XID_RES and FSM_LINK_CONNOUT_RES, where LINK_EA denotes the link that failed. If a connect-in procedure fails, the LINK_MGR generates an INOP(LINK_EA), which in turn causes a reset of FSM_LINK_CONNIN_RES. Successful completion of a connect-in or connect-out

procedure followed by successful completion of the XID exchange results in REQCONT being sent to the CP that originated the ACTCONNIN or CONNOUT. If a connect-in and a connect-out procedure are simultaneously active, the assumption is made that the connect-out procedure was the one that was successful.

STATION CONTACTING

CONTACT
DISCONTACT
CONTACTED

Principal FSM:
FSM_ALS_CONTACT_DISCONTACT_RES (page 11-122)

For every adjacent link station represented in the node resource list, the PU.SVC_MGR.NS contains one resource FSM, FSM_ALS_CONTACT_DISCONTACT_RES. The requests that are routed to FSM_ALS_CONTACT_DISCONTACT_RES originate from an SSCP or from the PUCP. In a PU_T1 or a PU_T2 these requests originate only from the PUCP.

The PU.SVC_MGR.NS prohibits initiation of a CONTACT or DISCONTACT procedure if some uninterruptible procedure is being processed by DLC.

For adjacent link stations the PU.SVC_MGR.NS performs such functions as: allowing CONTACTs from multiple CPs and later returning multiple CONTACTED requests, and commanding LINK_MGR to discontact only when the adjacent link station is no longer being shared.

Configurable Link Stations

Configurable link stations are capable of supporting both primary station and secondary station protocols. During the link-level contact procedure for these stations there is an exchange of format 2 Exchange Identification (XID) link commands and responses. (See Appendix E for the description of the format 2 XID.) This exchange allows the negotiation of certain characteristics of the pairing of the two stations, among them the choice of primary and secondary station.

Configurable stations are in PU_T4 and PU_T5 nodes and are used only for nonswitched link connections between PU_T4 and PU_T5 nodes. The connection may be by SDLC link or S/370 channel link. The active link becomes part of a transmission group.

ADJACENT LINK STATION LOADING, DUMPING, AND POWER-OFF

IPL INITIAL (IPLINIT)
IPL TEXT (IPLTEXT)
IPL FINAL (IPLFINAL)
DUMP INITIAL (DUMPINIT)
DUMP TEXT (DUMPTEXT)
DUMP FINAL (DUMPFINAL)
REMOTE POWER OFF (RPO)

Principal FSMs:

- FSM_ALS_SEC_DUMP_RES (page 11-122)
- FSM_ALS_SEC_IPL_RES (page 11-123)
- FSM_ALS_SEC_RPO_RES (page 11-123)

When a PU.SVC_MGR.NS receives IPLINIT or DUMPINIT for a secondary adjacent link station, it checks to see if there is any non-interruptible link procedure occurring at the link level. If so, the request is rejected with a negative response indicating Link Procedure in Process. If not, PU.SVC_MGR.NS starts the new procedure by sending the request to LINK_MGR and resets all other procedures by resetting the resource FSMs.

When PU.SVC_MGR.NS receives RPO it returns a negative response indicating RPO not Initiated, if the FSMs relating to CONTACT, DISCONTACT, IPL, or DUMP are not reset.

INOPERATIVE LINKS AND ADJACENT LINK STATIONS

INOPERATIVE (INOP)

Principal FSM: None

DLC.MGR detects inoperative conditions and determines whether they are attributable to an adjacent link station failure or to a link failure (link connection failure or link failure). These conditions are reported to PU.SVC_MGR.NS by LINK_MGR sending INOP. The processing of this request by PU.SVC_MGR.NS consists of resetting the resource FSMs by ALS_RESET (page 11-95) or LINK_RESET (page 11-94) and passing the INOP on to the appropriate half-session. This forwarding of INOP sometimes requires multiple INOPs for several half-sessions to be generated from a single INOP received from LINK_MGR.

This page intentionally left blank

LOADING A PU_T2 NODE

There are two ways that a PU_T2 can request a load module be moved to its node. One way is for the PU_T2 to send +RSP(ACTPU, IPL Required). The second way is for the PU_T2 to send LDREQD to the SSCP.

When the PU_T2 requests a load operation, it sets the Adjacent PU Load Capability bit to NOT_CAPABLE. This bit is contained in both the ACTPU response (control vector X'07') and LDREQD.

If the PU_T2 requests a load operation and the subarea PU adjacent to the PU_T2 node is able to perform the load operation, the BF.PU.SVC_MGR in the subarea PU sets the Adjacent PU Load Capability bit to CAPABLE. If the subarea PU is not able to load the PU_T2 node, the Adjacent PU Load Capability bit remains set to NOT_CAPABLE.

Upon receipt of LDREQD or +RSP(ACTPU, IPL required), the SSCP inspects the Adjacent PU Load Capability bit. If the bit is set to CAPABLE, the SSCP sends INITPROC to the subarea PU. The sending of INITPROC (see Chapter 7) directs the subarea PU to perform a PU_T4|5-PU_T2 load operation. If the Adjacent PU Load Capability bit is set to NOT_CAPABLE, the SSCP attempts to perform an SSCP-PU_T2 load operation.

PU_T4|5-PU_T2 LOAD OPERATION

```
NC IPL INITIAL (NC_IPL_INIT)
NC IPL TEXT    (NC_IPL_TEXT)
NC IPL FINAL   (NC_IPL_FINAL)
NC IPL ABORT   (NC_IPL_ABORT)
```

Principle FSMs:

FSM_ADJ_PU_LOAD	(page 11-124)
FSM_PU_T2_LOAD	(page 11-118)

A PU_T4|5-PU_T2 load operation is initiated by the subarea PU upon receipt of an INITPROC from the SSCP (see Chapter 7). When the subarea PU receives the INITPROC, certain validity checks are performed (e.g., whether the PU_T2 node requesting load is a loadable resource). If the subarea PU determines it is valid to load the PU_T2 node, it sends a +RSP(INITPROC) to the SSCP. If it is not valid for the subarea PU to load the PU_T2 node, a -RSP(INITPROC) (with the appropriate sense code) is sent to the SSCP. When the SSCP receives the negative response, it attempts to load the PU_T2 node itself.

If the subarea PU responds positively to the INITPROC, the subarea PU sends NC_IPL_INIT to the PU_T2 to begin the load

operation. When the PU_T2 receives the NC_IPL_INIT, the PU_T2 performs some validity checks to determine whether it can accept the load module from the subarea PU. If the PU_T2 can process the PU_T4|5-PU_T2 load operation, the PU_T2 sends a +RSP(NC_IPL_INIT) to the subarea PU. If the PU_T2 cannot process the PU_T4|5-PU_T2 load operation, the PU_T2 node sends a -RSP(NC_IPL_INIT) to the subarea PU.

Upon receipt of the response to NC_IPL_INIT from the PU_T2, the subarea PU starts the transmission of the load module via NC_IPL_TEXT requests. Upon receipt of an NC_IPL_TEXT request, the PU_T2 performs some validity checks (e.g., whether the PU_T2 is in an appropriate state to accept the load module). If no error is detected, the PU_T2 sends a positive response to the subarea PU. The subarea PU continues to send NC_IPL_TEXT requests until the transfer of the load module has been completed. The subarea PU requires a positive response to the outstanding NC_IPL_TEXT request before the next request may be sent.

When the subarea PU receives the response to the final NC_IPL_TEXT, the subarea PU sends NC_IPL_FINAL to the PU_T2. When the PU_T2 receives the NC_IPL_FINAL, certain validity checks are performed (e.g., the entry point location is checked). If the load module has been successfully transferred, the PU_T2 sends a +RSP(NC_IPL_FINAL) to the subarea PU.

Upon receipt of the positive response to NC_IPL_FINAL, the subarea PU sends PROCSTAT(IPL Successful) to the SSCP. The sending of PROCSTAT resets the ADJ_PU_LOAD FSM.

If, at any time during the load operation, the subarea PU receives a negative response from the PU_T2, the subarea PU sends NC_IPL_ABORT to the PU_T2. NC_IPL_ABORT is also sent when the subarea PU is not able to complete the load operation. NC_IPL_ABORT contains sense data indicating the reason for the failure. After the subarea PU sends NC_IPL_ABORT to the PU_T2, the subarea PU sends PROCSTAT(Procedure Failure) to the SSCP. This also contains sense data, which relays the cause of the failure to the SSCP. When the SSCP receives PROCSTAT(Procedure Failure) it attempts to load the PU_T2 node itself.

When the PU_T2 receives an NC_IPL_ABORT from the subarea PU, the PU_T2 positively responds. At this point, if the load operation was requested via response to ACTPU, the SSCP sends DACTPU to the PU_T2. If the load operation was requested via LDREQD, the PU_T2 may request another load or may send REQDISCONT.

SSCP-PU_T2 LOAD OPERATION

NS IPL INITIAL (NS_IPL_INIT)
NS IPL TEXT (NS_IPL_TEXT)
NS IPL FINAL (NS_IPL_FINAL)
NS IPL ABORT (NS_IPL_ABORT)

Principle FSM: None (see Chapter 7 for FSMs)

The SSCP attempts to perform an SSCP-PU_T2 load operation when it receives LDREQD(Adjacent PU Load Capability = NOT_CAPABLE) or +RSP(ACTPU, IPL required, Adjacent PU Load Capability = NOT_CAPABLE). The SSCP also attempts to perform an SSCP-PU_T2 load operation upon receipt of a -RSP(INITPROC) or PROCSTAT(Procedure Failure) from the subarea PU.

The SSCP-PU_T2 load operation is performed the same as the PU_T4|5-PU_T2 load operation with the following exceptions.

- The NC_IPL_INIT, NC_IPL_TEXT, NC_IPL_FINAL, and NC_IPL_ABORT RUs are replaced by NS_IPL_INIT, NS_IPL_TEXT, NS_IPL_FINAL, and NS_IPL_ABORT, respectively. (See Chapter 7 for a description of the NS_IPL RUs.)

- The SSCP-PU_T2 load requests flow on the SSCP-PU_T2 session.

- If the SSCP sends NS_IPL_ABORT and the load operation was requested via the response to ACTPU, the SSCP also sends DACTPU to the PU_T2.

- INITPROC and PROCSTAT are not sent for an SSCP-PU_T2 load operation.

CONFIGURATION NETWORK MANAGEMENT (CNM)

The PU.SVC_MGR.NS performs protocol checking on maintenance requests and responses. The PU.SVC_MGR.NS performs checks and prevents conflicting requests from being initiated. The actual function is performed by other components of the node.

LINK AND TG TRACE

ACTIVATE TRACE (ACTTRACE)
DEACTIVATE TRACE (DACTTRACE)
RECORD TRACE DATA (RECTRD)

Principal FSM: FSM_LINK_TRACE_RES (page 11-120)

An SSCP may request a link to provide trace data for maintenance purposes. If the link is part of a transmission group (TG) a request for a PIU trace for the entire TG may also be requested concurrently with the link level trace. The TG option is directly associated with a particular link. No other links in the same TG may be traced with the TG option if the TG option is already active for the transmission group.

In order for an SSCP to initiate a trace, it must have first successfully activated the link with an ACTLINK request. In addition the link may not be shared with another CP, nor may it already have a trace function active.

LINK LEVEL 1 DIAGNOSTIC TESTING

EXECUTE TEST (EXECTEST)
RECORD TEST DATA (RECTD)

Principal FSM: FSM_LINK_ACT_RES (page 11-119)

An SSCP may start diagnostic tests against the entire link. This class of diagnostics require that the SSCP be the only entry in the resource's CP list. No other procedure may be started on the link if a test is currently in progress. The test request EXECTEST is passed directly to LINK_MGR. LINK_MGR formats the results of the test into one or more RECTD RUs. These are passed to the CP in the resource's CP list.

LINK LEVEL 2 DIAGNOSTIC TESTING

TEST MODE (TESTMODE)
RECORD TEST RESULTS (RECTR)
REQUEST TEST PROCEDURE (REQTEST)

Principal FSM: FSM_ALS_TEST_RES (page 11-124)

An SSCP may start diagnostic tests against the link for a specific adjacent link station. These tests can be run concurrent with other activity on the link, however the SSCP must be the only entry in the adjacent link station's CP list. No other procedure may be active against the adjacent link station. The test function may be requested (REQTEST), however the PU.SVC_MGR.NS does not maintain any record of this request. The TESTMODE RU is passed to LINK_MGR if it does not violate protocol. LINK_MGR formats the test results (RECTR) and returns them to the the SSCP contained in the resource's CP list.

DISPLAY STORAGE

DISPLAY STORAGE (DISPSTOR)
RECORD STORAGE (RECSTOR)

Principal FSM: None

A request to return the local storage for a PU is handled by a UPM which builds the responses and the actual storage requested in one or more RECSTOR RUs. These RUs may flow at any time. The control point must have an active session with the PU at the time they flow.

MAINTENANCE STATISTICS

REQUEST MAINTENANCE STATISTICS (REQMS)
RECORD MAINTENANCE STATISTICS (RECMS)
RECORD FORMATTED MAINTENANCE STATISTICS (RECFMS)

Principal FSM: None

A request for maintenance statistics is handled by a UPM. This UPM builds the responses to the request and sends reply requests (either RECMS or RECFMS). The UPM also may originate these requests without a request from any SSCP. These RUs may flow at any time. The control point must have an active session with the PU at the time they flow.

DYNAMIC ASSIGNMENT OF NETWORK ADDRESSES BY PU

REQUEST NETWORK ADDRESS ASSIGNMENT (RNAA)

Principal FSM: None

An RNAA request may be issued by an SSCP to request the PU to assign network addresses for specific BF.PUs, BF.LUs, or LUs and to create corresponding entries in the node resource list. If network addresses have already been assigned for the specific BF.PUs or BF.LUs, a negative response indicating Function Active is returned. If the LU address exists, a negative response indicating Function Active is returned.

If the RNAA is for one or more BF.PU network addresses and storage resources are available for creating the node resource list entries, each BF.PU network address is assigned and the element address is entered into a newly created node resource list entry.

BF.LU network addresses can be added for PU_T1|2 nodes that have been added to, or moved within, the network. Also, BF.LU network addresses can be added by an implementation defined procedure for statically defined PU_T1|2 nodes. Additional decisions must be made for an RNAA requesting BF.LU network addresses. For PU_T1|2 nodes that have been added to, or moved within, the network, BF.LU network addresses can be assigned by an SSCP only after it assigns the BF.PU network address. For PU_T1|2 nodes that have been statically defined in the network, BF.LU network addresses can be assigned by an SSCP only after it activates the SSCP-PU_T1|2 session. If these checks are successfully passed and storage resources are available, each BF.LU network address is assigned and the element address is entered into a newly created node resource list entry.

When a BF.PU or BF.LU entry in the node resource list is created, the resource category, the element address of the associated resource, the local form of address, and the network address of the SSCP that sent the RNAA are entered into it. The remainder of the BF.PU or BF.LU parameters are specified by a subsequent SETCV.

An LU address can be assigned to an existing LU to provide parallel session capabilities (see Chapter 1). When the resource entity is created, the associated address is set to the LU element address passed in bytes 3 and 4 of the RNAA request.

FREEING OF NETWORK ADDRESSES

FREE NETWORK ADDRESSES (FNA)

Principal FSM: None

The algorithm for freeing BF.PU and BF.LU network addresses is defined in NS.FNA_PROC (page 11-55). Addresses are freed by discarding their associated entries from the node resource list.

A BF.PU (and ALS) network address cannot be freed unless the associated ALS_SEC_SUBTREE is reset, all associated BF.LU network addresses have been freed, and its current session count is 0. A BF.LU network address cannot be freed unless its current session count is 0. If one or more of the network addresses contained in FNA cannot be freed, none of the addresses is freed.

SET CONTROL VECTOR PROCESSING

The algorithm for SET CONTROL VECTOR processing is defined in NS.SETCV_PROC (page 11-64). If the SETCV control vector key value is valid, then additional key-specific checks are made. A SETCV with vector key = X'03' (SPU) is rejected if the appropriate ALS_SEC_SUBTREE is not reset, and a SETCV with vector key = X'04' (LU) is rejected if the current session count for the specified BF.LU is not 0. If all the checks are passed, the parameters in the control vector are entered into the appropriate data structure.

REQUESTING LU ACTIVATION

REQUEST ACTIVATE LOGICAL UNIT(REQACTLU)

Principal FSM: None

NS.REQACTLU_PROC (page 11-93) receives a REQACTLU from a UPM. The request contains the network name of the LU that is to be activated. A check is made to see if there are sufficient resources available (addresses, buffers, control blocks, etc.) to assign a network address to the LU. If so, the network address for the LU that is to be activated is obtained and placed in the RU. The node resource list is updated to reflect the addition of the LU to the node. The PU.SVC_MGR.NS then sends the RU to the CP.

REQUESTING THE FREEING OF A NETWORK ADDRESS

REQUEST FREE NETWORK ADDRESSES (REQFNA)

Principal FSM: None

NS.REQFNA_PROC (page 11-92) receives the REQFNA from a UPM. The request contains the network address of the LU to be freed and the type of termination--Normal, Orderly, Forced, or Cleanup. PU.SVC_MGR.NS checks the node resource list to verify the network address. If the network address of the LU is not found, or the network address is not an LU, a -RSP is sent to the calling UPM; otherwise, the request is sent to the SSCP.

NODE DATA BASE STRUCTURE

The node data base is structured as shown in Figure 11-5. This structure describes the logical hierarchy of the many resources within a node. Details of this structure are given in Appendix A.

The node control block (NCB) contains the element address of the PU, which permits locating the node resource entry for the PU. The node resource entry for the PU is a logical extension of the node control block. For the PU it contains the lost control point reset option and the share limit. A CP list is maintained for the PU listing all CPs that have successfully issued ACTPU. The CPCB list entry is deleted when a DACTPU is received or when a session with a control point is lost.

Each physical link attached to the PU is hierarchically associated with the PU. Since there is only one PU, it is not necessary to specify the relationship explicitly in the node resource entry for a link. The lost control point reset option, DLC role, switched or nonswitched, and share limit is contained in the resource entry. In addition, a list of CPs is associated with each link. An entry is made on the list when a CP successfully issues ACTLINK. The association is removed when the CP issues DACTLINK or when and INOP is generated.

Each link may attach to one or more adjacent link stations associated with it. The node maintains a representation of the status of the external adjacent link stations. The element address for an adjacent link station is unique to the particular node in which it is represented. For switched connections there is

one adjacent link station for the link to represent all possible users of the link. At a node containing a secondary link station, there is also only one adjacent link station (the primary station). At a node contining the primary link station for a multipoint link, there may be many adjacent link stations. Each resource entry for an adjacent link station points to the specific link to which it is associated. In addition, information about its lost control point reset option, DLC role, SDLC link address, share limit, and maximum BTU size is contained in the resource list. A list of CPs is maintained for the adjacent link station. An entry is added to this list when a CP successfully issues CONTACT and is removed when a CP issues DISCONTACT or when an INOP is generated.

Periperal nodes associated with adjacent link stations require boundary function support in a subarea node. A node resource entry is maintained for BF.PU. The element address used to represent the peripheral PU is the same element address used to represent the adjacent link station. The entry contains the BF local ID, PU type, and maintenance services profile for the BF.PU. In addition, if the address has been dynamically created by RNAA, the entry also contains the CP address that assigned the address for the BF.PU.

A boundary function LU (BF.LU) is required for every peripheral LU. A resource entry for a BF.LU contains the BF.PU address it is associated with, the local ID, and pacing count. The resource entry also contains a CP address, if assigned dynamically with RNAA.

Logical units within the node are represented by an entry for the LU. LUs that do not support parallel sessions are represented by a single entry. LUs that do support parallel sessions are represented by a single secondary LU address and multiple primary addresses. The primary LU resource entries contain their secondary LU element address as their associated resource.

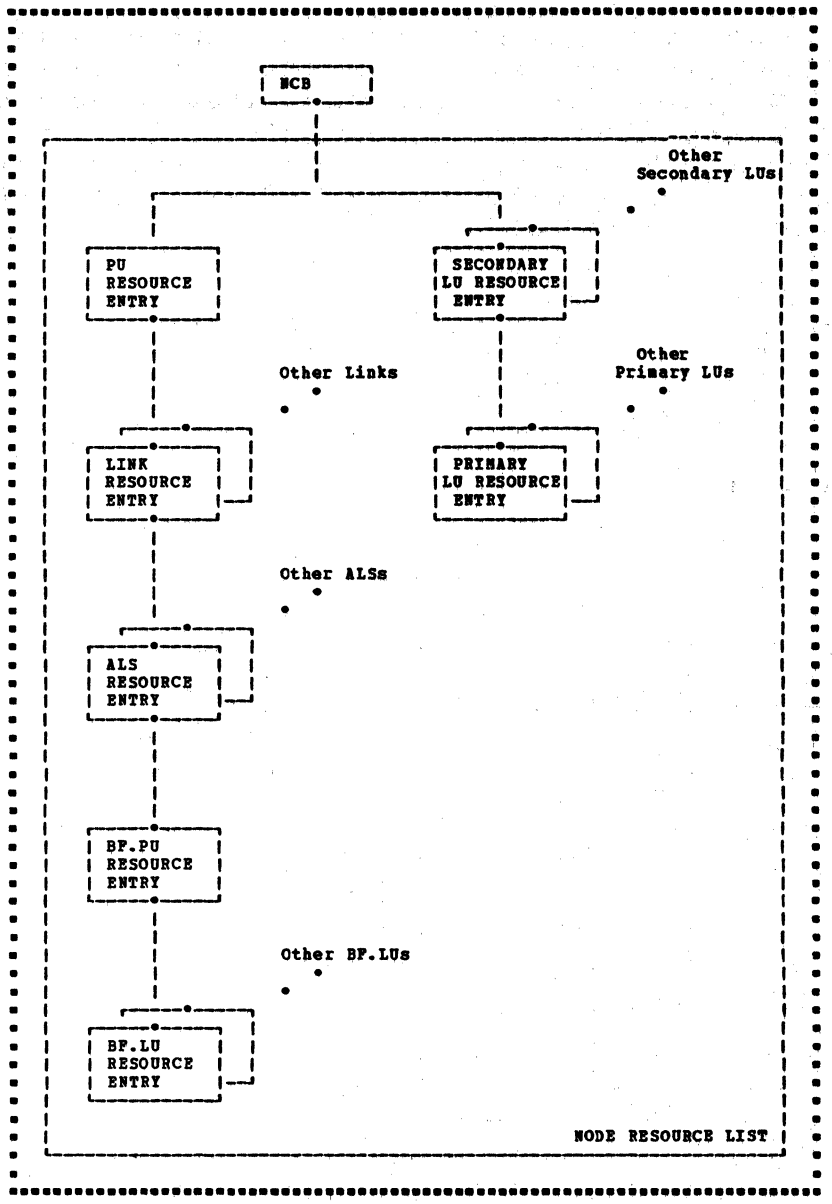


Figure 11-5. Structure of the Node Resource Data Base.

Each CP that activates the PU becomes an entry in the CPCB list and is removed when the session is deactivated. As that control point acquires resources the corresponding CPCB is associated with the particular resource by adding a CP_INDIRECT entry that point to the particular CPCB to the CP_INDIRECT list maintained for that resource.

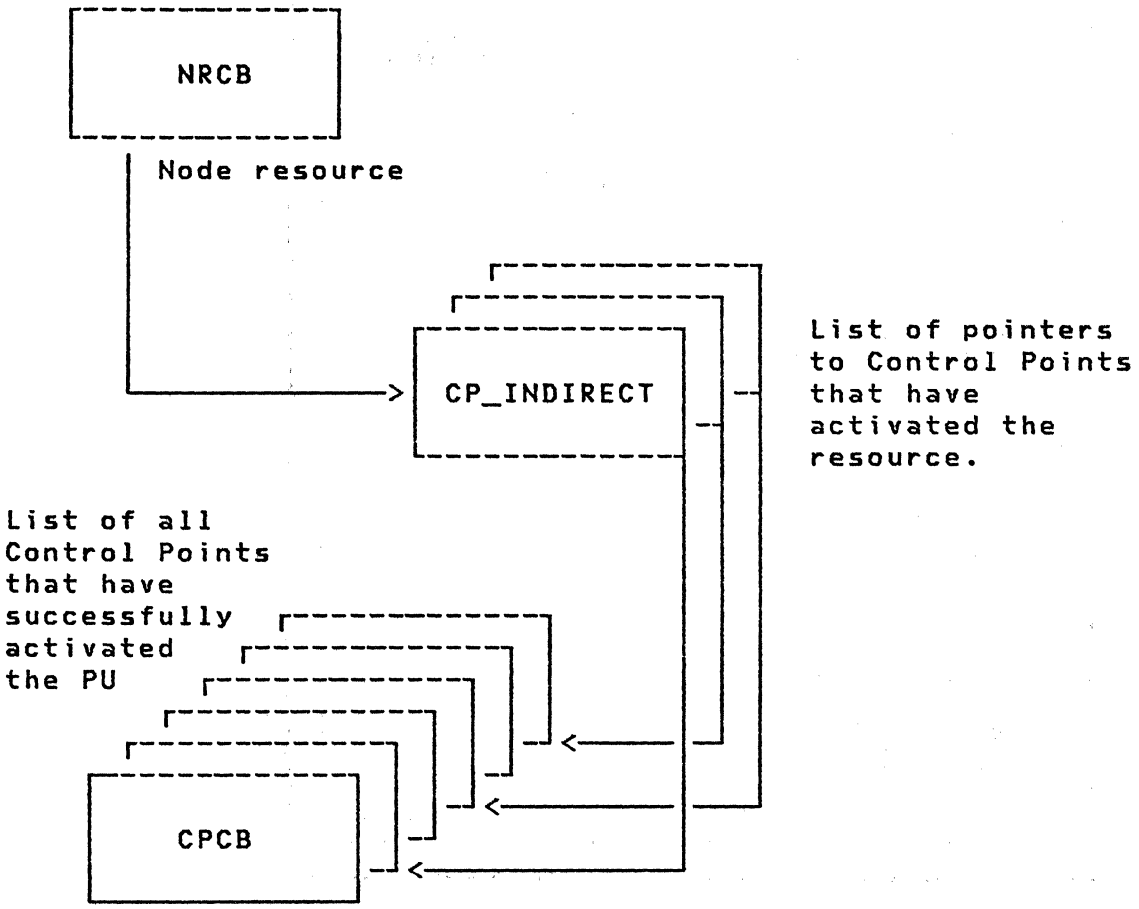


Figure 11-6. Relation of Node Resources to Control Points.

```
PU.SVC_MGR.NS.RCV: PROCEDURE;
                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  RECEIVES ALL INPUT TO THE PU.SVC_MGR.NS  AND ROUTES THE INPUT TO THE
│                 APPROPRIATE ROUTINE FOR FURTHER PROCESSING.   THE ROUTING THAT TAKES
│                 PLACE IS DETERMINED  BY THE PROCEDURE THAT SENDS THE  CURRENT PIU OR
│                 SIGNAL, AND THE CONTENT OF THE PIU OR SIGNAL.
│
│      INPUT:     THE CURRENT PIU OR A SIGNAL
│
│      OUTPUT:    REFER TO THE PROCEDURES THAT ARE  CALLED FROM THIS PROCEDURE FOR THE
│                 SPECIFIC OUTPUTS.
│
│      REFERS TO THE FOLLOWING PROCEDURE(S):
│                 ADJ_PU_LOAD_PROC                  PAGE 11-102
│                 FSM_CP_SESS_SDT                   PAGE 11-119
│                 FSM_XID_FORMAT_2                  PAGE 11-126
│                 NS.CS_RCV                         PAGE 11-34
│                 NS.DLC_CONFIG                     PAGE 11-66
│                 NS.DLC_RCV                        PAGE 11-76
│                 NS.MS_PROC                        PAGE 11-107
│                 NS.SC_PROC                        PAGE 11-30
│                 PU_T2_LOAD_PROC                   PAGE 11-100
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  SELECT ANYORDER;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │  WHEN THIS PROCEDURE IS  DISPATCHED BY SNS.RCV  │
                  │  OR  BY THE  PU  SERVICE  MANAGER ITSELF,  THE  │
                  │  SECOND BYTE OF  THE NS HEADER IS  CHECKED FOR  │
                  │  THE TYPE OF SERVICE REQUIRED.           │
                  └──────────────────────────────────────────┘
                                                                        */
  . WHEN(DISPATCHED_BY(SNS.RCV) |                    /* CHAPTER 6          */
  .      DISPATCHED_BY(PU.SVC_MGR.NS.RCV))           /* RETRY QUEUED REQUESTS */
  .    DO;
  .    . SELECT ANYORDER;
  .    . . WHEN(NS_CATEGORY(1:7) = CONFIGURATION_SERVICES)   /* BITS 1-7      */
  .    . .   CALL NS.CS_RCV;                                 /* PAGE 11-34    */
  .    . .
  .    . . WHEN(NS_CATEGORY(1:7) = MAINTENANCE_SERVICES)     /* BITS 1-7      */
  .    . .   CALL NS.MS_PROC;                                /* PAGE 11-107   */
  .    . END;
  .    END;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │  WHEN   THIS   PROCEDURE  IS   DISPATCHED   BY  │
                  │  UPM_CNMS, THE SECOND BYTE OF THE NS HEADER IS  │
                  │  CHECKED FOR THE TYPE OF SERVICE REQUIRED.      │
                  └──────────────────────────────────────────┘
                                                                        */
  . WHEN(DISPATCHED_BY(UPM_CNMS))
  .    DO;
  .    . FIND CPCB IN CPCB_LIST WHERE(CPCB.CP_SCB_ID = SCB_PTR);
  .    . IF CPCB_PTR ¬= NULL &
  .    .      FSM_CP_SESS_SDT = ACTIVE &                    /* PAGE 11-119   */
  .    .      NS_CATEGORY(1:7) = MAINTENANCE_SERVICES THEN  /* BITS 1-7      */
  .    .   CALL NS.MS_PROC;                                 /* PAGE 11-107   */
  .    . ELSE
  .    .   SEND SEND_CHECK TO SENDING_PROCEDURE;
  .    END;
```

```
                                                                        /*
                    ┌──────────────────────────────────────────────┐
                    │ WHEN THIS PROCEDURE IS DISPATCHED BY LINK_MGR │
                    │ THE NRCB ENTRY OF THE LINK OR ADJACENT LINK  │
                    │ STATION IS FOUND.  IF THE NRCB ENTRY IS FOR  │
                    │ AN ADJACENT LINK STATION, THEN THE LINK      │
                    │ ADDRESS IS CONTAINED IN THE ADJACENT LINK    │
                    │ STATION ENTRY.                               │
                    └──────────────────────────────────────────────┘
                                                                        */
  . WHEN(DISPATCHED_BY(PU.SVC_MGR.LINK_MGR))
  .   DO;
  .   . NRCB_PTR = LOCATE_NODE_RESOURCE(LSCB.EA);          /* APPENDIX B   */
  .   . IF NRCB.RESOURCE_CATEGORY = ALS THEN
  .   .   NRCB_PTR = LOCATE_NODE_RESOURCE(NRCB.ASSOCIATED_RESOURCE);
  .   .                                                    /* APPENDIX B   */
  .   . IF NRCB.PRI_SEC_ROLE = CONFIGURABLE &
  .   .      FSM_XID_FORMAT_2 ¬= ACTIVE THEN               /* PAGE 11-126  */
  .   .   CALL NS.DLC_CONFIG;                             /* PAGE 11-66   */
  .   .
  .   . ELSE
  .   .   CALL NS.DLC_RCV;                                /* PAGE 11-76   */
  .   END;
                                                                        /*
                    ┌──────────────────────────────────────────────┐
                    │ THIS PROCEDURE IS DISPATCHED BY COMMON       │
                    │ SESSION CONTROL FOR DACTPU AND ACTPU TO THIS │
                    │ NODE OR BY TC.SC FOR SDT.                    │
                    └──────────────────────────────────────────────┘
                                                                        */
  . WHEN(DISPATCHED_BY(PU.SVC_MGR.CSC_MGR.RCV) |           /* CHAPTER 13    */
  .      DISPATCHED_BY(TC.SC.RCV))                         /* CHAPTER 4    */
  .   CALL NS.SC_PROC;                                    /* PAGE 11-30   */
                                                                        /*
                    ┌──────────────────────────────────────────────┐
                    │     RESPONSES TO NC_IPL REQUESTS FROM PU_T2   │
                    └──────────────────────────────────────────────┘
                                                                        */
  . WHEN(DISPATCHED_BY(BF.PC))                             /* CHAPTER 3    */
  .   CALL ADJ_PU_LOAD_PROC;                               /* PAGE 11-102  */
                                                                        /*
                    ┌──────────────────────────────────────────────┐
                    │ NC_IPL REQUESTS FROM THE ADJACENT SUBAREA PU  │
                    └──────────────────────────────────────────────┘
                                                                        */
  . WHEN(DISPATCHED_BY(PC.T2.RCV))                         /* CHAPTER 3    */
  .   CALL PU_T2_LOAD_PROC;                                /* PAGE 11-100  */
  END;
  RETURN;
END PU.SVC_MGR.NS.RCV;
```

```
NS.SC_PROC: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  UPON RECEIPT OF  ACTPU, A POSITIVE OR NEGATIVE  RESPONSE IS CREATED.  │
│                 IF THE PU_ACT_RES FSM  IS RESET WHEN A VALID ACTPU  IS RECEIVED, ITS  │
│                 STATE  IS  CHANGED TO  ACTIVE.   UPON  INPUT  OF DACTPU  A  POSITIVE  │
│                 RESPONSE IS  CREATED.  IF THE  CP IS THE  ONLY CP IN  THE RESOURCE'S  │
│                 CPCB_LIST, THEN THE PU RESOURCE FSM  IS RESET.  WHEN SDT IS RECEIVED  │
│                 AND THE PU IS A SUBAREA NODE,  THE START DATA TRAFFIC FSM IS UPDATED  │
│                 TO ALLOW  REQUESTS TO  FLOW TO  THE CP.   THIS FSM  IS AUTOMATICALLY  │
│                 RESET WHENEVER THE CPCB IS DESTROYED.                                 │
│                                                                                       │
│      INPUT:     ACTPU OR DACTPU FROM PU.SVC_MGR.CSC_MGR.RCV AND SDT FROM TC.SC.RCV.   │
│                                                                                       │
│      OUTPUT:    POSITIVE OR NEGATIVE RESPONSE TO  ACTPU; POSITIVE RESPONSE TO DACTPU  │
│                 TO  PU.SVC_MGR.CSC_MGR.SEND;   OR  POSITIVE   RESPONSE  TO   SDT  TO  │
│                 TC.SC.SEND.                                                           │
│                                                                                       │
│      NOTE:      THE PROTOCOL BOUNDARY  THAT IS MAINTAINED BETWEEN  THE PU.SVC_MGR.NS   │
│                 AND PU.SVC_MGR.CSC_MGR AND TC.SC IS AS FOLLOWS:                       │
│                 1.  TH INFORMATION:                                                   │
│                     • SESSION IDENTIFICATION                                          │
│                 2.  RH INFORMATION:                                                   │
│                     • REQUEST/RESPONSE INDICATOR                                      │
│                     • SENSE DATA INCLUDED INDICATOR                                   │
│                 3.  RU INFORMATION (THE RU INFORMATION IS  IN THE FORMAT IN APPENDIX   │
│                     E).                                                               │
│                                                                                       │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                        │
│                 PU.SVC_MGR.NS.RCV                  PAGE 11-28                         │
│                                                                                       │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                                            │
│                 FSM_CP_SESS_SDT                    PAGE 11-119                        │
│                 FSM_PU_ACT_RES                     PAGE 11-118                        │
│                 NS.LCP_RESET_PROC                  PAGE 11-33                         │
│                 UPM_ACTPU_CPID_CHECK               PAGE 11-112                        │
│                 UPM_EXTRACT_NS_LSA_RQD             PAGE 11-114                        │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL RETURN_CODE BIT(1);
DCL CP_ENTRY_LIST PTR;
DCL SSCP_SCB_ID PTR;

SSCP_SCB_ID = SCB_PTR;
NRCB_PTR = LOCATE_NODE_RESOURCE(NCB.PU_EA);                  /* APPENDIX B          */

SELECT ANYORDER(RQ_CODE);
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │                DACTPU                    │
                  └──────────────────────────────────────────┘
                                                                        */
. WHEN(DACTPU)
.   DO;
.   . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);                   /* APPENDIX B          */
.   . SEND MU TO PU.SVC_MGR.CSC_MGR.SEND;                    /* CHAPTER 13          */
.   . CALL NS.LCP_RESET_PROC(SSCP_SCB_ID);                   /* PAGE 11-33          */
.   . IF NCB.PU_TYPE = (PU_T4 | PU_T5) THEN
.   .   SCAN CPCB_LIST PTR(CPCB_PTR);
.   .   . IF CPCB.ER_VR_SUPP = PRE_ER_VR THEN
.   .   .   MU_PTR = UPM_CREATE_RQ('ANSC');                  /* APPENDIX B          */
.   .   . ELSE
.   .   .   MU_PTR = UPM_CREATE_RQ('LCP');                   /* APPENDIX B          */
.   .   . SCB_PTR = CPCB.CP_SCB_ID;
.   .   . SEND MU TO PU.SVC_MGR.CSC_MGR.SEND;                /* CHAPTER 13          */
.   .   SCANEND;
.   END;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │                 SDT                      │
                  └──────────────────────────────────────────┘
                                                                        */
. WHEN(SDT)                                                  /* PU TYPES 4 AND 5 ONLY */
.   DO;                                                      /* TS PROFILE 5          */
.   . FIND CPCB IN CPCB_LIST WHERE(SSCP_SCB_ID = CPCB.CP_SCB_ID); /* SESSION ALWAYS EXISTS */
.   . CALL FSM_CP_SESS_SDT;                                  /* PAGE 11-119         */
.   . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);                   /* APPENDIX B          */
.   . SEND MU TO TC.SC.SEND;                                 /* CHAPTER 4           */
.   END;
```

```
                                                                                /*
                          ACTPU
                                                                                */
   . WHEN(ACTPU)
   .   SELECT ANYORDER;
   .   . WHEN(FSM_PU_ACT_RES = RESET)                          /* PAGE 11-118    */
   .   .   DO;
   .   .   . IF UPM_ACTPU_CPID_CHECK = OK THEN                 /* PAGE 11-112    */
   .   .   .   DO;
   .   .   .   . CREATE CPCB;
   .   .   .   . CPCB.CP_SCB_ID = SSCP_SCB_ID;
   .   .   .   . INSERT CPCB IN CPCB_LIST;
   .   .   .   . IF PID = PID4 THEN
   .   .   .   .   CPCB.ER_VR_SUPP =  ¬PRE_ER_VR;
   .   .   .   . ELSE
   .   .   .   .   CPCB.ER_VR_SUPP = PRE_ER_VR;
   .   .   .   . IF NCB.PU_TYPE = (PU_T4 | PU_T5) THEN
   .   .   .   .   CPCB.NS_LSA_RQD = UPM_EXTRACT_NS_LSA_RQD;  /* PAGE 11-114    */
   .   .   .   . IF NCB.PU_TYPE = (PU_T1 | PU_T2) THEN
   .   .   .   .   CALL FSM_CP_SESS_SDT('ACTIVE');            /* PAGE 11-119    */
   .   .   .   . CALL ADD_CP_ENTRY(NCB.PU_EA,SSCP_SCB_ID);    /* APPENDIX B     */
   .   .   .   . CALL FSM_PU_ACT_RES;                         /* PAGE 11-118    */
   .   .   .   . MU_PTR = UPM_CREATE_RSP('ACTPU');            /* APPENDIX B     */
   .   .   .   END;
   .   .   . ELSE
   .   .   .   CALL CHANGE_MU_TO_NEG_RSP(X'081D');            /* APPENDIX B     */
   .   .   .                                                  /* INVALID STATION/SSCP ID */
   .   .   . SEND MU TO PU.SVC_MGR.CSC_MGR.SEND;              /* CHAPTER 13     */
   .   .   END;
   .   .
   .   . WHEN(FSM_PU_ACT_RES = ACTIVE)                       /* PAGE 11-118    */
   .   .   DO;
   .   .   . IF FIND_CP_ENTRY(NCB.PU_EA,SSCP_SCB_ID) = NG &   /* APPENDIX B     */
   .   .   .      RESOURCE_TOTAL_SHARE_CNT(NCB.PU_EA) >= NRCB.SHARE_LIMIT THEN
   .   .   .                                                  /* APPENDIX B     */
   .   .   .   CALL CHANGE_MU_TO_NEG_RSP(X'082C'); /* APPENDIX B, SHARE LIMIT EXCEEDED */
   .   .   .
   .   .   . ELSE
   .   .   .   IF UPM_ACTPU_CPID_CHECK ¬= OK THEN             /* PAGE 11-112    */
   .   .   .     CALL CHANGE_MU_TO_NEG_RSP(X'081D');
   .   .   .                                  /* APPENDIX B, INVALID STATION/SSCP ID */
   .   .   .   ELSE
   .   .   .     DO;
   .   .   .     . IF ACTPU_RQ.TYPE_ACTIVATION = COLD |
   .   .   .     .      FIND_CP_ENTRY(NCB.PU_EA,SSCP_SCB_ID) = OK THEN/* APPENDIX B */
   .   .   .     .   DO;
   .   .   .     .   . CALL NS.LCP_RESET_PROC(SSCP_SCB_ID);  /* PAGE 11-33     */
   .   .   .     .   . IF NCB.PU_TYPE = (PU_T4 | PU_T5) &
   .   .   .     .   .      CPCB_PTR ¬= NULL & CPCB.ER_VR_SUPP = PRE_ER_VR THEN
   .   .   .     .   .   DO;
   .   .   .     .   .   . MU_PTR = UPM_CREATE_RQ('ANSC');   /* APPENDIX B     */
   .   .   .     .   .   . SEND MU TO SNS.SEND;              /* CHAPTER 6      */
   .   .   .     .   .   END;
   .   .   .     .   END;
   .   .   .     . FIND CPCB IN CPCB_LIST WHERE(CPCB.CP_SCB_ID = SSCP_SCB_ID);
   .   .   .     . IF CPCB_PTR = NULL THEN
   .   .   .     .   DO;
   .   .   .     .   . CREATE CPCB;
   .   .   .     .   . CPCB.CP_SCB_ID = SSCP_SCB_ID;
   .   .   .     .   . INSERT CPCB IN CPCB_LIST;
   .   .   .     .   . IF PID = PID4 THEN
   .   .   .     .   .   CPCB.ER_VR_SUPP =  ¬PRE_ER_VR;
   .   .   .     .   . ELSE
   .   .   .     .   .   CPCB.ER_VR_SUPP = PRE_ER_VR;
   .   .   .     .   . IF NCB.PU_TYPE = (PU_T4 | PU_T5) THEN
   .   .   .     .   .   CPCB.NS_LSA_RQD = UPM_EXTRACT_NS_LSA_RQD;/* PAGE 11-114  */
   .   .   .     .   . IF NCB.PU_TYPE = (PU_T1 | PU_T2) THEN
   .   .   .     .   .   CALL FSM_CP_SESS_SDT('ACTIVE');      /* PAGE 11-119    */
   .   .   .     .   . IF FSM_PU_ACT_RES = RESET THEN         /* PAGE 11-118    */
   .   .   .     .   .   CALL FSM_PU_ACT_RES;                /* PAGE 11-118    */
   .   .   .     .   END;
   .   .   .     . MU_PTR = UPM_CREATE_RSP('ACTPU');          /* APPENDIX B     */
   .   .   .     . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);      /* APPENDIX B     */
   .   .   .     .
   .   .   .     END;
   .   .   . SEND MU TO PU.SVC_MGR.CSC_MGR.SEND;              /* CHAPTER 13     */
   .   .   END;
   .   END;
   . END;
   . RETURN;
END NS.SC_PROC;
```

This page intentionally left blank

```
NS.LCP_RESET_PROC: PROCEDURE(SSCP_SCB_ID);

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE RESETS  FSM'S AS SPECIFIED BY THE  LOST CONTROL POINT │
  │                 RESET OPTIONS CHOSEN AT SYSTEM DEFINITION TIME OR IN SETCV.  │
  │                                                                              │
  │      INPUT:     SSCP_SCB_ID IDENTIFYING THE SSCP THAT HAS BEEN LOST          │
  │                                                                              │
  │      OUTPUT:    RESET SIGNAL TO  FSM_PU_ACT_RES; THE LINK AND  ADJACENT LINK STATION │
  │                 FSM'S  ARE RESET.   ENTRIES IN  THE  NRCB_LIST ARE  CHANGED TO  FREE │
  │                 NETWORK  ADDRESSES  AND  TO  REMOVE THE  ASSIGNMENT  OF  BF.PU'S  OR │
  │                 ADJACENT  LINK  STATIONS.   LU'S  ARE REMOVED  BY  THE  LU  SERVICES │
  │                 MANAGER.   BF.LU'S ARE  REMOVED BY  THE  BOUNDARY FUNCTION  SERVICES │
  │                 MANAGER.                                                     │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 NS.SC_PROC                          PAGE 11-30               │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 FSM_PU_ACT_RES                      PAGE 11-118              │
  │                 FSM_PU_T2_LOAD                      PAGE 11-118              │
  │                 NS.ALS_RESET                        PAGE 11-95               │
  │                 NS.LINK_RESET                       PAGE 11-94               │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL SSCP_SCB_ID PTR;
  DCL RES_EA BIT(16);

  FIND CPCB IN CPCB_LIST WHERE(SSCP_SCB_ID = CPCB.CP_SCB_ID);
  IF CPCB_PTR = NULL THEN                  /* POSSIBLE WHEN AN ACTPU(COLD) IS RECEIVED   */
    RETURN;

  SCAN NRCB_LIST PTR(NRCB_PTR);
  .
  . RES_EA = NRCB.ELEMENT_ADDRESS;
  .
  . IF DETERMINE_LCP_RESET_OPTION(RES_EA) = STOP &          /* APPENDIX B             */
  .       FIND_CP_ENTRY(RES_EA,SSCP_SCB_ID) = OK &          /* APPENDIX B             */
  .       RESOURCE_TOTAL_SHARE_CNT(RES_EA) = 1 THEN         /* APPENDIX B             */
  .    SELECT ANYORDER(NRCB.RESOURCE_CATEGORY);
  .    .
  .    . WHEN(PU)
  .    .    DO;
  .    .    . CALL FSM_PU_ACT_RES('RESET');               /* PAGE 11-118            */
  .    .    . IF NCB.PU_TYPE = T2 THEN
  .    .    .    CALL FSM_PU_T2_LOAD('RESET');            /* PAGE 11-118            */
  .    .    END;
  .    .
  .    . WHEN(LINK)
  .    .    CALL NS.LINK_RESET(RES_EA,LINK_FAILURE);     /* PAGE 11-94             */
  .    .
  .    . WHEN(ALS)
  .    .    DO;
  .    .    . CALL NS.ALS_RESET(RES_EA);                 /* PAGE 11-94             */
  .    .    . IF NRCB.ASSIGNING_CP_SCB_ID = SSCP_SCB_ID THEN
  .    .    .    REMOVE NRCB FROM NRCB_LIST DISCARD;
  .    .    END;
  .    .
  .    . WHEN(BF.PU)
  .    .    DO;
  .    .    . IF NRCB.ASSIGNING_CP_SCB_ID = SSCP_SCB_ID THEN
  .    .    .    REMOVE NRCB FROM NRCB_LIST DISCARD;
  .    .    END;
  .    .
  .    . OTHERWISE;
  .    .
  .    END;
  .
  . CALL DELETE_CP_ENTRY(RES_EA,SSCP_SCB_ID);             /* APPENDIX B             */
  .
  SCANEND;

  REMOVE CPCB FROM CPCB_LIST DISCARD;

  RETURN;
END NS.LCP_RESET_PROC;
```

```
NS.CS_RCV: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE  RECEIVES ALL RU'S SENT  FROM THE SNS  LAYER (CHAPTER
│                 6).  THE  NETWORK SERVICES  CONFIGURATION REQUEST  CODE (LOCATED  IN
│                 BYTE 2, RELATIVE TO ZERO) IS USED TO ROUTE THE RU TO THE APPROPRIATE
│                 ROUTINE FOR PROCESSING OF THE REQUEST OR RESPONSE.
│
│      INPUT:     CONFIGURATION SERVICES RU'S FROM NS.RCV
│
│      OUTPUT:    RU'S (USUALLY RESPONSES)  TO SNS.SEND (CHAPTER 6),  RU'S TO LINK_MGR
│                 FOR PROCESSING, INOP  REQUESTS TO SNS.SEND.  REFER  TO THE PROCEDURE
│                 HANDLING  THE SPECIFIC  RU'S  FOR THE  OUTPUT FOR  THE  RU THAT  WAS
│                 RECEIVED.
│
│      NOTE:      THE PROTOCOL BOUNDARY  THAT IS MAINTAINED BETWEEN  THE PU.SVC_MGR.NS
│                 AND SNS IS DEFINED AS FOLLOWS:
│                 1. TH INFORMATION:
│                     • SESSION IDENTIFICATION (HSID)
│                     • SEQUENCE NUMBER (ONLY FOR REQUESTS RECEIVED BY PU.SVC_MGR.NS)
│                 2. RH INFORMATION:
│                     • REQUEST/RESPONSE INDICATOR
│                     • SENSE DATA INCLUDED INDICATOR FOR NEGATIVE RESPONSES
│                 3. RU INFORMATION  (FOR THE FAPL DESCRIPTION, THE  RU INFORMATION IS
│                     IN THE SAME FORMAT AS THE RU DESCRIPTION IN APPENDIX E).
│
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                 PU.SVC_MGR.NS.RCV                   PAGE 11-28
│
│      REFERS TO THE FOLLOWING PROCEDURE(S):
│                 ADJ_PU_LOAD_PROC                    PAGE 11-102
│                 NS.ACTLINK_PROC                     PAGE 11-36
│                 NS.ADDLINK_ADDLINKSTA_PROC          PAGE 11-62
│                 NS.CONN_PROC                        PAGE 11-40
│                 NS.CONTACT_PROC                     PAGE 11-42
│                 NS.DACTLINK_PROC                    PAGE 11-37
│                 NS.DELETENR_PROC                    PAGE 11-63
│                 NS.DISCONTACT_PROC                  PAGE 11-45
│                 NS.DUMP_PROC                        PAGE 11-48
│                 NS.FNA_PROC                         PAGE 11-55
│                 NS.LOAD_PROC                        PAGE 11-46
│                 NS.RNAA_PROC                        PAGE 11-52
│                 NS.RPO_PROC                         PAGE 11-50
│                 NS.SETCV_PROC                       PAGE 11-64
│                 PU_T2_LOAD_PROC                     PAGE 11-100
│                 UPM_ANA_PROC                        PAGE 11-112
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
IF RRI = RSP THEN
  DO;
  . IF RQ_CODE = (NS_IPL_INIT | NS_IPL_TEXT |
  .      NS_IPL_FINAL | NS_IPL_ABORT) THEN
  .   CALL PU_T2_LOAD_PROC;                          /* PAGE 11-100      */
  . ELSE
  .   DISCARD MU;
  . RETURN;
  END;
                                                                        /*
                ┌──────────────────────────────────────────┐
                │  THE FOLLOWING CHECK IS VALID FOR ALL FMD RU'S  │
                │  SINCE  BYTES 3  AND 4  OF THE  RU ARE  EITHER  │
                │  RESERVED(EQUIVALENT TO PU ELEMENT ADDRESS) OR  │
                │  CONTAIN A TRUE NETWORK ADDRESS.                │
                └──────────────────────────────────────────┘
                                                                        */
ELSE
                                                     /* RU IS A REQUEST  */
  DO;
  . NRCB_PTR = LOCATE_NODE_RESOURCE(NSC_RQ.TARGET_ADDRESS);   /* APPENDIX B */
  . IF NRCB_PTR = NULL |
  .      NRCB.RESOURCE_CATEGORY ¬= (LU | LINK | ALS | PU | BF.PU | BF.LU) THEN
  .   DO;
  .   . CALL CHANGE_MU_TO_NEG_RSP(X'0801'); /* APPENDIX B, RESOURCE NOT AVAILABLE */
  .   . SEND MU TO SNS.SEND;                         /* CHAPTER 6        */
  .   . RETURN;
  .   END;
  END;
```

```
   SELECT ANYORDER;                                                  /* REQUEST CODE SELECTION   */
   .
   . WHEN(NS_RQ_CODE = (ABCONN | ABCONNOUT | ACTCONNIN | CONNOUT | DACTCONNIN))
   .   CALL NS.CONN_PROC;                                           /* PAGE 11-40               */
   .
   . WHEN(NS_RQ_CODE = ACTLINK)
   .   CALL NS.ACTLINK_PROC;                                        /* PAGE 11-36               */
   .
   . WHEN(NS_RQ_CODE = DACTLINK)
   .   CALL NS.DACTLINK_PROC;                                       /* PAGE 11-37               */
   .
   . WHEN(NS_RQ_CODE = (ADDLINK | ADDLINKSTA))
   .   CALL NS.ADDLINK_ADDLINKSTA_PROC;                             /* PAGE 11-62               */
   .
   . WHEN(NS_RQ_CODE = ANA)
   .   CALL UPM_ANA_PROC;                                           /* PAGE 11-112              */
   .
   . WHEN(NS_RQ_CODE = CONTACT)
   .   CALL NS.CONTACT_PROC;                                        /* PAGE 11-42               */
   .
   . WHEN(NS_RQ_CODE = DELETENR)
   .   CALL NS.DELETENR_PROC;                                       /* PAGE 11-63               */
   .
   . WHEN(NS_RQ_CODE = DISCONTACT)
   .   CALL NS.DISCONTACT_PROC;                                     /* PAGE 11-45               */
   .
   . WHEN(NS_RQ_CODE = FNA)
   .   CALL NS.FNA_PROC;                                            /* PAGE 11-55               */
   .
   . WHEN(NS_RQ_CODE = (DUMPINIT | DUMPTEXT | DUMPFINAL))
   .   CALL NS.DUMP_PROC;                                           /* PAGE 11-48               */
   .
   . WHEN(NS_RQ_CODE = INITPROC)
   .   CALL ADJ_PU_LOAD_PROC;                                       /* PAGE 11-102              */
   .
   . WHEN(NS_RQ_CODE = (IPLINIT | IPLTEXT | IPLFINAL))
   .   CALL NS.LOAD_PROC;                                           /* PAGE 11-46               */
   .
   . WHEN(NS_RQ_CODE = RNAA)
   .   CALL NS.RNAA_PROC;                                           /* PAGE 11-52               */
   .
   . WHEN(NS_RQ_CODE = RPO)
   .   CALL NS.RPO_PROC;                                            /* PAGE 11-50               */
   .
   . WHEN(NS_RQ_CODE = SETCV)
   .   CALL NS.SETCV_PROC;                                          /* PAGE 11-64               */
   .
   . OTHERWISE
   .   DO;
   .   . IF RRI = RQ THEN
   .   .   DO;
   .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'1003'); /* APPENDIX B, FUNCTION NOT SUPPORTED        */
   .   .   . SEND MU TO SNS.SEND;                                   /* CHAPTER 6                */
   .   .   END;
   .   END;
   .
   END;
   RETURN;
END NS.CS_RCV;
```

```
NS.ACTLINK_PROC: PROCEDURE;

                                                                                /*
┌──────────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  WHEN  ACTLINK IS  THE  INPUT, THIS  PROCEDURE  GENERATES A  NEGATIVE  │
│                 RESPONSE IF A LINK TEST IS IN PROGRESS, THE LINK IS BEING TRACED, OR  │
│                 THE SHARE  LIMIT HAS ALREADY BEEN  REACHED.  IF THE LINK  IS PENDING  │
│                 RESET OR A  RESET IS IN PROGRESS,  A -RSP(0818) IS GENERATED  OR THE  │
│                 REQUEST IS QUEUED  PENDING COMPLETION OF THE RESET.  IF  THE LINK IS  │
│                 ALREADY PEND ACTIVE, THE RU IS DISCARDED AFTER ADDING THE CP ADDRESS  │
│                 TO THE CP_LIST.   IF THE RESOURCE FSM IS ALREADY  ACTIVE, A POSITIVE  │
│                 RESPONSE IS  GENERATED AND THE CP  ADDRESS IS ADDED TO  THE CP_LIST.  │
│                 IF THE RESOURCE FSM IS RESET, ACTLINK IS SENT TO THE APPROPRIATE DLC  │
│                 AND TO THE RESOURCE FSM AND THE CP ADDRESS IS ADDED TO THE CP_LIST.   │
│                                                                              │
│      INPUT:     ACTLINK FROM SNS.RCV (CHAPTER 6)                             │
│                                                                              │
│      OUTPUT:    POSITIVE AND NEGATIVE  RESPONSES TO ACTLINK TO  SNS.SEND (CHAPTER 6)  │
│                 IN THE APPROPRIATE HALF-SESSION; ACTLINK TO DLC.  THE REQUEST MAY BE  │
│                 QUEUED PENDING A RESET COMPLETION.                           │
│                                                                              │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
│                 NS.CS_RCV                       PAGE 11-34                   │
│                                                                              │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
│                 FSM_LINK_ACT_RES                PAGE 11-119                  │
└──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL LINK_EA BIT(16);

  NRCB_PTR = LOCATE_NODE_RESOURCE(NSC_RQ.TARGET_ADDRESS);
                                                          /* APPENDIX B          */
  LINK_EA = NRCB.ELEMENT_ADDRESS;
  FIND LSCB IN LSCB_LIST WHERE(LSCB.EA = LINK_EA);

  IF FSM_LINK_ACT_RES = TEST_IN_PROGRESS THEN             /* PAGE 11-119         */
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'0818');  /* APPENDIX B, LINK PROC IN PROGRESS  */
    . SEND MU TO SNS.SEND;                                /* CHAPTER 6           */
    END;

  ELSE
    IF FSM_LINK_ACT_RES = (PEND_RESET | RESET_IN_PROGRESS) THEN  /* PAGE 11-119  */
      DO;
      . IF FIND_CP_ENTRY(LINK_EA,SCB_PTR) = OK THEN       /* APPENDIX B          */
      .   DO;
      .   .
      .   . CALL CHANGE_MU_TO_NEG_RSP(X'0818');  /* APPENDIX B, LINK PROC IN PROGRESS  */
      .   . SEND MU TO SNS.SEND;                          /* CHAPTER 6           */
      .   END;
      .
      . ELSE
      .   CALL ENQUEUE_RU_FOR_RESOURCE(LINK_EA);          /* APPENDIX B          */
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │  SEE NS.SIG_RSP_PRI|SEC PAGES 11-86 AND 11-88,  │
                  │  WHERE THIS REQUEST IS DEQUEUED.         │
                  └──────────────────────────────────────────┘
                                                                                */
      END;

    ELSE
      IF (FSM_LINK_ACT_RES = (ACTIVE | PEND_ACTIVE)) &    /* PAGE 11-119         */
           RESOURCE_TOTAL_SHARE_CNT(LINK_EA) >= NRCB.SHARE_LIMIT THEN
                                                          /* APPENDIX B          */
        DO;
        . CALL CHANGE_MU_TO_NEG_RSP(X'082C');  /* APPENDIX B, SHARE LIMIT EXCEEDED  */
        . SEND MU TO SNS.SEND;                            /* CHAPTER 6           */
        END;

      ELSE
                                      /* FSM_LINK_TRACE_RES = RESET | SHARE LIMIT OK  */
        DO;
        . CALL ADD_CP_ENTRY(LINK_EA,SCB_PTR);             /* APPENDIX B          */
        . IF FSM_LINK_ACT_RES = ACTIVE THEN               /* PAGE 11-119         */
        .   DO;
        .   . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);        /* APPENDIX B          */
        .   . SEND MU TO SNS.SEND;                        /* CHAPTER 6           */
        .   END;
        . ELSE
        .   IF FSM_LINK_ACT_RES = RESET THEN              /* PAGE 11-119         */
        .     DO;
        .     . CALL FSM_LINK_ACT_RES;     /* ACTLINK   PAGE 11-119              */
        .     . SEND MU TO PU.SVC_MGR.LINK_MGR;
        .     END;
        .
        .   ELSE
        .     IF FSM_LINK_ACT_RES = PEND_ACTIVE THEN      /* PAGE 11-119         */
        .       DISCARD MU;
        END;

  RETURN;
END NS.ACTLINK_PROC;
```

```
NS.DACTLINK_PROC: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │     FUNCTION:  WHEN  DACTLINK IS  THE INPUT,  THIS PROCEDURE  GENERATES A  NEGATIVE  │
 │                RESPONSE  IF LINK  TEST  IS IN  PROGRESS.  IF  THE  LINK IS  ALREADY  │
 │                PENDING RESET OR  A RESET IS IN  PROGRESS, THE REQUEST IS  QUEUED IF  │
 │                FROM A  DIFFERENT CP THAN THE  ONE CURRENTLY INITIATING  THE ACTION.  │
 │                IF  THE  RESOURCE FSM  IS  ALREADY  RESET,  A POSITIVE  RESPONSE  IS  │
 │                GENERATED.  IF THE RESOURCE FSM IS  ACTIVE OR PENDING ACTIVE, THE CP  │
 │                LIST IS CHECKED.  IF ANY CP'S APPEAR  ON THE LIST OTHER THAN THE ONE  │
 │                ISSUING THIS DACTLINK,  THEN A POSITIVE RESPONSE TO  THE DACTLINK IS  │
 │                GENERATED.  OTHERWISE, THE  DACTLINK IS SENT TO THE  LINK_MGR AND TO  │
 │                THE RESOURCE FSM.                                                     │
 │                                                                                      │
 │     INPUT:     DACTLINK FROM SNS.RCV (CHAPTER 6)                                     │
 │                                                                                      │
 │     OUTPUT:    POSITIVE AND NEGATIVE RESPONSES TO  DACTLINK TO SNS.SEND (CHAPTER 6)  │
 │                IN THE APPROPRIATE  HALF-SESSION; DACTLINK TO DLC.   THE REQUEST MAY   │
 │                ALSO BE QUEUED PENDING COMPLETION OF LINK RESET.                      │
 │                                                                                      │
 │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                                        │
 │                NS.CS_RCV                          PAGE 11-34                         │
 │                                                                                      │
 │     REFERS TO THE FOLLOWING PROCEDURE(S):                                            │
 │                DACTLINK_RCV_CHECKS                PAGE 11-39                         │
 │                FSM_LINK_ACT_RES                   PAGE 11-119                        │
 │                FSM_LINK_TRACE_RES                 PAGE 11-120                        │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL LINK_EA BIT(16);

  NRCB_PTR = LOCATE_NODE_RESOURCE(NSC_RQ.TARGET_ADDRESS);        /* APPENDIX B          */
  LINK_EA = NRCB.ELEMENT_ADDRESS;
  FIND LSCB IN LSCB_LIST WHERE(LSCB.EA = LINK_EA);

  IF DACTLINK_RCV_CHECKS(LINK_EA) = OK THEN                      /* PAGE 11-39          */
    DO;
    . IF FSM_LINK_ACT_RES = RESET THEN                           /* PAGE 11-119         */
    .
    .   DO;
    .   . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);                  /* APPENDIX B          */
    .   . SEND MU TO SNS.SEND;                                   /* CHAPTER 6           */
    .   END;
    . ELSE
    .   IF FSM_LINK_ACT_RES = TEST_IN_PROGRESS THEN
    .     DO;                                                    /* PAGE 11-119         */
    .     . CALL CHANGE_MU_TO_NEG_RSP(X'0818');  /* APPENDIX B, LINK PROC IN PROGRESS */
    .     . SEND MU TO SNS.SEND;                                 /* CHAPTER 6           */
    .     END;
    .   ELSE
    .     IF FSM_LINK_ACT_RES = (PEND_RESET | RESET_IN_PROGRESS) THEN /* PAGE 11-119   */
    .       DO;
    .       . IF FIND_CP_ENTRY(LINK_EA,SCB_PTR) = OK THEN        /* APPENDIX B          */
    .       .   DO;
    .       .   .
    .       .   . CALL CHANGE_MU_TO_NEG_RSP(X'0818');  /* APPENDIX B, LINK PROC IN PROGRESS */
    .       .   . SEND MU TO SNS.SEND;                           /* CHAPTER 6           */
    .       .   END;
    .       . ELSE
    .       .   CALL ENQUEUE_RU_FOR_RESOURCE(LINK_EA);           /* APPENDIX B          */
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │  SEE NS.SIG_RSP_PRI|SEC PAGES 11-86 AND 11-88,  │
                    │  WHERE THIS REQUEST IS DEQUEUED.         │
                    └──────────────────────────────────────────┘
                                                                        */
    .       END;
    .     ELSE
    .       IF FSM_LINK_ACT_RES = ACTIVE |                       /* PAGE 11-119         */
    .            FSM_LINK_ACT_RES = PEND_ACTIVE THEN
    .         DO;
    .         . IF RESOURCE_TOTAL_SHARE_CNT(LINK_EA) > 1 THEN    /* APPENDIX B          */
    .         .   DO;
    .         .   . CALL DELETE_CP_ENTRY(LINK_EA,SCB_PTR);       /* APPENDIX B          */
    .         .   . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);         /* APPENDIX B          */
    .         .   . SEND MU TO SNS.SEND;                         /* CHAPTER 6           */
    .         .   .
    .         .   END;
    .         . ELSE
    .         .   DO;
    .         .   . CALL FSM_LINK_TRACE_RES('RESET');            /* PAGE 11-120         */
    .         .   . CALL FSM_LINK_ACT_RES;                       /* PAGE 11-119         */
    .         .   . SEND MU TO PU.SVC_MGR.LINK_MGR;
    .         .   END;
    .         .
    .         END;
    END;
  ELSE
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'081A');  /* APPENDIX B, REQUEST SEQUENCE ERROR */
    . SEND MU TO SNS.SEND;                                       /* CHAPTER 6           */
    END;

  RETURN;
END NS.DACTLINK_PROC;
```

This page intentionally left blank

```
DACTLINK_RCV_CHECKS: PROCEDURE(LINK_EA) RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO  PERFORM STATE  RECEIVE  CHECKS ON  A GROUP  OF  FSM'S FOR  EVERY │
  │                 ADJACENT LINK STATION ASSOCIATED WITH A GIVEN LINK.          │
  │                                                                              │
  │      INPUT:     THE ELEMENT ADDRESS OF THE LINK                              │
  │                                                                              │
  │      OUTPUT:    OK, IF ALL FSM'S ARE IN THE RESET STATE; NG, IF NOT          │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 NS.DACTLINK_PROC                    PAGE 11-37               │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 ALS_SEC_SUBTREE_CHECK               PAGE 11-97               │
  │                 FSM_ALS_CONNECTED_RES               PAGE 11-121              │
  │                 FSM_ALS_CONTACT_DISCONTACT_RES      PAGE 11-122              │
  │                 FSM_LINK_CONNIN_RES                 PAGE 11-120              │
  │                 FSM_LINK_CONNOUT_RES                PAGE 11-121              │
  │                 FSM_XID_FORMAT_2                    PAGE 11-126              │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL LINK_EA BIT(16);
  DCL CHECK BIT(1);
  DCL SAVE_NRCB_PTR PTR;

  CHECK = OK;
  SAVE_NRCB_PTR = NRCB_PTR;
  IF NRCB.SWITCHED_LINK = SWITCHED &
       (FSM_LINK_CONNIN_RES ¬= RESET |                                  /* PAGE 11-120  */
       FSM_LINK_CONNOUT_RES ¬= RESET) THEN                              /* PAGE 11-121  */
    CHECK = NG;
  SCAN NRCB_LIST PTR(NRCB_PTR) WHILE(CHECK = OK);
  .
  . IF NRCB.RESOURCE_CATEGORY = ALS &
  .       NRCB.ASSOCIATED_RESOURCE = LINK_EA THEN
  .    DO;
  .    . IF NRCB.PRI_SEC_ROLE = CONFIGURABLE THEN
  .    .    DO;
  .    .    . FIND LSCB IN LSCB_LIST WHERE(LSCB.EA = LINK_EA);
  .    .    . IF FSM_XID_FORMAT_2 ¬= RESET THEN                         /* PAGE 11-126  */
  .    .    .    CHECK = NG;
  .    .    END;
  .    . ELSE
  .    .    SELECT ANYORDER(NRCB.LINK_DLC_ROLE);
  .    .    .
  .    .    . WHEN(PRIMARY)                                            /* ADJACENT LINK STATION */
  .    .    .                                  /* IS A PRIMARY LINK STATION */
  .    .    .    DO;
  .    .    .    . IF NRCB.SWITCHED_LINK = SWITCHED &
  .    .    .    .       FSM_ALS_CONNECTED_RES ¬= RESET THEN           /* PAGE 11-121  */
  .    .    .    .    CHECK = NG;
  .    .    .    .
  .    .    .    . IF FSM_ALS_CONTACT_DISCONTACT_RES ¬= RESET THEN    /* PAGE 11-122  */
  .    .    .    .    CHECK = NG;
  .    .    .    END;
  .    .    .
  .    .    . WHEN(SECONDARY)                                          /* ADJACENT LINK STATION */
  .    .    .                                  /* IS A SECONDARY LINK STATION */
  .    .    .    DO;
  .    .    .    . IF NRCB.SWITCHED_LINK = SWITCHED &
  .    .    .    .       FSM_ALS_CONNECTED_RES ¬= RESET THEN           /* PAGE 11-121  */
  .    .    .    .    CHECK = NG;
  .    .    .    . ELSE
  .    .    .    .    CHECK = ALS_SEC_SUBTREE_CHECK(NRCB.ELEMENT_ADDRESS);
  .    .    .    .                                                     /* PAGE 11-97   */
  .    .    .    END;
  .    .    END;
  .    END;
  .
  SCANEND;

  NRCB_PTR = SAVE_NRCB_PTR;

  RETURN(CHECK);
END DACTLINK_RCV_CHECKS;
```

```
NS.CONN_PROC: PROCEDURE;

                                                                                /*
┌──────────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS ROUTINE  HANDLES LINK  CONNECTION PROCEDURES.   FOR A  SWITCHED │
│                 LINK, THE SHARE LIMIT IS ONE.  ALL REQUESTS ARE REJECTED IF THE LINK │
│                 HAS NOT  BEEN ACTIVATED OR  THE LINK  IS NOT A  SWITCHED CONNECTION. │
│                 ABCONN IS REJECTED  IF THE CONNECTION IS NOT  ACTIVE; OTHERWISE, THE │
│                 REQUEST IS FORWARDED TO LINK_MGR FOR PROCESSING.             │
│                                                                              │
│      INPUT:     ACTCONNIN, DACTCONNIN, CONNOUT, ABCONNOUT,  AND ABCONN REQUESTS FROM │
│                 SNS.RCV (CHAPTER 6)                                          │
│                                                                              │
│      OUTPUT:    REQUESTS TO DLC; -RSP TO SNS.SEND                            │
│                                                                              │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
│                 NS.CS_RCV                       PAGE 11-34                   │
│                                                                              │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
│                 FSM_ALS_CONNECTED_RES           PAGE 11-121                  │
│                 FSM_LINK_CONNIN_RES             PAGE 11-120                  │
│                 FSM_LINK_CONNOUT_RES            PAGE 11-121                  │
│                 NS.ALS_RESET                    PAGE 11-95                   │
└──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL LINK_EA BIT(16);
  DCL SAVE_NRCB_PTR PTR;

  NRCB_PTR = LOCATE_NODE_RESOURCE(NSC_RQ.TARGET_ADDRESS);        /* APPENDIX B    */
  LINK_EA = NRCB.ELEMENT_ADDRESS;
  FIND LSCB IN LSCB_LIST WHERE(LSCB.EA = LINK_EA);
  IF FIND_CP_ENTRY(LINK_EA,SCB_PTR) = NG THEN                    /* APPENDIX B    */
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'0817');  /* APPENDIX B, LINK INACTIVE          */
    . SEND MU TO SNS.SEND;                                       /* CHAPTER 6     */
    . RETURN;
    END;
  IF NRCB.SWITCHED_LINK ¬= SWITCHED THEN
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'080C');  /* APPENDIX B, PROCEDURE NOT SUPPORTED */
    . SEND MU TO SNS.SEND;                                       /* CHAPTER 6     */
    . RETURN;
    END;
  SELECT ANYORDER(NS_RQ_CODE);
                                                                                /*
                    ┌──────────────────────────────────────┐
                    │              ACTCONNIN               │
                    └──────────────────────────────────────┘
                                                                                */
  . WHEN(ACTCONNIN)
  .   DO;
  .   . IF FSM_LINK_CONNIN_RES = RESET THEN
  .   .   DO;
  .   .   . CALL FSM_LINK_CONNIN_RES;                           /* PAGE 11-120   */
  .   .   . SEND MU TO PU.SVC_MGR.LINK_MGR;
  .   .   END;
  .   . ELSE
  .   .   DO;
  .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'0815');  /* APPENDIX B, FUNCTION ACTIVE   */
  .   .   . SEND MU TO SNS.SEND;                                 /* CHAPTER 6     */
  .   .   END;
  .   END;
                                                                                /*
                    ┌──────────────────────────────────────┐
                    │              DACTCONNIN              │
                    └──────────────────────────────────────┘
                                                                                */
  . WHEN(DACTCONNIN)
  .   DO;
  .   . IF FSM_LINK_CONNIN_RES = ACTIVE THEN
  .   .   DO;                                                   /* PAGE 11-120   */
  .   .   . CALL FSM_LINK_CONNIN_RES;                           /* PAGE 11-120   */
  .   .   . SEND MU TO PU.SVC_MGR.LINK_MGR;
  .   .   END;
  .   . ELSE
  .   .   DO;
  .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'0816');  /* APPENDIX B, FUNCTION INACTIVE */
  .   .   . SEND MU TO SNS.SEND;                                 /* CHAPTER 6     */
  .   .   END;
  .   END;
```

```
                                                                                /*
                    ┌──────────────────────────────────────┐
                    │               CONNOUT                │
                    └──────────────────────────────────────┘
                                                                                */
  . WHEN(CONNOUT)
  .   DO;
  .   . IF FSM_LINK_CONNOUT_RES ¬= RESET THEN                  /* PAGE 11-121    */
  .   .   DO;
  .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'0815');  /* APPENDIX B, FUNCTION ACTIVE  */
  .   .   . SEND MU TO SNS.SEND;                               /* CHAPTER 6      */
  .   .   . RETURN;
  .   .   END;
  .   . ELSE
  .   .   DO;
  .   .   . SAVE_NRCB_PTR = NRCB_PTR;
  .   .   . NRCB_PTR = FIND_ALS_FOR_RESOURCE(LINK_EA);         /* APPENDIX B     */
  .   .   . IF FSM_ALS_CONNECTED_RES ¬= RESET THEN
  .   .   .   DO;                                              /* PAGE 11-121    */
  .   .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'0801');  /* APPENDIX B, RESOURCE NOT AVAILABLE */
  .   .   .   . SEND MU TO SNS.SEND;                           /* CHAPTER 6      */
  .   .   .   . RETURN;
  .   .   .   END;
  .   .   . NRCB_PTR = SAVE_NRCB_PTR;
  .   .   END;
  .   . CALL FSM_LINK_CONNOUT_RES;                            /* PAGE 11-121    */
  .   . SEND MU TO PU.SVC_MGR.LINK_MGR;
  .   END;
                                                                                /*
                    ┌──────────────────────────────────────┐
                    │             ABCCNNOUT                │
                    └──────────────────────────────────────┘
                                                                                */
  . WHEN(ABCONNOUT)
  .   DO;
  .   . IF FSM_LINK_CONNOUT_RES = ACTIVE THEN                 /* PAGE 11-121    */
  .   .   DO;
  .   .   . CALL FSM_LINK_CONNOUT_RES;                        /* PAGE 11-121    */
  .   .   . SEND MU TO PU.SVC_MGR.LINK_MGR;
  .   .   END;
  .   . ELSE
  .   .   DO;
  .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'0816');  /* APPENDIX B, FUNCTION INACTIVE */
  .   .   . SEND MU TO SNS.SEND;
  .   .   END;
  .   END;
                                                                                /*
                    ┌──────────────────────────────────────┐
                    │               ABCONN                 │
                    └──────────────────────────────────────┘
                                                                                */
  . WHEN(ABCONN)
  .   DO;
  .   . NRCB_PTR = FIND_ALS_FOR_RESOURCE(LINK_EA);             /* APPENDIX B     */
  .   . IF FSM_ALS_CONNECTED_RES = ACTIVE THEN                /* PAGE 11-121    */
  .   .   DO;
  .   .   . CALL NS.ALS_RESET(NRCB.ELEMENT_ADDRESS);          /* PAGE 11-95     */
  .   .   . CALL FSM_ALS_CONNECTED_RES;                       /* PAGE 11-121    */
  .   .   . SEND MU TO PU.SVC_MGR.LINK_MGR;
  .   .   END;
  .   . ELSE
  .   .   DO;
  .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'0816');  /* APPENDIX B, FUNCTION INACTIVE */
  .   .   . SEND MU TO SNS.SEND;                               /* CHAPTER 6      */
  .   .   END;
  .   END;
  END;
  RETURN;
END NS.CONN_PROC;
```

```
NS.CONTACT_PROC: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THE ADJACENT LINK STATION ELEMENT ADDRESS  IN THE CONTACT RU IS USED      │
│                 TO DETERMINE WHICH  ADJACENT LINK STATION TO  CONTACT.  THE RESOURCE      │
│                 FSM'S FOR THIS ALS ARE THEN CHECKED  TO SEE IF THE CONTACT IS VALID.      │
│                 IF IT IS, THE CONTACT FUNCTION IS PERFORMED.                              │
│                                                                                           │
│      INPUT:     CONTACT FROM SNS.RCV                                                      │
│                                                                                           │
│      OUTPUT:    CONTACT TO  THE CONTACT  AND CONT_DISCONTACT  RESOURCE FSM'S;  RESET       │
│                 SIGNAL   TO  THE   DISCONTACT,  IPL,   AND   DUMP  RESOURCE   FSM'S;      │
│                 ±RSP(CONTACT) TO SNS.SEND; CONTACTED(LOADED) TO SNS.SEND                  │
│                                                                                           │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                            │
│                 NS.CS_RCV                          PAGE 11-34                             │
│                                                                                           │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                                                │
│                 CONTACT_CONFIG                     PAGE 11-44                             │
│                 FSM_ALS_CONTACT_DISCONTACT_RES     PAGE 11-122                            │
│                 FSM_ALS_SEC_DUMP_RES               PAGE 11-122                            │
│                 FSM_ALS_SEC_IPL_RES                PAGE 11-123                            │
│                 FSM_ALS_SEC_RPO_RES                PAGE 11-123                            │
│                 LINK_STATUS_CHECKS                 PAGE 11-51                             │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL CP_ACTIVE_ID PTR;
  DCL ALS_EA BIT(16);

  IF LINK_STATUS_CHECKS(NSC_RQ.TARGET_ADDRESS) = NG THEN        /* PAGE 11-51      */
    RETURN;

  NRCB_PTR = FIND_ALS_FOR_RESOURCE(NSC_RQ.TARGET_ADDRESS);      /* APPENDIX B      */
  ALS_EA = NRCB.ELEMENT_ADDRESS;
  FIND LSCB IN LSCB_LIST WHERE(LSCB.EA = ALS_EA);
  CP_ACTIVE_ID = SCB_PTR;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │     CONTACT FOR A CONFIGURABLE LINK STATION     │
                  └──────────────────────────────────────────┘
                                                                        */
  IF NRCB.PRI_SEC_ROLE = CONFIGURABLE THEN
    CALL CONTACT_CONFIG(ALS_EA);                                /* PAGE 11-44      */

  ELSE
    SELECT ANYORDER;
    .
    . WHEN(FSM_ALS_CONTACT_DISCONTACT_RES = RESET)              /* PAGE 11-122     */
    .   DO;
    .   . IF NRCB.LINK_DLC_ROLE = PRIMARY &
    .   .      (FSM_ALS_SEC_IPL_RES ¬= RESET |                  /* PAGE 11-123     */
    .   .      FSM_ALS_SEC_DUMP_RES ¬= RESET |                  /* PAGE 11-122     */
    .   .      FSM_ALS_SEC_RPO_RES ¬= RESET) THEN               /* PAGE 11-123     */
    .   .   DO;
    .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'0818');  /* APPENDIX B, LINK PROC IN PROGRESS  */
    .   .   . SEND MU TO SNS.SEND;
    .   .   END;
    .   .
    .   . ELSE
    .   .   DO;
    .   .   . CALL FSM_ALS_CONTACT_DISCONTACT_RES;              /* PAGE 11-122     */
    .   .   . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);              /* APPENDIX B      */
    .   .   . SEND MU TO SNS.SEND;                              /* CHAPTER 6       */
    .   .   . SEND 'CONTACT' TO PU.SVC_MGR.LINK_MGR;
    .   .   . CALL ADD_CP_ENTRY(ALS_EA,CP_ACTIVE_ID);           /* APPENDIX B      */
    .   .   END;
    .   END;
```

```
.  WHEN(FSM_ALS_CONTACT_DISCONTACT_RES = ACTIVE |
.       FSM_ALS_CONTACT_DISCONTACT_RES = PEND_ACTIVE)
.    DO;                                                    /* PAGE 11-122          */
.    .
.    . IF FIND_CP_ENTRY(ALS_EA,CP_ACTIVE_ID) = OK THEN      /* APPENDIX B           */
.    .   DO;
.    .   . CALL CHANGE_MU_TO_NEG_RSP(X'0815');  /* APPENDIX B, FUNCTION ACTIVE      */
.    .   . SEND MU TO SNS.SEND;                             /* CHAPTER 6            */
.    .   END;
.    .
.    . ELSE
.    .   IF RESOURCE_TOTAL_SHARE_CNT(ALS_EA) < NRCB.SHARE_LIMIT THEN
.    .     DO;                                              /* APPENDIX B           */
.    .     . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);           /* APPENDIX B           */
.    .     . SEND MU TO SNS.SEND;                           /* CHAPTER 6            */
.    .     . CALL ADD_CP_ENTRY(ALS_EA,CP_ACTIVE_ID);        /* APPENDIX B           */
.    .     . IF FSM_ALS_CONTACT_DISCONTACT_RES = ACTIVE THEN /* PAGE 11-122          */
.    .     .   DO;
.    .     .   . MU_PTR = UPM_CREATE_RQ('CONTACTED(LOADED)'); /* APPENDIX B         */
.    .     .   . SEND MU TO SNS.SEND;                       /* CHAPTER 6            */
.    .     .   END;
.    .     END;
.    .
.    .   ELSE
.    .     DO;
.    .     . CALL CHANGE_MU_TO_NEG_RSP(X'082C');  /* APPENDIX B, RESOURCE SHARE LIMIT */
.    .     . SEND MU TO SNS.SEND;                           /* CHAPTER 6            */
.    .     END;
.    END;
.
. OTHERWISE
.   CALL ENQUEUE_RU_FOR_RESOURCE(ALS_EA);                   /* APPENDIX B           */
                                                            /*
                  ┌──────────────────────────────────────┐
                  │ SEE NS.SIG_RSP_PRI|SEC PAGES 11-86 AND 11-88, │
                  │ WHERE THIS REQUEST IS DEQUEUED.      │
                  └──────────────────────────────────────┘
                                                            */
  END;

 RETURN;
END NS.CONTACT_PROC;
```

```
CONTACT_CONFIG: PROCEDURE(ALS_EA);

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE  IS CALLED  BY NS.CONTACT_PROC.   IT HANDLES  CONTACT  │
  │                 REQUESTS RECEIVED  BY A PU_T4  OR PU_T5 NODE  FOR A LINK  STATION IN  │
  │                 ANOTHER  PU_T4 OR  PU_T5  NODE, IN  THE  CASE  WHERE THAT  STATION'S  │
  │                 PRIMARY/SECONDARY ROLE IS CONFIGURABLE.                               │
  │                                                                                       │
  │      INPUT:     THE CURRENT MESSAGE UNIT IS A  CONTACT REQUEST.  ALS_EA, THE ELEMENT   │
  │                 ADDRESS OF  THE STATION TO  BE CONTACTED,  IS PASSED AS  A PARAMETER  │
  │                 FROM  THE CALLING  PROCEDURE.  SCB_PTR  ADDRESSES  THE CORRECT  SCB;  │
  │                 LSCB_PTR ADDRESSES THE CORRECT LSCB;  NRCB_PTR ADDRESSES THE CORRECT  │
  │                 NRCB.                                                                 │
  │                                                                                       │
  │      OUTPUT:    RESPONSE TO CONTACT TO SNS.SEND, XID TO LINK_MGR.                      │
  │                                                                                       │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                        │
  │                 NS.CONTACT_PROC                    PAGE 11-42                         │
  │                                                                                       │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                            │
  │                 FSM_ALS_CONTACT_DISCONTACT_RES     PAGE 11-122                        │
  │                 FSM_ALS_SEC_DUMP_RES               PAGE 11-122                        │
  │                 FSM_ALS_SEC_IPL_RES                PAGE 11-123                        │
  │                 FSM_ALS_SEC_RPO_RES                PAGE 11-123                        │
  │                 FSM_XID_FORMAT_2                   PAGE 11-126                        │
  │                 XID_FORMAT_2_BUILD                 PAGE 11-71                         │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL ALS_EA BIT(16);
  IF FSM_ALS_CONTACT_DISCONTACT_RES = PEND_RESET |                      /* PAGE 11-122      */
       FSM_ALS_CONTACT_DISCONTACT_RES = RESET_IN_PROGRESS THEN
    CALL ENQUEUE_RU_FOR_RESOURCE(ALS_EA);                               /* APPENDIX B       */
  IF FSM_ALS_SEC_IPL_RES ¬= RESET |                                     /* PAGE 11-123      */
       FSM_ALS_SEC_DUMP_RES ¬= RESET |                                  /* PAGE 11-122      */
       FSM_ALS_SEC_RPO_RES ¬= RESET THEN                                /* PAGE 11-123      */
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'0818');  /* APPENDIX B, LINK PROCEDURE IN PROCESS      */
    . SEND MU TO SNS.SEND;                                              /* CHAPTER 6        */
    END;

  ELSE
    SELECT ANYORDER;
    .
    . WHEN(FSM_XID_FORMAT_2 = RESET)                                    /* PAGE 11-126      */
    .   DO;
    .   . CALL FSM_XID_FORMAT_2;          /* PAGE 11-126, INPUT IS CONTACT                */
    .   .
    .   . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);                         /* APPENDIX B       */
    .   . SEND MU TO SNS.SEND;                                          /* CHAPTER 6        */
    .   . CALL ADD_CP_ENTRY(ALS_EA,SCB_PTR);                           /* APPENDIX B       */
    .   .
    .   . TGCB_PTR = LSCB.TGCBPTR;
    .   . IF LSCB.TGCBPTR = NULL THEN     /* STATION NOT YET ASSIGNED TO A TG             */
    .   .   LSCB.XID_SEND.TGN = X'00';
    .   . ELSE
    .   .                                 /* STATION ALREADY ASSIGNED TO A TG             */
    .   .   LSCB.XID_SEND.TGN = TGCB.TGN;
    .   . LSCB.CONTACTED_STATUS = X'00';
    .   . LSCB.XID_SEND.ERROR_STATUS = 0;
    .   .
    .   . LSCB.XID_SEND.CONTACT_OR_LOAD_STAT = CMD_SENDER;
    .   . CALL XID_FORMAT_2_BUILD;                                      /* PAGE 11-71       */
    .   END;
    . OTHERWISE
    .   IF NRCB.SHARE_LIMIT > RESOURCE_TOTAL_SHARE_CNT(ALS_EA) THEN
    .     DO;
    .     . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);                        /* APPENDIX B       */
    .     . SEND MU TO SNS.SEND;                                        /* CHAPTER 6        */
    .     . CALL ADD_CP_ENTRY(ALS_EA,SCB_PTR);                          /* APPENDIX B       */
    .     . IF FSM_XID_FORMAT_2 = ACTIVE THEN                           /* PAGE 11-126      */
    .     .   DO;
    .     .   . MU_PTR = UPM_CREATE_RQ('CONTACTED(LOADED)');            /* APPENDIX B       */
    .     .   . SEND MU TO SNS.SEND;                                    /* CHAPTER 6        */
    .     .   END;
    .     END;
    .   ELSE
    .     DO;
    .     . CALL CHANGE_MU_TO_NEG_RSP(X'082C');
    .     .       /* APPENDIX B, RESOURCE-SHARING LIMIT REACHED                           */
    .     . SEND MU TO SNS.SEND;                                        /* CHAPTER 6        */
    .     END;
    END;

  RETURN;

END CONTACT_CONFIG;
```

```
NS.DISCONTACT_PROC: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THE ADJACENT  LINK STATION ELEMENT ADDRESS  ON THE DISCONTACT  RU IS │
 │                 USED TO  DETERMINE WHICH ADJACENT  LINK STATION TO  DISCONTACT.  THE │
 │                 RESOURCE  FSM'S  FOR  THIS  ALS  ARE THEN  CHECKED  TO  SEE  IF  THE │
 │                 DISCONTACT  IS VALID.   IF IT  IS  THEN THE  DISCONTACT FUNCTION  IS │
 │                 PERFORMED.                                           │
 │                                                                      │
 │      INPUT:     DISCONTACT FROM SNS.RCV (CHAPTER 6)                  │
 │                                                                      │
 │      OUTPUT:    DISCONTACT TO THE DISCONTACT  AND CONTACT-DISCONTACT RESOURCE FSM'S; │
 │                 RESET  SIGNAL  TO  THE  CONTACT,  IPL,  AND  DUMP  RESOURCE  FSM'S; │
 │                 ±RSP(DISCONTACT) TO SNS.SEND                         │
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 NS.CS_RCV                       PAGE 11-34           │
 │                                                                      │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
 │                 FSM_ALS_CONTACT_DISCONTACT_RES  PAGE 11-122          │
 │                 FSM_ALS_SEC_DUMP_RES            PAGE 11-122          │
 │                 FSM_ALS_SEC_IPL_RES             PAGE 11-123          │
 │                 FSM_ALS_SEC_RPO_RES             PAGE 11-123          │
 │                 LINK_STATUS_CHECKS              PAGE 11-51           │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL CP_ACTIVE_ID PTR;
  DCL ALS_EA BIT(16);

  CP_ACTIVE_ID = SCB_PTR;
  IF LINK_STATUS_CHECKS(NSC_RQ.TARGET_ADDRESS) = NG THEN          /* PAGE 11-51          */
    RETURN;
  NRCB_PTR = FIND_ALS_FOR_RESOURCE(NSC_RQ.TARGET_ADDRESS);
                                                                  /* APPENDIX B          */
  ALS_EA = NRCB.ELEMENT_ADDRESS;
  FIND LSCB IN LSCB_LIST WHERE(LSCB.EA = ALS_EA);
  IF FIND_CP_ENTRY(ALS_EA,CP_ACTIVE_ID) = NG THEN                 /* APPENDIX B          */
    DO;
    . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);                        /* APPENDIX B          */
    . SEND MU TO SNS.SEND;                                        /* CHAPTER 6           */
    END;
  ELSE
    IF FSM_ALS_CONTACT_DISCONTACT_RES = ACTIVE |
         FSM_ALS_CONTACT_DISCONTACT_RES = PEND_ACTIVE THEN
      DO;                                                         /* PAGE 11-122         */
      . IF NRCB.LINK_DLC_ROLE = SECONDARY &
      .       (FSM_ALS_SEC_IPL_RES ¬= RESET |                     /* PAGE 11-123         */
      .        FSM_ALS_SEC_DUMP_RES ¬= RESET |                    /* PAGE 11-122         */
      .        FSM_ALS_SEC_RPO_RES ¬= RESET) THEN                 /* PAGE 11-123         */
      .    DO;
      .    . CALL CHANGE_MU_TO_NEG_RSP(X'0818');  /* APPENDIX B, LINK PROC IN PROGRESS */
      .    . SEND MU TO SNS.SEND;                                 /* CHAPTER 6           */
      .    END;
      . ELSE
      .    IF RESOURCE_TOTAL_SHARE_CNT(ALS_EA) > 1 THEN          /* APPENDIX B          */
      .      DO;
      .      . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);              /* APPENDIX B          */
      .      . SEND MU TO SNS.SEND;                              /* CHAPTER 6           */
      .      . CALL DELETE_CP_ENTRY(ALS_EA,CP_ACTIVE_ID);        /* APPENDIX B          */
      .      END;
      .    ELSE
      .      DO;
      .      . CALL FSM_ALS_CONTACT_DISCONTACT_RES;              /* PAGE 11-122         */
      .      . SEND MU TO PU.SVC_MGR.LINK_MGR;
      .      . CALL DELETE_CP_ENTRY(ALS_EA,CP_ACTIVE_ID);        /* APPENDIX B          */
      .      END;
      END;
    ELSE
      IF FSM_ALS_CONTACT_DISCONTACT_RES = PEND_RESET |            /* PAGE 11-122         */
           FSM_ALS_CONTACT_DISCONTACT_RES = RESET_IN_PROGRESS THEN
        CALL ENQUEUE_RU_FOR_RESOURCE(ALS_EA);                     /* APPENDIX B          */
      ELSE
                                                                  /*  TEST_IN_PROGRESS   */
        DO;
        . CALL CHANGE_MU_TO_NEG_RSP(X'0818');  /* APPENDIX B, LINK PROC IN PROGRESS */
        . SEND MU TO SNS.SEND;                                    /* CHAPTER 6           */
        END;
  RETURN;
END NS.DISCONTACT_PROC;
```

```
NS.LOAD_PROC: PROCEDURE:

                                                                                /*
┌──────────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  REQUESTS  THAT HAVE  TARGETS THAT  ARE NOT  SECONDARY ADJACENT  LINK │
│                 STATIONS  ARE REJECTED.   WHEN IPLINIT  IS RECEIVED,  THE FSM'S  FOR │
│                 DUMP, IPL, AND  RPO ARE CHECKED TO VERIFY THAT  ALL THESE PROCEDURES │
│                 ARE  INTERRUPTIBLE.   IF  NOT,  THE REQUEST  IS  REJECTED.   IF  THE │
│                 PROCEDURES  ARE  ALL  INTERRUPTIBLE, THEN  THE  RESOURCE  FSM'S  ARE │
│                 UPDATED  TO  INDICATE THAT  AN  IPL  IS  BEGINNING.  IF  IPLTEXT  OR │
│                 IPLFINAL IS  THE INPUT WHEN THE  IPL RESOURCE FSM DOES  NOT INDICATE │
│                 IPL IN  PROGRESS, THEN IT IS  REJECTED; OTHERWISE, THE  IPL RESOURCE │
│                 FSM IS CALLED.                                                       │
│                                                                                      │
│      INPUT:     IPLINIT, IPLTEXT, AND IPLFINAL REQUESTS                              │
│                                                                                      │
│      OUTPUT:    IPLINIT, IPLTEXT, AND IPLFINAL TO THE IPL RESOURCE FSM; RESET SIGNAL │
│                 TO THE DUMP, CONTACT, AND DISCONTACT RESOURCE FSM'S;                 │
│                 -RSP(IPLINIT,0801|0809|0817|0818|0849) AND                           │
│                 -RSP(IPLTEXT|IPLFINAL,0809|0817|0849) TO SNS.SEND                    │
│                                                                                      │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                       │
│                 NS.CS_RCV                          PAGE 11-34                        │
│                                                                                      │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                                           │
│                 ALS_SEC_SUBTREE_INTERRUPT          PAGE 11-99                        │
│                 FSM_ALS_CONTACT_DISCONTACT_RES     PAGE 11-122                       │
│                 FSM_ALS_SEC_IPL_RES                PAGE 11-123                       │
│                 FSM_XID_FORMAT_2                   PAGE 11-126                       │
│                 LINK_STATUS_CHECKS                 PAGE 11-51                        │
│                 NS.ALS_PROC_RESET                  PAGE 11-96                        │
│                 UPM_SAVE_SNF                       PAGE 11-116                       │
└──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL ALS_EA BIT(16);
  DCL IPL_CP_SCB_ID PTR;

  IPL_CP_SCB_ID = SCB_PTR;
  IF LINK_STATUS_CHECKS(NSC_RQ.TARGET_ADDRESS) = NG THEN          /* PAGE 11-51      */
    RETURN;
  NRCB_PTR = FIND_ALS_FOR_RESOURCE(NSC_RQ.TARGET_ADDRESS);        /* APPENDIX B      */
  ALS_EA = NRCB.ELEMENT_ADDRESS;
  FIND LSCB IN LSCB_LIST WHERE(LSCB.EA = ALS_EA);
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │ A REMOTE PU_T4 ON A  CONFIGURABLE LINK MAY BE │
                  │ IPLED  ONLY  AFTER IT  HAS  BEEN  SUCESSFULLY │
                  │ CONTACTED.                                    │
                  └──────────────────────────────────────────┘
                                                                                */
  IF NRCB.PRI_SEC_ROLE = CONFIGURABLE &
       FSM_XID_FORMAT_2 ¬= ACTIVE THEN                            /* PAGE 11-126     */
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'0818');  /* APPENDIX B, LINK PROCEDURE IN PROGRESS */
    . SEND MU TO SNS.SEND;                                       /* CHAPTER 6       */
    . RETURN;
    END;
  IF NRCB.LINK_DLC_ROLE = SECONDARY THEN
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'0849');  /* APPENDIX B, INVALID REQUESTED PROC     */
    . SEND MU TO SNS.SEND;                                       /* CHAPTER 6       */
    . RETURN;
    END;
```

```
SELECT ANYORDER(NS_RQ_CODE);
                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │                IPLINIT                   │
                    └──────────────────────────────────────────┘
                                                                                */
. WHEN(IPLINIT)
.   DO;
.   . IF ALS_SEC_SUBTREE_INTERRUPT(ALS_EA) = OK THEN          /* PAGE 11-99    */
.   .   DO;
                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │ IPL OF A SHARED RESOURCE REQUIRES THAT ALL │
                    │ OTHER CP'S RECEIVE AN INOP TO NOTIFY THEM │
                    │ THAT PU HAS BEEN RE-INITIALIZED.         │
                    └──────────────────────────────────────────┘
                                                                                */
.   .   . SCAN NRCB.CP_INDIRECT_LIST PTR(CP_INDIRECT_PTR);
.   .   . . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
.   .   . . IF CPCB.CP_SCB_ID ¬= IPL_CP_SCB_ID THEN
.   .   . .   DO;
.   .   . .   . MU_PTR = UPM_CREATE_RQ('INOP');              /* APPENDIX B    */
.   .   . .   . INOP_RQ.INOP_REASON = X'6';                  /* IPL IN PROGRESS */
.   .   . .   . SCB_PTR = CPCB.CP_SCB_ID;
.   .   . .   . SEND MU TO SNS.SEND;                         /* CHAPTER 6     */
.   .   . .   END;
.   .   . SCANEND;
.   .   . CALL NS.ALS_PROC_RESET(ALS_EA);                    /* PAGE 11-96    */
.   .   . CALL ADD_CP_ENTRY(ALS_EA,IPL_CP_SCB_ID);           /* APPENDIX B    */
.   .   . CALL FSM_ALS_SEC_IPL_RES;      /* IPLINIT      PAGE 11-123          */
.   .   . CALL UPM_SAVE_SNF;                                 /* PAGE 11-116   */
.   .   . SEND MU TO PU.SVC_MGR.LINK_MGR;
.   .   END;
.   . ELSE
.   .   IF FSM_ALS_CONTACT_DISCONTACT_RES = PEND_RESET |     /* PAGE 11-122   */
.   .        FSM_ALS_CONTACT_DISCONTACT_RES = RESET_IN_PROGRESS THEN
.   .     CALL ENQUEUE_RU_FOR_RESOURCE(ALS_EA);              /* APPENDIX B    */
.   .   ELSE
.   .     DO;                                                /* TEST_IN_PROGRESS */
.   .     . CALL CHANGE_MU_TO_NEG_RSP(X'0818');  /* APPENDIX B, LINK PROC IN PROGRESS */
.   .     . SEND MU TO SNS.SEND;                             /* CHAPTER 6     */
.   .     END;
.   END;
                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │          IPLTEXT AND IPLFINAL            │
                    └──────────────────────────────────────────┘
                                                                                */
. WHEN(IPLTEXT,IPLFINAL)
.   DO;
.   . IF FIND_CP_ENTRY(ALS_EA,SCB_PTR) = OK &                 /* APPENDIX B    */
.   .      FSM_ALS_SEC_IPL_RES = INIPL THEN                   /* PAGE 11-123   */
.   .   DO;
.   .   . CALL FSM_ALS_SEC_IPL_RES;                          /* PAGE 11-123   */
.   .   . CALL UPM_SAVE_SNF;                                 /* PAGE 11-116   */
.   .   . SEND MU TO PU.SVC_MGR.LINK_MGR;
.   .   END;
.   . ELSE
.   .   DO;
.   .   . CALL CHANGE_MU_TO_NEG_RSP(X'081A');  /* APPENDIX B, REQUEST SEQUENCE ERROR */
.   .   . SEND MU TO SNS.SEND;                               /* CHAPTER 6     */
.   .   END;
.   END;
  END;
  RETURN;
END NS.LOAD_PROC;
```

```
NS.DUMP_PROC: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  REQUESTS THAT HAVE TARGETS THAT ARE  NOT SECONDARY DLC ADJACENT LINK │
│                 STATIONS ARE  REJECTED.  WHEN DUMPINIT IS  THE INPUT THE  DUMP, IPL, │
│                 AND RPO  FSM'S ARE CHECKED TO  VERIFY THAT ALL THESE  PROCEDURES ARE │
│                 INTERRUPTIBLE.  IF NOT, THE REQUEST  IS REJECTED.  IF THE PROCEDURES │
│                 ARE ALL INTERRUPTIBLE, THEN THE  RESOURCE AND HALF-SESSION FSM'S ARE │
│                 UPDATED  TO INDICATE  THAT  A DUMP  IS  BEGINNING.   IF DUMPTEXT  OR │
│                 DUMPFINAL IS THE INPUT WHEN THE  DUMP RESOURCE FSM DOES NOT INDICATE │
│                 DUMP IN PROGRESS, THEN IT IS REJECTED;  OTHERWISE IT IS THE INPUT TO │
│                 THE DUMP RESOURCE FSM.                                   │
│                                                                          │
│      INPUT:     DUMPINIT, DUMPTEXT, DUMPFINAL REQUESTS                   │
│                                                                          │
│      OUTPUT:    DUMPINIT, DUMPTEXT,  AND DUMPFINAL TO  THE DUMP RESOURCE  FSM, RESET │
│                 SIGNAL TO THE RESOURCE FSM'S;                            │
│                 -RSP(DUMPINIT,0801|0809|0817|0818|0849) AND              │
│                 -RSP(DUMPTEXT|DUMPFINAL, 0809|0817|0849) TO SNS.SEND     │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 NS.CS_RCV                          PAGE 11-34            │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 ALS_SEC_SUBTREE_INTERRUPT          PAGE 11-99            │
│                 FSM_ALS_CONTACT_DISCONTACT_RES     PAGE 11-122           │
│                 FSM_ALS_SEC_DUMP_RES               PAGE 11-122           │
│                 FSM_XID_FORMAT_2                   PAGE 11-126           │
│                 LINK_STATUS_CHECKS                 PAGE 11-51            │
│                 NS.ALS_PROC_RESET                  PAGE 11-96            │
│                 UPM_SAVE_SNF                       PAGE 11-116           │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL ALS_EA BIT(16);
  DCL DUMP_CP_SCB_ID PTR;

  DUMP_CP_SCB_ID = SCB_PTR;
  IF LINK_STATUS_CHECKS(NSC_RQ.TARGET_ADDRESS) = NG THEN        /* PAGE 11-51      */
    RETURN;
  NRCB_PTR = FIND_ALS_FOR_RESOURCE(NSC_RQ.TARGET_ADDRESS);
                                                                /* APPENDIX B      */
  ALS_EA = NRCB.ELEMENT_ADDRESS;
  FIND LSCB IN LSCB_LIST WHERE(LSCB.EA = ALS_EA);
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │ A REMOTE PU_T4 ON A  CONFIGURABLE LINK MAY BE │
                  │ DUMPED  ONLY AFTER  IT  HAS BEEN  SUCESSFULLY │
                  │ CONTACTED.                               │
                  └──────────────────────────────────────────┘
                                                                        */
  IF NRCB.PRI_SEC_ROLE = CONFIGURABLE &
       FSM_XID_FORMAT_2 ¬= ACTIVE THEN                          /* PAGE 11-126     */
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'0818');  /* APPENDIX B, LINK PROCEDURE IN PROGRESS */
    . SEND MU TO SNS.SEND;                                      /* CHAPTER 6       */
    . RETURN;
    END;
  IF NRCB.LINK_DLC_ROLE = SECONDARY THEN
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'0849');  /* APPENDIX B, INVALID REQUESTED PROC */
    . SEND MU TO SNS.SEND;                                      /* CHAPTER 6       */
    . RETURN;
    END;
```

```
SELECT ANYORDER(NS_RQ_CODE);
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │                DUMPINIT                  │
                  └──────────────────────────────────────────┘
                                                                                */
. WHEN(DUMPINIT)
.   DO;
.   . IF ALS_SEC_SUBTREE_INTERRUPT(ALS_EA) = OK THEN          /* PAGE 11-99    */
.   .   DO;
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │ DUMP OF  A SHARED RESOURCE REQUIRES  THAT ALL │
                  │ OTHER  CP'S RECEIVE  AN INOP  TO NOTIFY  THEM │
                  │ THAT PU IS BEING DUMPED.                 │
                  └──────────────────────────────────────────┘
                                                                                */
.   .   . SCAN NRCB.CP_INDIRECT_LIST PTR(CP_INDIRECT_PTR);
.   .   . . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
.   .   . . IF CPCB.CP_SCB_ID ¬= DUMP_CP_SCB_ID THEN
.   .   . .   DO;
.   .   . .   . MU_PTR = UPM_CREATE_RQ('INOP');              /* APPENDIX B    */
.   .   . .   . INOP_RQ.INOP_REASON = X'6';                 /* DUMP IN PROGRESS */
.   .   . .   . SCB_PTR = CPCB.CP_SCB_ID;
.   .   . .   . SEND MU TO SNS.SEND;                        /* CHAPTER 6     */
.   .   . .   END;
.   .   . SCAN END;
.   .   . CALL NS.ALS_PROC_RESET(ALS_EA);                   /* PAGE 11-96    */
.   .   . CALL ADD_CP_ENTRY(ALS_EA,DUMP_CP_SCB_ID);         /* APPENDIX B    */
.   .   . CALL FSM_ALS_SEC_DUMP_RES;     /* DUMPINIT  PAGE 11-122             */
.   .   . CALL UPM_SAVE_SNF;                                /* PAGE 11-116   */
.   .   . SEND MU TO PU.SVC_MGR.LINK_MGR;
.   .   END;
.   . ELSE
.   .   IF FSM_ALS_CONTACT_DISCONTACT_RES = PEND_RESET |    /* PAGE 11-122   */
.   .        FSM_ALS_CONTACT_DISCONTACT_RES = RESET_IN_PROGRESS THEN
.   .     CALL ENQUEUE_RU_FOR_RESOURCE(ALS_EA);             /* APPENDIX B    */
.   .   ELSE
.   .                                                       /* TEST_IN_PROGRESS */
.   .     DO;
.   .     . CALL CHANGE_MU_TO_NEG_RSP(X'0818');  /* APPENDIX B, LINK PROC IN PROGRESS */
.   .     . SEND MU TO SNS.SEND;                            /* CHAPTER 6     */
.   .     END;
.   END;
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │          DUMPTEXT | DUMPFINAL            │
                  └──────────────────────────────────────────┘
                                                                                */
. WHEN(DUMPTEXT,DUMPFINAL)
.   DO;
.   . IF FIND_CP_ENTRY(ALS_EA,SCB_PTR) = OK &               /* APPENDIX B    */
.   .      FSM_ALS_SEC_DUMP_RES = INDUMP THEN               /* PAGE 11-122   */
.   .   DO;
.   .   . CALL FSM_ALS_SEC_DUMP_RES;     /* DUMPTEXT | DUMPFINAL  PAGE 11-122 */
.   .   . CALL UPM_SAVE_SNF;                                /* PAGE 11-116   */
.   .   . SEND MU TO PU.SVC_MGR.LINK_MGR;
.   .   END;
.   . ELSE
.   .   DO;
.   .   . CALL CHANGE_MU_TO_NEG_RSP(X'081A');  /* APPENDIX B, REQUEST SEQUENCE ERROR */
.   .   . SEND MU TO SNS.SEND;                              /* CHAPTER 6     */
.   .   END;
.   END;
  END;
  RETURN;
END NS.DUMP_PROC;
```

```
NS.RPO_PROC: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  REQUESTS THAT HAVE TARGETS THAT ARE  NOT SECONDARY DLC ADJACENT LINK │
 │                 STATIONS  ARE REJECTED.   THE CURRENT  STATE  OF ALL  THE FSM'S  FOR  │
 │                 CONTACT, DISCONTACT, XID, IPL, DUMP, AND RPO ARE CHECKED.  IF ALL OF  │
 │                 THEM ARE RESET,  THEN THE RPO IS  PROCESSED.  IF ANY OF  THE CHECKED  │
 │                 FSM'S IS NOT RESET, THE RPO IS REJECTED.                              │
 │                                                                                       │
 │      INPUT:     RPO REQUESTS                                                          │
 │                                                                                       │
 │      OUTPUT:    RPO TO  THE RPO RESOURCE FSM;  -RSP(RPO,0801|0809|0817|0834|0849) TO  │
 │                 SNS.SEND                                                              │
 │                                                                                       │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                        │
 │                 NS.CS_RCV                          PAGE 11-34                         │
 │                                                                                       │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                                            │
 │                 ALS_SEC_SUBTREE_CHECK              PAGE 11-97                         │
 │                 FSM_ALS_CONTACT_DISCONTACT_RES     PAGE 11-122                        │
 │                 FSM_ALS_SEC_RPO_RES                PAGE 11-123                        │
 │                 FSM_XID_FORMAT_2                   PAGE 11-126                        │
 │                 LINK_STATUS_CHECKS                 PAGE 11-51                         │
 │                 UPM_SAVE_SNF                       PAGE 11-116                        │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL ALS_EA BIT(16);
  DCL RPO_CP_SCB_ID PTR;

  RPO_CP_SCB_ID = SCB_PTR;
  IF LINK_STATUS_CHECKS(NSC_RQ.TARGET_ADDRESS) = NG THEN          /* PAGE 11-51          */
    RETURN;
  NRCB_PTR = FIND_ALS_FOR_RESOURCE(NSC_RQ.TARGET_ADDRESS);
                                                                  /* APPENDIX B          */
  ALS_EA = NRCB.ELEMENT_ADDRESS;
  FIND LSCB IN LSCB_LIST WHERE(LSCB.EA = ALS_EA);
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │  A REMOTE PU_T4 ON A  CONFIGURABLE LINK MAY BE  │
                  │  POWERED   OFF   ONLY  AFTER   IT   HAS   BEEN  │
                  │  SUCESSFULLY CONTACTED.                  │
                  └──────────────────────────────────────────┘
                                                                        */
  IF NRCB.PRI_SEC_ROLE = CONFIGURABLE &
      FSM_XID_FORMAT_2 ¬= ACTIVE THEN                             /* PAGE 11-126         */
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'0818');  /* APPENDIX B, LINK PROCEDURE IN PROGRESS    */
    . SEND MU TO SNS.SEND;                                        /* CHAPTER 6           */
    . RETURN;
    END;

  IF NRCB.LINK_DLC_ROLE = SECONDARY THEN
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'0849');  /* APPENDIX B, INVALID REQUESTED PROC        */
    . SEND MU TO SNS.SEND;                                        /* CHAPTER 6           */
    END;
  ELSE
    IF FSM_ALS_CONTACT_DISCONTACT_RES = PEND_RESET |              /* PAGE 11-122         */
         FSM_ALS_CONTACT_DISCONTACT_RES = RESET_IN_PROGRESS THEN
      CALL ENQUEUE_RU_FOR_RESOURCE(ALS_EA);                       /* APPENDIX B          */
    ELSE
      IF ALS_SEC_SUBTREE_CHECK(ALS_EA) = NG THEN                  /* PAGE 11-97          */
        DO;
        . CALL CHANGE_MU_TO_NEG_RSP(X'0834');  /* APPENDIX B, RPO NOT INITIATED        */
        . SEND MU TO SNS.SEND;                                    /* CHAPTER 6           */
        END;
      ELSE
        DO;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │  RPO TO  A SHARED  RESOURCE REQUIRES  THAT ALL  │
                  │  OTHER  CP'S RECEIVE  AN INOP  TO NOTIFY  THEM  │
                  │  THAT PU IS BEING POWERED OFF.           │
                  └──────────────────────────────────────────┘
                                                                        */
        . SCAN NRCB.CP_INDIRECT_LIST PTR(CP_INDIRECT_PTR);
        . . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
        . . IF CPCB.CP_SCB_ID ¬= RPO_CP_SCB_ID THEN
        . .   DO;
        . .   . MU_PTR = UPM_CREATE_RQ('INOP');                   /* APPENDIX B          */
        . .   . INOP_RQ.INOP_REASON = X'7';                       /* RPO IN PROGRESS     */
        . .   . SCB_PTR = CPCB.CP_SCB_ID;
        . .   . SEND MU TO SNS.SEND;                              /* CHAPTER 6           */
        . .   END;
        . SCANEND;
        . CALL FSM_ALS_SEC_RPO_RES;                               /* RPO    PAGE 11-123  */
        . CALL DELETE_ALL_CP_ENTRIES(ALS_EA);                     /* APPENDIX B          */
        . CALL ADD_CP_ENTRY(ALS_EA,RPO_CP_SCB_ID);                /* APPENDIX B          */
        . CALL UPM_SAVE_SNF;                                      /* PAGE 11-116         */
        . SEND MU TO PU.SVC_MGR.LINK_MGR;
        END;
  RETURN;
END NS.RPO_PROC;
```

```
LINK_STATUS_CHECKS: PROCEDURE(ALS_EA) RETURNS(BIT(1));

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE VERIFIES  THE EXISTENCE OF THE  ALS_EA, THE EXISTENCE
 │                 OF A LINK  ASSOCIATED WITH THE ALS_EA, THE CONNECTION  STATUS OF THE
 │                 LINK, CHECKS THAT  THE CP IS ON  THE CP LIST FOR  THE LINK RESOURCE,
 │                 AND WHETHER  A TEST  IS IN PROGRESS  ON THAT  LINK OR  ADJACENT LINK
 │                 STATION.
 │
 │      INPUT:     ADJACENT LINK STATION ADDRESS
 │
 │      OUTPUT:    RC IS SET TO OK IF ALL  CHECKS ARE PASSED; OTHERWISE, THE REQUEST IS
 │                 CONVERTED TO A  -RSP AND RETURNED TO THE  ORIGINATING CONTROL POINT.
 │                 RC IS ALSO SET TO NG IF A -RSP IS GENERATED.
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 NS.CONTACT_PROC                    PAGE 11-42
 │                 NS.DISCONTACT_PROC                 PAGE 11-45
 │                 NS.DUMP_PROC                       PAGE 11-48
 │                 NS.LOAD_PROC                       PAGE 11-46
 │                 NS.RPO_PROC                        PAGE 11-50
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 FSM_ALS_CONNECTED_RES              PAGE 11-121
 │                 FSM_ALS_TEST_RES                   PAGE 11-124
 │                 FSM_LINK_ACT_RES                   PAGE 11-119
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL LINK_EA BIT(16);
  DCL ALS_EA BIT(16);
  DCL RC BIT(1);
  DCL SAVE_NRCB_PTR PTR;
  DCL SAVE_ALS_PTR PTR;

  SAVE_NRCB_PTR = NRCB_PTR;
  RC = OK;
  NRCB_PTR = FIND_ALS_FOR_RESOURCE(ALS_EA);                    /* APPENDIX B      */
  SAVE_ALS_PTR = NRCB_PTR;
  IF NRCB_PTR = NULL THEN
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'0801');  /* APPENDIX B, RESOURCE NOT AVAILABLE  */
    . SEND MU TO SNS.SEND;                                     /* CHAPTER 6       */
    . RC = NG;
    . NRCB_PTR = SAVE_NRCB_PTR;
    . RETURN(RC);
    END;

  NRCB_PTR = FIND_LINK_FOR_RESOURCE(ALS_EA);                   /* APPENDIX B      */
  IF NRCB_PTR = NULL |
       NRCB.RESOURCE_TYPE ¬= LINK THEN
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'0801');  /* APPENDIX B, RESOURCE NOT AVAILABLE  */
    . SEND MU TO SNS.SEND;                                     /* CHAPTER 6       */
    . RC = NG;
    . NRCB_PTR = SAVE_NRCB_PTR;
    . RETURN(RC);
    END;

  LINK_EA = NRCB.ELEMENT_ADDRESS;
  IF FIND_CP_ENTRY(LINK_EA,SCB_PTR) = NG |                     /* APPENDIX B      */
       FSM_LINK_ACT_RES ¬= ACTIVE THEN                         /* PAGE 11-119     */
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'0817');  /* APPENDIX B, LINK INACTIVE           */
    . SEND MU TO SNS.SEND;                                     /* CHAPTER 6       */
    . RC = NG;
    . NRCB_PTR = SAVE_NRCB_PTR;
    . RETURN(RC);
    END;
  NRCB_PTR = SAVE_ALS_PTR;
  IF NRCB.SWITCHED_LINK = SWITCHED &
       FSM_ALS_CONNECTED_RES ¬= ACTIVE THEN
    DO;                                                        /* PAGE 11-121     */
    . CALL CHANGE_MU_TO_NEG_RSP(X'0801');  /* APPENDIX B, RESOURCE NOT AVAILABLE  */
    . SEND MU TO SNS.SEND;                                     /* CHAPTER 6       */
    . RC = NG;
    . NRCB_PTR = SAVE_NRCB_PTR;
    . RETURN(RC);
    END;

  IF FSM_ALS_TEST_RES = TEST_IN_PROGRESS THEN                  /* PAGE 11-124     */
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'0809');  /* APPENDIX B, MODE INCONSISTENCY      */
    . SEND MU TO SNS.SEND;                                     /* CHAPTER 6       */
    . RC = NG;
    . NRCB_PTR = SAVE_NRCB_PTR;
    . RETURN;
    END;

  NRCB_PTR = SAVE_NRCB_PTR;
  RETURN(RC);
END LINK_STATUS_CHECKS;
```

```
NS.RNAA_PROC: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE DETERMINES WHETHER THE ANY OF THE RESOURCES SPECIFIED │
  │                 IN THE RNAA HAVE ALREADY BEEN  ASSIGNED NETWORK ADDRESSES BY ANOTHER │
  │                 SSCP.  IF ANY  HAVE, THE REQUEST IS  REJECTED; IF NONE HAVE,  AND IF │
  │                 THE PU HAS ENOUGH STORAGE TO ASSIGN THE ADDRESSES, THE ADDRESSES ARE │
  │                 ASSIGNED.  ADDITIONS ARE MADE TO THE NRCB_LIST IF REQUIRED.  │
  │                                                                              │
  │      INPUT:     RNAA REQUESTS FROM SNS.RCV (CHAPTER 6)                       │
  │                                                                              │
  │      OUTPUT:    -RSP( 0809|0812|0815 ) OR +RSP CONTAINING NETWORK ADDRESSES ASSIGNED │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 NS.CS_RCV                       PAGE 11-34                   │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 NS.BF_LU_ADD                    PAGE 11-60                   │
  │                 NS.BF_PU_AND_ALS_ADD            PAGE 11-61                   │
  │                 NS.LU_ADD                       PAGE 11-62                   │
  │                 RNAA_PU_OWNERSHIP_CK            PAGE 11-54                   │
  │                 RNAA_VALIDITY_CHECK             PAGE 11-53                   │
  │                 UPM_RNAA_RESOURCE_CHECK         PAGE 11-116                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF RNAA_VALIDITY_CHECK = NG THEN                          /* PAGE 11-53      */
    CALL CHANGE_MU_TO_NEG_RSP(X'0815');    /* APPENDIX B, FUNCTION ACTIVE       */

  ELSE

    IF (RNAA_RQ.ASSIGNMENT_TYPE = BF.LU) &
        (RNAA_PU_OWNERSHIP_CK = NG) THEN                    /* PAGE 11-54      */
      CALL CHANGE_MU_TO_NEG_RSP(X'0809');  /* APPENDIX B, MODE INCONSISTENCY    */

    ELSE
      DO;
      . IF UPM_RNAA_RESOURCE_CHECK = NG THEN                /* PAGE 11-116     */
      .   CALL CHANGE_MU_TO_NEG_RSP(X'0812'); /* APPENDIX B, INSUFFICIENT RESOURCE */
      .
      . ELSE
      .
      .   SELECT ANYORDER(RNAA_RQ.ASSIGNMENT_TYPE);
      .   .
      .   . WHEN(RNAA_BF_LU)
      .   .
      .   .   CALL NS.BF_LU_ADD;                           /* PAGE 11-60      */
      .   .
      .   . WHEN(RNAA_BF_PU)
      .   .
      .   .   CALL NS.BF_PU_AND_ALS_ADD;                   /* PAGE 11-61      */
      .   .
      .   . WHEN(RNAA_LU)
      .   .
      .   .   CALL NS.LU_ADD;                              /* PAGE 11-62      */
      .   .
      .   END;
      END;

  SEND MU TO SNS.SEND;                                      /* CHAPTER 6       */

  RETURN;

END NS.RNAA_PROC;
```

```
RNAA_VALIDITY_CHECK: PROCEDURE RETURNS(BIT(1));

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS  PROCEDURE  DETERMINES  WHETHER THE  REQUESTED  RESOURCES  WERE │
 │                 ASSIGNED  BY ANY  CURRENTLY  ACTIVE SSCP.  IF  NOT,  IT RETURNS  OK. │
 │                 OTHERWISE, IT RETURNS NG.                                    │
 │                                                                              │
 │      INPUT:     THE RNAA REQUEST, WHICH IS THE CURRENT MESSAGE UNIT           │
 │                                                                              │
 │      OUTPUT:    THE APPROPRIATE RETURN CODE VALUE                            │
 │                                                                              │
 │      NOTE:      THE CREATION OF PARALLEL SESSIONS  REQUIRES A UNIQUE NETWORK ADDRESS │
 │                 FOR EACH  PRIMARY LU.  THESE PLU  ADDRESSES ARE ASSOCIATED  WITH THE │
 │                 SINGLE SLU ADDRESS FOR THE LU.                               │
 │                                                                              │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
 │                 NS.RNAA_PROC                          PAGE 11-52             │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);
  DCL I FIXED BIN;
  DCL P PTR;

  RC = OK;

  SELECT ANYORDER(RNAA_RQ.ASSIGNMENT_TYPE);
  .
  . WHEN(RNAA_BF_PU)
  .
  .   DO I = 1 TO RNAA_RQ.ENTRY_CNT WHILE(RC = OK);
  .   .
  .   . SCAN NRCB_LIST PTR(NRCB_PTR) WHILE(RC = OK);
  .   . .
  .   . . IF NRCB.ASSOCIATED_RESOURCE = RNAA_RQ.TARGET_ADDRESS &
  .   . .      NRCB.RESOURCE_CATEGORY = ALS &
  .   . .      NRCB.ALS_DLC_HDR_ADDR = RNAA_RQ.SUBFIELD(I) &
  .   . .      NRCB.ASSIGNING_CP_SCB_ID ¬= NULL &
  .   . .      NRCB.ASSIGNING_CP_SCB_ID ¬= SCB_PTR THEN
  .   . .
  .   . .   RC = NG;
  .   . .
  .   . SCANEND;
  .   .
  .   END;
  .
  . WHEN(RNAA_BF_LU)
  .
  .   DO I = 1 TO RNAA_RQ.ENTRY_CNT WHILE(RC = OK);
  .   .
  .   . SCAN NRCB_LIST PTR(NRCB_PTR) WHILE(RC = OK);
  .   . .
  .   . . IF NRCB.ASSOCIATED_RESOURCE = RNAA_RQ.TARGET_ADDRESS &
  .   . .      NRCB.RESOURCE_CATEGORY = BF.LU &
  .   . .      NRCB.BF_LOCAL_ID = RNAA_RQ.SUBFIELD(I,8:15) &
  .   . .      NRCB.ASSIGNING_CP_SCB_ID ¬= NULL &
  .   . .      NRCB.ASSIGNING_CP_SCB_ID ¬= SCB_PTR THEN
  .   . .
  .   . .   RC = NG;
  .   . .
  .   . SCANEND;
  .   .
  .   END;
  .
  . WHEN(RNAA_LU)
  .   DO;
  .   .
  .   . NRCB_PTR = LOCATE_NODE_RESOURCE(RNAA_RQ.TARGET_ADDRESS);  /* APPENDIX B          */
  .   . IF NRCB_PTR = NULL |
  .   .      NRCB.RESOURCE_CATEGORY ¬= LU |
  .   .      NRCB.ASSOCIATED_RESOURCE ¬= 0 THEN                  /* NOT A SECONDARY LU  */
  .   .   RC = NG;
  .   . ELSE
  .   .   DO;
                                                                                /*
                 ┌──────────────────────────────────────────┐
                 │          CHECK FOR AN SSCP-LU SESSION    │
                 └──────────────────────────────────────────┘
                                                                                */
  .   .   . FIND P->SCB IN SCB_LIST
  .   .   .      WHERE(P->SCB.PARTNER_SA = SCB.PARTNER_SA &
  .   .   .      P->SCB.PARTNER_EA = SCB.PARTNER_EA &
  .   .   .      P->SCB.THIS_SA = NCB.NODE_SUBAREA_ADDRESS &
  .   .   .      P->SCB.THIS_EA = RNAA_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
  .   .   . IF P = NULL THEN
  .   .   .   RC = NG;
  .   .   END;
  .   END;
  END;

  RETURN(RC);
END RNAA_VALIDITY_CHECK;
```

```
RNAA_PU_OWNERSHIP_CK: PROCEDURE RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  FOR ASSIGNMENT OF BF.LU'S, THIS  PROCEDURE CHECKS THAT THE REQUESTER │
  │                 HAS  EITHER  REQUESTED ADDRESS  ASSIGNMENT  FOR  BF.PU OR  THAT  THE │
  │                 BF.PU'S CP_LIST  DOES NOT CONTAIN ANY  OTHER SSCP AND  THE REQUESTER │
  │                 HAS ACTIVATED THE TARGET BF.PU.                              │
  │                                                                              │
  │      INPUT:     THE RNAA REQUEST, WHICH IS THE CURRENT MESSAGE UNIT           │
  │                                                                              │
  │      OUTPUT:    OK, IF THE CHECK IS SUCCESSFUL; NG, IF NOT SUCCESSFUL        │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 NS.RNAA_PROC                          PAGE 11-52             │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);
  DCL P PTR;

  RC = NG;

  NRCB_PTR = LOCATE_NODE_RESOURCE(RNAA_RQ.TARGET_ADDRESS);       /* APPENDIX B      */

  IF NRCB.ASSIGNING_CP_SCB_ID = SCB_PTR THEN
    RC = OK;

  IF NRCB.ASSIGNING_CP_SCB_ID = NULL THEN
    DO;
    . FIND P->SCB IN SCB_LIST
    .      WHERE(P->SCB.PARTNER_SA = SCB.PARTNER_SA &
    .      P->SCB.PARTNER_EA = SCB.PARTNER_EA &
    .      P->SCB.THIS_SA = NCB.NODE_SUBAREA_ADDRESS &
    .      P->SCB.THIS_EA = RNAA_RQ.TARGET_ADDRESS & NCB.NODE_ELEMENT_MASK);
    . IF P ¬= NULL &                        /* ACTIVE SESSION WITH BF.PU       */
    .      FSM_SESS_BF_SSCP_LU = ACT THEN                  /* CHAPTER 13      */
    .   RC = OK;
    END;

  RETURN(RC);
END RNAA_PU_OWNERSHIP_CK;
```

```
NS.FNA_PROC: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE FREES NETWORK ADDRESSES WHEN  A VALID FNA IS RECEIVED
  │                 BY THE PU.                                                   │
  │                                                                              │
  │      INPUT:     FNA REQUESTS FROM SNS.RCV ( CHAPTER 6 )                      │
  │                                                                              │
  │      OUTPUT:    FNA RESPONSES TO SNS.SEND ( CHAPTER 6 ) AND AN UPDATED NRCB_LIST
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 NS.CS_RCV                          PAGE 11-34                │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 FNA_BF_LU_PROC                     PAGE 11-59                │
  │                 FNA_BF_PU_AND_ALS_PROC             PAGE 11-58                │
  │                 FNA_LU_PROC                        PAGE 11-59                │
  │                 FNA_VALIDITY_CHECK                 PAGE 11-56                │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RETURN_VALUE BIT(16);
  IF FNA_RQ.TARGET_ADDRESS = 0 THEN
    DO;
    . NRCB_PTR = LOCATE_NODE_RESOURCE(FNA_RQ.SUBFIELD(1));       /* APPENDIX B   */
    . IF NRCB.RESOURCE_CATEGORY = BF.PU THEN
    .   NRCB_PTR = FIND_ALS_FOR_RESOURCE(FNA_RQ.SUBFIELD(1));   /* APPENDIX B   */
    . FNA_RQ.TARGET_ADDRESS = NRCB.ASSOCIATED_RESOURCE;
    END;
  RETURN_VALUE = FNA_VALIDITY_CHECK;                             /* PAGE 11-56   */
  IF RETURN_VALUE ¬= 0 THEN
    CALL CHANGE_MU_TO_NEG_RSP(RETURN_VALUE);                     /* APPENDIX B   */
  ELSE
    DO;
    . NRCB_PTR = LOCATE_NODE_RESOURCE(FNA_RQ.TARGET_ADDRESS);    /* APPENDIX B   */
    . SELECT ANYORDER(NRCB.RESOURCE_CATEGORY);
    . .
    . . WHEN(LINK)
    . .   CALL FNA_BF_PU_AND_ALS_PROC;                           /* PAGE 11-58   */
    . .
    . . WHEN(BF.PU)
    . .   CALL FNA_BF_LU_PROC;                                   /* PAGE 11-59   */
    . .
    . . WHEN((PU | LU))
    . .   CALL FNA_LU_PROC;                                      /* PAGE 11-59   */
    . END;
    . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);                       /* APPENDIX B   */
    END;
  SEND MU TO SNS.SEND;                                           /* CHAPTER 6    */
  RETURN;
END NS.FNA_PROC;
```

```
FNA_VALIDITY_CHECK: PROCEDURE RETURNS(FIXED BINARY(16));
                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  CHECKS THAT THE  CONDITIONS REQUIRED TO ALLOW THE  REQUESTED FNA ARE │
 │                 MET.  IF  SO,  THE  PROCEDURE  RETURNS   0;  ELSE,  IT  RETURNS  THE │
 │                 APPROPRIATE SENSE CODE.                                      │
 │                                                                              │
 │      INPUT:     THE FNA REQUEST, WHICH IS THE CURRENT MESSAGE UNIT            │
 │                                                                              │
 │      OUTPUT:    ZERO IF  THE FNA  CAN BE  ACCOMPLISHED; THE  APPROPRIATE SENSE  CODE │
 │                 VALUE IF IT CANNOT BE EXECUTED                               │
 │                                                                              │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
 │                 NS.FNA_PROC                    PAGE 11-55                    │
 │                                                                              │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
 │                 ALS_SEC_SUBTREE_CHECK          PAGE 11-97                    │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RETURN_VALUE BIT(16);
  DCL P PTR;
  DCL P1 PTR;
  DCL I FIXED BIN;
  DCL TARGET_ELEMENT BIT(16);

  RETURN_VALUE = 0;
  NRCB_PTR = LOCATE_NODE_RESOURCE(FNA_RQ.TARGET_ADDRESS);        /* APPENDIX B      */
  TARGET_ELEMENT = NRCB.ELEMENT_ADDRESS;
  SELECT ANYORDER;
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │  FREE ALL ALS'S AND  BF.PU'S ASSOCIATED WITH A  │
                  │  LINK                                    │
                  └──────────────────────────────────────────┘
                                                                                */
  . WHEN((NRCB.RESOURCE_CATEGORY = LINK) & (FNA_RQ.ENTRY_CNT = ALL))
  .   SCAN NRCB_LIST PTR(NRCB_PTR) WHILE(RETURN_VALUE = 0);
  .   . IF NRCB.ASSOCIATED_RESOURCE = TARGET_ELEMENT THEN        /* IF ALS          */
  .   .   IF ALS_SEC_SUBTREE_CHECK(NRCB.ELEMENT_ADDRESS) = NG THEN /* PAGE 11-97    */
  .   .     RETURN_VALUE = X'0809';                             /* MODE INCONSISTENCY */
  .   .   ELSE
  .   .     DO;
  .   .     . FIND P->NRCB IN NRCB_LIST                         /* FIND BF.PU      */
  .   .     .      WHERE(P->NRCB.RESOURCE_TYPE = BF.PU &
  .   .     .      P->NRCB.ELEMENT_ADDRESS = NRCB.ELEMENT_ADDRESS);
  .   .     . IF P ¬= NULL THEN
  .   .     .                          /* IS THIS BF.PU ASSIGNED BY ANOTHER CP?    */
  .   .     .   IF P->NRCB.ASSIGNING_CP_SCB_ID ¬= NULL &
  .   .     .        P->NRCB.ASSIGNING_CP_SCB_ID ¬= SCB_PTR THEN
  .   .     .     RETURN_VALUE = X'081A';                       /* REQUEST SEQUENCE ERROR */
  .   .     .   ELSE
  .   .     .     DO;                                           /* SESSION EXIST ±  */
  .   .     .     . IF NAU_SESSION_COUNT(P->NRCB.ELEMENT_ADDRESS) ¬= 0 THEN
  .   .     .     .                                             /* APPENDIX B       */
  .   .     .     .   RETURN_VALUE = X'0809';                   /* MODE INCONSISTENCY */
  .   .     .     . FIND P1->NRCB IN NRCB_LIST                  /* BF.LUS EXIST ?   */
  .   .     .     .      WHERE(P1->NRCB.RESOURCE_TYPE = BF.LU &
  .   .     .     .      P1->NRCB.ASSOCIATED_RESOURCE = P->NRCB.ELEMENT_ADDRESS);
  .   .     .     . IF P1 ¬= NULL THEN
  .   .     .     .   RETURN_VALUE = X'0809';                   /* MODE INCONSISTENCY */
  .   .     .     END;
  .   .     END;
  .   SCANEND;
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │  FREE ALL BF.LU'S ASSOCIATED WITH A BF.PU, ALL  │
                  │  PRIMARY LU'S ASSOCIATED WITH  A SECONDARY LU,  │
                  │  OR  ALL  LU'S  ASSOCIATED WITH  A  PU.   EACH  │
                  │  ELEMENT ADDRESS TO BE FREED IS CHECKED TO SEE  │
                  │  THAT IT WAS NOT ASSIGNED BY ANOTHER CP AND NO  │
                  │  SESSION EXISTS WITH THE RESOURCE.              │
                  └──────────────────────────────────────────┘
                                                                                */
  . WHEN((NRCB.RESOURCE_CATEGORY = (BF.PU | ALS | PU | LU)) &
  .      (FNA_RQ.ENTRY_CNT = ALL))
  .   SCAN NRCB_LIST PTR(NRCB_PTR) WHILE(RETURN_VALUE = 0);
  .   . IF NRCB.ASSOCIATED_RESOURCE = TARGET_ELEMENT &
  .   .      (NRCB.RESOURCE_TYPE = BF.LU |
  .   .      NRCB.RESOURCE_TYPE = LU) THEN
  .   .   IF NRCB.ASSIGNING_CP_SCB_ID ¬= NULL &
  .   .        NRCB.ASSIGNING_CP_SCB_ID ¬= SCB_PTR THEN
  .   .     RETURN_VALUE = X'081A';                           /* REQUEST SEQUENCE ERROR */
  .   .
  .   .   ELSE
  .   .     IF NAU_SESSION_COUNT(NRCB.ELEMENT_ADDRESS) ¬= 0 THEN /* APPENDIX B     */
  .   .       RETURN_VALUE = X'0809';                         /* MODE INCONSISTENCY */
  .   SCANEND;
```

```
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │        FREE A SPECIFIC ALS AND BF.PU │
                    └──────────────────────────────────────┘
                                                                        */
  . WHEN((NRCB.RESOURCE_CATEGORY = LINK) & (FNA_RQ.ENTRY_CNT ¬= ALL))
  .   DO I = 1 TO FNA_RQ.ENTRY_CNT WHILE(RETURN_VALUE = 0);
  .   . FIND NRCB IN NRCB_LIST                                  /* LOCATE ALS RESOURCE      */
  .   .      WHERE(NRCB.ELEMENT_ADDRESS = FNA_RQ.SUBFIELD(I) &
  .   .      NRCB.RESOURCE_TYPE = ALS);
  .   . IF NRCB_PTR = NULL THEN
  .   .   RETURN_VALUE = X'0806';                               /* RESOURCE UNKNOWN         */
  .   . ELSE
  .   .                                  /* VERIFY THAT ALS IS RESET                      */
  .   .   IF ALS_SEC_SUBTREE_CHECK(NRCB.ELEMENT_ADDRESS) = NG THEN /* PAGE 11-97           */
  .   .     RETURN_VALUE = X'0809';                             /* MODE INCONSISTENCY       */
  .   .   ELSE
  .   .     DO;
  .   .     . FIND P->NRCB IN NRCB_LIST                         /* LOCATE BF.PU RESOURCE    */
  .   .     .      WHERE(P->NRCB.RESOURCE_TYPE = BF.PU &
  .   .     .      P->NRCB.ELEMENT_ADDRESS = NRCB.ELEMENT_ADDRESS);
  .   .     . IF NRCB_PTR ¬= NULL THEN     /* ASSIGNED BY ANOTHER CP?                       */
  .   .     .   IF P->NRCB.ASSIGNING_CP_SCB_ID ¬= NULL &
  .   .     .        P->NRCB.ASSIGNING_CP_SCB_ID ¬= SCB_PTR THEN
  .   .     .     RETURN_VALUE = X'081A';                       /* REQUEST SEQUENCE ERROR   */
  .   .     .
  .   .     .   ELSE
  .   .     .     DO;                                           /* DO ANY SESSIONS EXIST    */
  .   .     .     . IF (NAU_SESSION_COUNT(P->NRCB.ELEMENT_ADDRESS) ¬= 0) THEN
  .   .     .     .                                             /* APPENDIX B               */
  .   .     .     .   RETURN_VALUE = X'0809';                   /* MODE INCONSISTENCY       */
  .   .     .     . FIND P1->NRCB IN NRCB_LIST                  /* ANY BF.LU'S EXIST?       */
  .   .     .     .      WHERE(P1->NRCB.RESOURCE_TYPE = BF.LU &
  .   .     .     .      P1->NRCB.ASSOCIATED_RESOURCE = P->NRCB.ELEMENT_ADDRESS);
  .   .     .     . IF P1 ¬= NULL THEN
  .   .     .     .   RETURN_VALUE = X'0809';                   /* MODE INCONSISTENCY       */
  .   .     .     END;
  .   .     END;
  .   END;
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │ FREE SPECIFIC BF.LU'S OR LU'S. CHECK THAT │
                    │ THE RESOURCE EXISTS, WAS NOT ASSIGNED BY  │
                    │ ANOTHER CP, AND CURRENTLY HAS NO SESSIONS.│
                    └──────────────────────────────────────┘
                                                                        */
  . WHEN((NRCB.RESOURCE_CATEGORY = (BF.PU | ALS | PU | LU)) &
  .      (FNA_RQ.ENTRY_CNT ¬= ALL))
  .   DO I = 1 TO FNA_RQ.ENTRY_CNT WHILE(RETURN_VALUE = 0);
  .   . NRCB_PTR = LOCATE_NODE_RESOURCE(FNA_RQ.SUBFIELD(I));    /* APPENDIX B               */
  .   . IF NRCB_PTR = NULL |
  .   .      (NRCB.RESOURCE_CATEGORY ¬= BF.LU &
  .   .      NRCB.RESOURCE_CATEGORY ¬= LU) THEN
  .   .   RETURN_VALUE = X'0806';                               /* RESOURCE UNKNOWN         */
  .   . ELSE
  .   .   IF NRCB.ASSIGNING_CP_SCB_ID ¬= NULL &
  .   .        NRCB.ASSIGNING_CP_SCB_ID ¬= SCB_PTR THEN
  .   .     RETURN_VALUE = X'081A';                             /* REQUEST SEQUENCE ERROR   */
  .   .
  .   .   ELSE
  .   .     IF NAU_SESSION_COUNT(NRCB.ELEMENT_ADDRESS) ¬= 0 THEN /* APPENDIX B             */
  .   .       RETURN_VALUE = X'0809';                           /* MODE INCONSISTENCY       */
  .   END;
  END;

  RETURN(RETURN_VALUE);
END FNA_VALIDITY_CHECK;
```

```
FNA_BF_PU_AND_ALS_PROC: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  REMOVES THE ENTRIES  FOR THE BF.PU AND THE  ASSOCIATED ADJACENT LINK │
   │                 STATIONS FROM THE NODE RESOURCE LIST.                │
   │                                                                      │
   │      INPUT:     THE FNA REQUEST, WHICH IS THE CURRENT MESSAGE UNIT    │
   │                                                                      │
   │      OUTPUT:    NONE                                                 │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 NS.FNA_PROC                          PAGE 11-55      │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL ALS_EA BIT(16);
  DCL LINK_EA BIT(16);
  DCL P POINTER;
  DCL I FIXED BIN;

  SELECT ANYORDER(FNA_RQ.ENTRY_CNT);
  .
  . WHEN(ALL)
  .   SCAN NRCB_LIST PTR(NRCB_PTR);
  .   .
  .   . IF NRCB.ASSOCIATED_RESOURCE = FNA_RQ.TARGET_ADDRESS &
  .   .      NRCB.RESOURCE_CATEGORY = ALS THEN
  .   .   DO;
  .   .   . ALS_EA = NRCB.ELEMENT_ADDRESS;
  .   .   .
  .   .   . REMOVE NRCB FROM NRCB_LIST DISCARD;
  .   .   .
  .   .   . SCAN NRCB_LIST PTR(P);
  .   .   . . IF P->NRCB.ASSOCIATED_RESOURCE = ALS_EA &
  .   .   . .      P->NRCB.RESOURCE_CATEGORY = BF.PU THEN
  .   .   . .   REMOVE P->NRCB FROM NRCB_LIST DISCARD;
  .   .   . SCANEND;
  .   .   .
  .   .   END;
  .   SCANEND;
  .
  . WHEN(¬ALL)
  .   DO I = 1 TO FNA_RQ.ENTRY_CNT;
  .   .
  .   . P = LOCATE_NODE_RESOURCE(FNA_RQ.SUBFIELD(I));          /* APPENDIX B          */
  .   . IF P ¬= NULL THEN
  .   .   DO;
  .   .   . ALS_EA = P->NRCB.ASSOCIATED_RESOURCE;
  .   .   .
  .   .   . REMOVE P->NRCB FROM NRCB_LIST DISCARD;
  .   .   .
  .   .   . P = LOCATE_NODE_RESOURCE(ALS_EA);                 /* APPENDIX B          */
  .   .   . IF P ¬= NULL THEN
  .   .   .
  .   .   .   REMOVE P->NRCB FROM NRCB_LIST DISCARD;
  .   .   .
  .   .   END;
  .   END;
  END;

  RETURN;
END FNA_BF_PU_AND_ALS_PROC;
```

```
FNA_BF_LU_PROC: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  REMOVES ENTRIES FOR BF.LU NODE RESOURCES FROM THE NRCB_LIST. │
  │                                                                              │
  │      INPUT:     THE FNA REQUEST, WHICH IS THE CURRENT MESSAGE UNIT           │
  │                                                                              │
  │      OUTPUT:    NONE                                                         │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 NS.FNA_PROC                          PAGE 11-55              │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL P POINTER;
  DCL I FIXED BIN;

  SELECT ANYORDER(FNA_RQ.ENTRY_CNT);
  .
  . WHEN(ALL)
  .   SCAN NRCB_LIST PTR(NRCB_PTR);
  .   . IF NRCB.ASSOCIATED_RESOURCE = FNA_RQ.TARGET_ADDRESS &
  .   .      NRCB.RESOURCE_CATEGORY = BF.LU THEN
  .   .   REMOVE NRCB FROM NRCB_LIST DISCARD;
  .   SCANEND;
  .
  . WHEN(¬ALL)
  .   DO I = 1 TO FNA_RQ.ENTRY_CNT;
  .   . P = LOCATE_NODE_RESOURCE(FNA_RQ.SUBFIELD(I));          /* APPENDIX B      */
  .   . IF P->NRCB.RESOURCE_CATEGORY = BF.LU THEN
  .   .   REMOVE P->NRCB FROM NRCB_LIST DISCARD;
  .   END;
  END;

  RETURN;
END FNA_BF_LU_PROC;
```

```
FNA_LU_PROC: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  REMOVES LU NODE RESOURCES FROM THE NRCB_LIST.                │
  │                                                                              │
  │      INPUT:     THE FNA REQUEST THAT IS THE CURRENT MESSAGE UNIT             │
  │                                                                              │
  │      OUTPUT:    THE REQUESTED LU'S ARE REMOVED FROM THE NRCB_LIST AND DISCARDED.
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 NS.FNA_PROC                          PAGE 11-55              │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL P POINTER;
  DCL I FIXED BIN;

  SELECT ANYORDER(FNA_RQ.ENTRY_CNT);
  .
  . WHEN(ALL)
  .   SCAN NRCB_LIST PTR(NRCB_PTR);
  .   . IF NRCB.ASSOCIATED_RESOURCE = FNA_RQ.TARGET_ADDRESS &
  .   .      NRCB.RESOURCE_CATEGORY = LU THEN
  .   .   REMOVE NRCB FROM NRCB_LIST DISCARD;
  .   SCANEND;
  .
  . WHEN(¬ALL)
  .   DO I = 1 TO FNA_RQ.ENTRY_CNT;
  .   .
  .   . P = LOCATE_NODE_RESOURCE(FNA_RQ.SUBFIELD(I));          /* APPENDIX B      */
  .   . REMOVE P->NRCB FROM NRCB_LIST DISCARD;
  .   .
  .   END;
  .
  END;
  RETURN;
END FNA_LU_PROC;
```

```
NS.BF_LU_ADD: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  ADDS NODE RESOURCE ENTRIES FOR  BF.LU'S TO THE NRCB_LIST; ADDITIONAL │
  │                 PARAMETERS ARE ADDED LATER BY SETCV.                         │
  │                                                                              │
  │      INPUT:     THE RNAA REQUEST, WHICH IS THE CURRENT MESSAGE UNIT           │
  │                                                                              │
  │      OUTPUT:    THE +RSP CONTAINING THE ADDRESSES ASSIGNED TO ALL REQUESTED ENTRIES │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 NS.RNAA_PROC                    PAGE 11-52                   │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 UPM_2_BYTE_NA_ASSIGN            PAGE 11-117                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL ASSIGNED_ADDR BIT(16);
  DCL I FIXED BIN;

  DO I = 1 TO RNAA_RQ.ENTRY_CNT;
  .
  . ASSIGNED_ADDR = 0;
  . SCAN NRCB_LIST PTR(NRCB_PTR) WHILE(ASSIGNED_ADDR = 0);
  . . IF NRCB.ASSOCIATED_RESOURCE = RNAA_RQ.TARGET_ADDRESS &
  . .       NRCB.RESOURCE_CATEGORY = BF.LU &
  . .       NRCB.BF_LOCAL_ID = RNAA_RQ.SUBFIELD(I) THEN
  . .    ASSIGNED_ADDR = NRCB.ELEMENT_ADDRESS;
  . .
  . SCANEND;
  .
  . IF ASSIGNED_ADDR ¬= 0 THEN
  .    DO;
  .    . RNAA_RQ.SUBFIELD(I) = ASSIGNED_ADDR;
  .    . NRCB.ASSIGNING_CP_SCB_ID = SCB_PTR;
  .    END;
  .
  . ELSE
  .    DO;
  .    . CREATE NRCB PTR(NRCB_PTR);
  .    . NRCB.RESOURCE_CATEGORY = BF.LU;
  .    . NRCB.ASSOCIATED_RESOURCE = RNAA_RQ.TARGET_ADDRESS;
  .    . NRCB.BF_LOCAL_ID = RNAA_RQ.SUBFIELD(I,8:15);          /* BYTE 2          */
  .    . NRCB.ASSIGNING_CP_SCB_ID = SCB_PTR;
  .    . RNAA_RQ.SUBFIELD(I) = UPM_2_BYTE_NA_ASSIGN;           /* PAGE 11-117     */
  .    . NRCB.ELEMENT_ADDRESS = RNAA_RQ.SUBFIELD(I) & NCB.NODE_ELEMENT_MASK;
  .    .                          /* RETURN ELEMENT ADDRESS IN THE RU             */
  .    . INSERT NRCB IN NRCB_LIST;
  .    END;
  .
  END;
  CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);                        /* APPENDIX B      */
  RETURN;
END NS.BF_LU_ADD;
```

```
NS.BF_PU_AND_ALS_ADD: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  ADDS NRCB  ENTRIES FOR THE  BF.PU AND  ADJACENT LINK STATION  TO THE
  │                 NRCB_LIST, IF REQUIRED.                                      │
  │                                                                              │
  │      INPUT:     THE RNAA REQUEST, WHICH IS THE CURRENT MESSAGE UNIT           │
  │                                                                              │
  │      OUTPUT:    +RSP CONTAINING THE ADDRESSES ASSIGNED TO ALL ENTRIES        │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 NS.RNAA_PROC                        PAGE 11-52               │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 UPM_2_BYTE_NA_ASSIGN                PAGE 11-117              │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL ALS_EA BIT(16);
  DCL ALS_NA BIT(16);
  DCL I FIXED BIN;

  IF RNAA_RQ.ENTRY_CNT > 0 THEN
    DO I = 1 TO RNAA_RQ.ENTRY_CNT;
    .
    . FIND NRCB IN NRCB_LIST
    .      WHERE(NRCB.ELEMENT_ADDRESS = RNAA_RQ.TARGET_ADDRESS &
    .      NRCB.RESOURCE_CATEGORY = ALS &
    .      NRCB.ALS_DLC_HDR_ADDR = RNAA_RQ.SUBFIELD(I));
    . IF NRCB_PTR ¬= NULL THEN
    .   DO;
    .   . ALS_EA = NRCB.ELEMENT_ADDRESS;
    .   . RNAA_RQ.SUBFIELD(I) = ALS_EA;
    .   . NRCB.ASSIGNING_CP_SCB_ID = SCB_PTR;
    .   END;
    .
    . ELSE
    .   DO;
    .   . ALS_NA = UPM_2_BYTE_NA_ASSIGN;                    /* PAGE 11-117    */
    .   . ALS_EA = ALS_NA & NCB.NODE_ELEMENT_MASK;
    .   . CREATE NRCB PTR(NRCB_PTR);                        /* FOR ALS_EA     */
    .   . NRCB.RESOURCE_CATEGORY = ALS;
    .   . NRCB.ASSOCIATED_RESOURCE = RNAA_RQ.TARGET_ADDRESS;
    .   . NRCB.ALS_DLC_HDR_ADDR = RNAA_RQ.SUBFIELD(I);
    .   . NRCB.ASSIGNING_CP_SCB_ID = SCB_PTR;
    .   . NRCB.ELEMENT_ADDRESS = ALS_EA;
    .   . INSERT NRCB IN NRCB_LIST;
    .   END;
    . FIND NRCB IN NRCB_LIST
    .      WHERE(NRCB.ELEMENT_ADDRESS = ALS_EA &
    .      NRCB.RESOURCE_CATEGORY = BF.PU);
    . IF NRCB_PTR ¬= NULL THEN
    .   NRCB.ASSIGNING_CP_SCB_ID = SCB_PTR;
    . ELSE
    .   DO;
    .   . CREATE NRCB PTR(NRCB_PTR);                        /* FOR BF.PU      */
    .   .
    .   . NRCB.RESOURCE_CATEGORY = BF.PU;
    .   . NRCB.ASSOCIATED_RESOURCE = ALS_EA;
    .   . NRCB.BF_LOCAL_ID = 0;
    .   . NRCB.ASSIGNING_CP_SCB_ID = SCB_PTR;
    .   . RNAA_RQ.SUBFIELD(I) = ALS_NA;
    .   . NRCB.ELEMENT_ADDRESS = RNAA_RQ.SUBFIELD(I);
    .   . INSERT NRCB IN NRCB_LIST;
    .   END;
    .
    END;
  CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);                      /* APPENDIX B     */
  RETURN;
END NS.BF_PU_AND_ALS_ADD;
```

```
NS.LU_ADD: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  ADDS NODE RESOURCE ENTRIES FOR LU'S TO THE NRCB_LIST.        │
  │                                                                              │
  │      INPUT:     THE RNAA REQUEST, WHICH IS THE CURRENT MESSAGE UNIT          │
  │                                                                              │
  │      OUTPUT:    THE +RSP TO RNAA REQUEST                                     │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 NS.RNAA_PROC                        PAGE 11-52               │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 UPM_2_BYTE_NA_ASSIGN                PAGE 11-117              │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  CREATE NRCB PTR(NRCB_PTR);

  RNAA_RQ.ENTRY_CNT = 1;
  RNAA_RQ.SUBFIELD(1) = UPM_2_BYTE_NA_ASSIGN;                  /* PAGE 11-117    */

  NRCB.RESOURCE_CATEGORY = LU;
  NRCB.ASSOCIATED_RESOURCE = RNAA_RQ.TARGET_ADDRESS;
  NRCB.ASSIGNING_CP_SCB_ID = SCB_PTR;
  NRCB.ELEMENT_ADDRESS = RNAA_RQ.SUBFIELD(1) & NCB.NODE_ELEMENT_MASK;
  INSERT NRCB IN NRCB_LIST;

  CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);                         /* APPENDIX B     */

  RETURN;
END NS.LU_ADD;



NS.ADDLINK_ADDLINKSTA_PROC: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  RETURNS  THE REQUESTED  NETWORK ADDRESS  IN  THE POSITIVE  RESPONSE;
  │                 RETURNS A NEGATIVE  RESPONSE IF THE SPECIFIC  RESOURCE IDENTIFIER IS
  │                 UNKNOWN, IF THERE ARE INSUFFICIENT RESOURCES  TO ACT ON THE REQUEST,
  │                 OR IF THE PARAMETERS ARE INVALID.                            │
  │                                                                              │
  │      INPUT:     ADDLINK AND ADDLINKSTA REQUESTS FROM SNS.RCV (CHAPTER 6)      │
  │                                                                              │
  │      OUTPUT:    +RSP(ADDLINK);      -RSP(ADDLINK,0806|0812);       +RSP(ADDLINKSTA);
  │                 -RSP(ADDLINKSTA,0806|0812|0835)                              │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 NS.CS_RCV                           PAGE 11-34               │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 UPM_ADDLINK                         PAGE 11-112              │
  │                 UPM_ADDLINKSTA                      PAGE 11-112              │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SELECT ANYORDER(RQ_CODE);
  .
  . WHEN(ADDLINK)
  .   DO;
  .   . CALL UPM_ADDLINK;                                    /* PAGE 11-112    */
  .   . SEND MU TO SNS.SEND;                                 /* CHAPTER 6      */
  .   END;
  .
  . WHEN(ADDLINKSTA)
  .   DO;
  .   . CALL UPM_ADDLINKSTA;                                 /* PAGE 11-112    */
  .   . SEND MU TO SNS.SEND;                                 /* CHAPTER 6      */
  .   END;
  END;

  RETURN;
END NS.ADDLINK_ADDLINKSTA_PROC;
```

```
NS.DELETENR_PROC: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE VERIFIES THAT THE ADDRESS IN THE REQUEST IS VALID AND
  │                 THAT THE RESOURCE IS NOT BEING ACTIVELY USED.  IF THE CHECKS FAIL, A
  │                 -RSP IS RETURNED TO THE SENDER.
  │
  │      INPUT:     THE DELETENR RQ FROM SNS.RCV
  │
  │      OUTPUT:    +RSP(DELETENR); -RSP(DELETENR,0806|081A)
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 NS.CS_RCV                          PAGE 11-34
  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):
  │                 ALS_SEC_SUBTREE_CHECK              PAGE 11-97
  │                 FSM_LINK_ACT_RES                   PAGE 11-119
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  NRCB_PTR = LOCATE_NODE_RESOURCE(DELETENR_RQ.RESOURCE_ADDRESS); /* APPENDIX B   */

  IF NRCB_PTR = NULL THEN
    CALL CHANGE_MU_TO_NEG_RSP(X'0806');     /* APPENDIX B, RESOURCE UNKNOWN     */

  ELSE
    IF (NRCB.RESOURCE_CATEGORY = LINK &
         FSM_LINK_ACT_RES ¬= RESET) |                          /* PAGE 11-119   */
         (NRCB.RESOURCE_CATEGORY = ALS &
         ALS_SEC_SUBTREE_CHECK(NRCB.ELEMENT_ADDRESS) = NG) THEN  /* PAGE 11-97   */
      CALL CHANGE_MU_TO_NEG_RSP(X'081A');  /* APPENDIX B, REQUEST SEQUENCE ERROR */

    ELSE
      DO;
      . REMOVE NRCB FROM NRCB_LIST DISCARD;
      . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);                   /* APPENDIX B    */
      END;

  SEND MU TO SNS.SEND;                                         /* CHAPTER 6     */

  RETURN;
END NS.DELETENR_PROC;
```

```
NS.SETCV_PROC: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  PROCESSES  THE SETCV  REQUEST;  UPDATES THE  NODE  RESOURCE LIST  AS      │
│                 APPROPRIATE.                                             │
│                                                                          │
│      INPUT:     THE SETCV REQUEST, WHICH IS THE CURRENT MESSAGE UNIT      │
│                                                                          │
│      OUTPUT:    RSP TO SNS.SEND                                          │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 NS.CS_RCV                          PAGE 11-34            │
│                 NS.MS_PROC                         PAGE 11-107           │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 ALS_SEC_SUBTREE_CHECK              PAGE 11-97            │
│                 FSM_ALS_CONTACT_DISCONTACT_RES     PAGE 11-122           │
│                 UPM_SETCV_KEY1                     PAGE 11-116           │
│                 UPM_SETCV_KEY3                     PAGE 11-116           │
│                 UPM_SETCV_KEY4                     PAGE 11-116           │
│                 UPM_SETCV_KEY8                     PAGE 11-117           │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL RETURN_VALUE BIT(16);
DCL ALS_EA BIT(16);
DCL CV_PTR PTR;
DCL SETCV_RQ_KEY BIT(8) BASED(CV_PTR);
CV_PTR = ADDR(SETCV_RQ.CONTROL_VECTOR);

SELECT ANYORDER(SETCV_RQ_KEY);
.
. WHEN(KEY1)                                            /* DATE-TIME CONTROL VECTOR */
.   DO;
.   . CALL UPM_SETCV_KEY1;                              /* PAGE 11-116              */
.   . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);              /* APPENDIX B               */
.   . SEND MU TO SNS.SEND;                              /* CHAPTER 6                */
.   END;
.
. WHEN(KEY3)                          /* SDLC SECONDARY CONTROL VECTOR                */
.   DO;
.   . NRCB_PTR = LOCATE_NODE_RESOURCE(SETCV_RQ.TARGET_ADDRESS);
.   .                                                   /* APPENDIX B               */
.   . ALS_EA = NRCB.ELEMENT_ADDRESS;
.   . IF FSM_ALS_CONTACT_DISCONTACT_RES = PEND_RESET |   /* PAGE 11-122              */
.   .      FSM_ALS_CONTACT_DISCONTACT_RES = RESET_IN_PROGRESS THEN
.   .   CALL ENQUEUE_RU_FOR_RESOURCE(ALS_EA);           /* APPENDIX B               */
.   . ELSE
.   .   IF ALS_SEC_SUBTREE_CHECK(NRCB.ASSOCIATED_RESOURCE) ¬= OK THEN
.   .                                                   /* PAGE 11-97               */
.   .     DO;
.   .     . CALL CHANGE_MU_TO_NEG_RSP(X'0809'); /* APPENDIX B, MODE INCONSISTENCY     */
.   .     . SEND MU TO SNS.SEND;                        /* CHAPTER 6                */
.   .     END;
.   .
.   .   ELSE
.   .     DO;
.   .     . IF CV_PTR->CONTROL_VECTOR_TYPE_03.PU_TYPE = B'01' THEN
.   .     .   NRCB.RESOURCE_TYPE = PU_T2;
.   .     . IF CV_PTR->CONTROL_VECTOR_TYPE_03.PU_TYPE = B'10' THEN
.   .     .   NRCB.RESOURCE_TYPE = PU_T1;
.   .     . NRCB.LCP_RESET_OPTION =
.   .     .     CV_PTR->CONTROL_VECTOR_TYPE_03.LCP_RESET_OPTION;
.   .     . CALL UPM_SETCV_KEY3;                        /* PAGE 11-116              */
.   .     . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);        /* APPENDIX B               */
.   .     . SEND MU TO SNS.SEND;                        /* CHAPTER 6                */
.   .     END;
.   END;
```

```
. WHEN(KEY4)                                                    /* LU CONTROL VECTOR        */
.   DO;
.   . NRCB_PTR = LOCATE_NODE_RESOURCE(SETCV_RQ.TARGET_ADDRESS);
.   .                                                           /* APPENDIX B               */
.   . IF NAU_SESSION_COUNT(NRCB.ELEMENT_ADDRESS) ¬= 0 THEN
.   .                                                           /* APPENDIX B               */
.   .   CALL CHANGE_MU_TO_NEG_RSP(X'0809'); /* APPENDIX B, MODE INCONSISTENCY               */
.   .
.   . ELSE
.   .   DO;
.   .   . NRCB.SEC_RCV_PACING_CNT =
.   .   .       CV_PTR->CONTROL_VECTOR_TYPE_04.SECONDARY_RCV_PACING_CNT;
.   .   .                                   /* CPMGR RECEIVE PACING COUNT                   */
.   .   . CALL UPM_SETCV_KEY4;                                  /* PAGE 11-116              */
.   .   . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);                  /* APPENDIX B               */
.   .   END;
.   . SEND MU TO SNS.SEND;                                      /* CHAPTER 6                */
.   END;
.
. WHEN(KEY8)                                                    /* INTENSIVE MODE           */
.   DO;
.   . NRCB_PTR = LOCATE_NODE_RESOURCE(SETCV_RQ.TARGET_ADDRESS);
.   .                                                           /* APPENDIX B               */
.   . ALS_EA = NRCB.ELEMENT_ADDRESS;
.   . IF FSM_ALS_CONTACT_DISCONTACT_RES = PEND_RESET |          /* PAGE 11-122              */
.   .      FSM_ALS_CONTACT_DISCONTACT_RES = RESET_IN_PROGRESS THEN
.   .   CALL ENQUEUE_RU_FOR_RESOURCE(ALS_EA);                   /* APPENDIX B               */
.   . ELSE
.   .   DO;
.   .   . RETURN_VALUE = UPM_SETCV_KEY8;                       /* PAGE 11-117              */
.   .   . IF RETURN_VALUE ¬= 0 THEN
.   .   .   CALL CHANGE_MU_TO_NEG_RSP(RETURN_VALUE);           /* APPENDIX B               */
.   .   . ELSE
.   .   .   CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);               /* APPENDIX B               */
.   .   . SEND MU TO SNS.SEND;                                  /* CHAPTER 6                */
.   .   END;
.   END;
.
. OTHERWISE
.   DO;
.   . CALL CHANGE_MU_TO_NEG_RSP(X'0823'); /* APPENDIX B, UNKNOWN CONTROL VECTOR             */
.   . SEND MU TO SNS.SEND;                                      /* CHAPTER 6                */
.   END;
END;
                                                                                             /*
┌──────────────────────────────────────────────────────────────────────────────────────────┐
│      THESE UPM'S  RETAIN PARAMETERS,  AND APPROPRIATE  CHECKS ARE  MADE WITHIN  THIS     │
│      PROCEDURE  TO  POSITIVELY  RESPOND,  EXCEPT FOR  KEY  8;  UPM_SETCV_KEY(8)  MAY     │
│      GENERATE A NEGATIVE RESPONSE.                                                       │
└──────────────────────────────────────────────────────────────────────────────────────────┘
                                                                                             */
  RETURN;
END NS.SETCV_PROC;
```

```
NS.DLC_CONFIG: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE  IS CALLED BY  PU.SVC_MGR.NS.RCV TO ROUTE  INPUT FROM │
  │                 DLC FOR A LINK STATION WHOSE PRIMARY/SECONDARY ROLE IS CONFIGURABLE.  │
  │                                                                              │
  │      INPUT:     THE INPUT  SENT FROM LINK_MGR.   THE LSCB_PTR ADDRESSES  THE CORRECT  │
  │                 LSCB.                                                        │
  │                                                                              │
  │      OUTPUT:    THE MESSAGE UNIT TO THE APPROPRIATE PROCEDURE                │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 PU.SVC_MGR.NS.RCV                  PAGE 11-28                │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 FSM_ALS_CONTACT_DISCONTACT_RES     PAGE 11-122               │
  │                 NS.INOP_PROC                       PAGE 11-90                │
  │                 STATION_CONTACTED                  PAGE 11-72                │
  │                 XID_FORMAT_2_RCV                   PAGE 11-67                │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  SELECT ANYORDER;
  .
  . WHEN(¬(INPUT(SIGNAL)) & MUCB.XID = ON & XID.FORMAT = X'2')
  .   CALL XID_FORMAT_2_RCV;                                  /* PAGE 11-67     */
  .
  . WHEN(INPUT('CONTACTED'))
  .   DO;
  .   . CALL STATION_CONTACTED·                               /* PAGE 11-72     */
  .   . DISCARD MU;
  .   END;
  .
  . WHEN(INPUT('ALS_RESET_COMPLETE'))
  .   CALL FSM_ALS_CONTACT_DISCONTACT_RES('ALS_RESET_COMPLETE'); /* PAGE 11-122   */
  .
  . OTHERWISE
  .   CALL NS.INOP_PROC(LSCB.EA);                             /* PAGE 11-90     */
  END;
  RETURN;

END NS.DLC_CONFIG;
```

```
XID_FORMAT_2_RCV: PROCEDURE;

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE IS CALLED FROM NS.DLC_CONFIG  TO HANDLE AN XID FORMAT
 │                 2 THAT HAS BEEN RECEIVED.  CHECKS ON THE CONSISTENCY AND CORRECTNESS
 │                 OF  PARAMETERS  WITHIN  THE  XID,  AND  ON  THE  AGREEMENT  BETWEEN
 │                 PARAMETERS IN THE XID AND THE TGCB, ARE PERFORMED.  FSM_XID_FORMAT_2
 │                 AND FSM_TGN ARE CALLED.  IF NEITHER THE PARAMETER CHECKS NOR THE FSM
 │                 CALLS FIND  AN ERROR,  THEN THIS  XID IS  ACCEPTED.  IF  FSM_TGN AND
 │                 FSM_XID_FORMAT_2 INDICATE  THAT THERE IS  AGREEMENT BETWEEN  THE TWO
 │                 STATIONS ON TG NUMBER AND ON  PRIMARY AND SECONDARY STATUS, THEN THE
 │                 PROCEDURE SUCCESSFUL_XID_EXCHANGE IS CALLED  TO COMPLETE THE CONTACT
 │                 PROCEDURE.  IF  AGREEMENT HAS NOT YET  BEEN REACHED ON TG  NUMBER OR
 │                 PRIMARY AND SECONDARY STATUS, AN XID IS BUILT AND SENT.
 │
 │      INPUT:     THE CURRENT MESSAGE UNIT IS XID FORMAT 2 FROM THE CALLING PROCEDURE.
 │                 THE LSCB_PTR ADDRESSES THE CORRECT LSCB.
 │
 │      OUTPUT:    XID TO LINK_MGR
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 FSM_TGN                          PAGE 11-125
 │                 NS.DLC_CONFIG                    PAGE 11-66
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 FSM_TGN                          PAGE 11-125
 │                 FSM_XID_FORMAT_2                 PAGE 11-126
 │                 MULTI_LINK_TESTS                 PAGE 11-70
 │                 SUCCESSFUL_XID_EXCHANGE          PAGE 11-72
 │                 XID_ERR_RCV                      PAGE 11-73
 │                 XID_ERR_SEND                     PAGE 11-75
 │                 XID_FORMAT_CHECK_1               PAGE 11-68
 │                 XID_FORMAT_CHECK_2               PAGE 11-69
 │                 XID_FORMAT_2_BUILD               PAGE 11-71
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL XID BASED(ADDR(RU)) LIKE LSCB.XID_SEND;

  LSCB.XID_RCV = XID;
  IF XID_FORMAT_CHECK_1 = NG THEN                          /* PAGE 11-68        */
    RETURN;

  IF XID_2.ERROR_STATUS ¬= X'0' THEN                       /* ERROR             */
    DO;
    . CALL XID_ERR_RCV;                                    /* PAGE 11-73        */
    . RETURN;
    END;

  IF XID_FORMAT_CHECK_2 = NG THEN                          /* PAGE 11-69        */
    RETURN;

  CALL FSM_XID_FORMAT_2;                                   /* PAGE 11-126       */
  IF LSCB.XID_SEND.ERROR_STATUS ¬= X'0' THEN               /*  ERROR            */
    DO;
    . CALL XID_ERR_SEND;                                   /* PAGE 11-75        */
    . RETURN;
    END;

  CALL FSM_TGN;                                            /* PAGE 11-125       */
  IF LSCB.XID_SEND.ERROR_STATUS ¬= X'0' THEN               /* ERROR             */
    DO;
    . CALL XID_ERR_SEND;                                   /* PAGE 11-75        */
    . RETURN;
    END;

  TGCB_PTR = LSCB.TGCBPTR;
  IF (XID_2.TGN ¬= 0) & (XID_2.MULTI_LINK ¬= TGCB.MULTI_LINK_SUPP) THEN
    DO;
    . LSCB.XID_SEND.ERROR_STATUS = EXCHANGED_PARMS_INCOMPAT;
    . LSCB.CONTACTED_STATUS = INCOMPATIBLE_STATIONS;
    . CALL XID_ERR_SEND;                                   /* PAGE 11-75        */
    END;
  ELSE
    IF  ¬EMPTY(TGCB.ASSOC_LSCB_LIST) THEN
      CALL MULTI_LINK_TESTS;                               /* PAGE 11-70        */

  IF LSCB.XID_SEND.ERROR_STATUS = X'0' THEN                /* NO ERROR          */
    DO;
    . DISCARD MU;                                          /* DISCARD XID       */
    . IF FSM_TGN = MATCH &                                 /* PAGE 11-125       */
    .       (FSM_XID_FORMAT_2 = PEND_ACT_PRI_2 |
    .       FSM_XID_FORMAT_2 = PEND_ACT_SEC_1) THEN        /* PAGE 11-126       */
    .   CALL SUCCESSFUL_XID_EXCHANGE;                      /* PAGE 11-72        */
    . ELSE
    .   CALL XID_FORMAT_2_BUILD;                           /* PAGE 11-71        */
    END;

  RETURN;
END XID_FORMAT_2_RCV;
```

```
XID_FORMAT_CHECK_1: PROCEDURE RETURNS(BIT(1));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE IS  CALLED FROM XID_FORMAT_2_RCV TO  CHECK THE FORMAT │
   │                 OF A RECEIVED XID.  THESE FORMAT CHECKS ENSURE THAT THE RECEIVED XID │
   │                 FORMAT 2 IS  COMPLETE ENOUGH THAT FURTHER CHECKING OF  THE VALUES IN │
   │                 THE XID FIELDS IS WORTHWHILE.                                │
   │                                                                              │
   │      INPUT:     THE CURRENT  MESSAGE UNIT IS  AN XID  FORMAT 2 COMMAND  OR RESPONSE. │
   │                 LSCB_PTR ADDRESSES THE CORRECT LSCB.                         │
   │                                                                              │
   │      OUTPUT:    RC OF OK OR NG TO THE CALLING PROCEDURE                      │
   │                                                                              │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
   │                 XID_FORMAT_2_RCV                    PAGE 11-67               │
   │                                                                              │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
   │                 XID_ERR_SEND                        PAGE 11-75               │
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);

  RC = OK;
  IF (XID.PU_TYPE ¬= SUBAREA_NODE) |
       (XID_2.FID_4_SUPPORTED ¬= SUPPORTED) |
       (XID_2.DLC_TYPE ¬= (SDLC | CHAN370)) |
       (XID_2.DLC_TYPE = SDLC & MUCB.XID_LENGTH ¬= 43) |
       (XID_2.DLC_TYPE = CHAN370 & MUCB.XID_LENGTH ¬= 41) THEN
    DO;
    . LSCB.XID_SEND.ERROR_STATUS = EXCHANGED_PARMS_INCOMPAT;
    . LSCB.CONTACTED_STATUS = INCOMPATIBLE_STATIONS;
    . CALL XID_ERR_SEND;                                    /* PAGE 11-75      */
    . RC = NG;
    END;

  RETURN(RC);

END XID_FORMAT_CHECK_1;
```

```
XID_FORMAT_CHECK_2: PROCEDURE RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE IS  CALLED FROM XID_FORMAT_2_RCV TO  CHECK THE FORMAT
  │                 OF A RECEIVED XID COMMAND OR RESPONSE.
  │
  │      INPUT:     XID IS THE CURRENT MESSAGE UNIT.  THE LSCB_PTR ADDRESSES THE CORRECT
  │                 LSCB.
  │
  │      OUTPUT:    RETURN CODE OK OR NG TO THE CALLING PROCEDURE
  │
  │      NOTE:      THE CMD_SENDER  IS THE PRIMARY LINK  STATION; THE RSP_SENDER  IS THE
  │                 SECONDARY LINK STATION.
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 XID_FORMAT_2_RCV                    PAGE 11-67
  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):
  │                 UPM_CHAN370_CHECK                   PAGE 11-113
  │                 XID_ERR_SEND                        PAGE 11-75
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);

  RC = OK;
  SELECT ANYORDER(LSCB.DLC_TYPE);
  .
  . WHEN(SDLC)
  .   IF (XID_2.DLC_TYPE ¬= SDLC) |
  .        (XID_2_SDLC.STA_ROLE_PRI = NO & XID_2.CONTACT_OR_LOAD_STAT = CMD_SENDER) |
  .        (XID_2_SDLC.STA_ROLE_SEC = NO & XID_2.CONTACT_OR_LOAD_STAT = RSP_SENDER) |
  .        (XID_2_SDLC.STA_ROLE_SEC = YES & XID_2_SDLC.SDLC_INIT_RCV =  ¬SIM) |
  .                                                    /* SIM CHECK IS OPTIONAL    */
  .        (XID_2_SDLC.STA_ROLE_PRI = NO & XID_2_SDLC.STA_ROLE_SEC = NO) |
  .        XID_2_SDLC.STA_XMIT_RCV_CAP ¬= LSCB.LOCAL_STATION.STA_XMT_RCV_CAP |
  .        XID_2_SDLC.CMD_RSP_PROFILE ¬= SNA_LINK THEN
  .     DO;
  .     . LSCB.XID_SEND.ERROR_STATUS = EXCHANGED_PARMS_INCOMPAT;
  .     . LSCB.CONTACTED_STATUS = INCOMPATIBLE_STATIONS;
  .     . CALL XID_ERR_SEND;                          /* PAGE 11-75               */
  .     . RC = NG;
  .     END;
  .
  . WHEN(CHAN370)
  .   DO;
  .   . IF (XID_2.DLC_TYPE ¬= CHAN370) |
  .   .      (LSCB.LOCAL_STATION.STATION_TYPE = SECONDARY & XID_2_CHL.INIT_BUFFS = X'00') |
  .   .      (UPM_CHAN370_CHECK = NG) THEN              /* PAGE 11-113              */
  .   .   DO;
  .   .   . LSCB.XID_SEND.ERROR_STATUS = EXCHANGED_PARMS_INCOMPAT;
  .   .   . LSCB.CONTACTED_STATUS = INCOMPATIBLE_STATIONS;
  .   .   . CALL XID_ERR_SEND;                        /* PAGE 11-75               */
  .   .   . RC = NG;
  .   .   END;
  .   END;
  END;

  RETURN(RC);

END XID_FORMAT_CHECK_2;
```

```
MULTI_LINK_TESTS: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE IS CALLED FROM  XID_FORMAT_2_RCV WHEN XID IS RECEIVED
   │                 FOR  A LINK  THAT IS  ASSIGNED TO  A TG  FOR WHICH  ANOTHER LINK  IS
   │                 CURRENTLY ACTIVE.  AN XID  INDICATING AN ERROR IS SENT IF  THE TG IS
   │                 NOT A MULTIPLE-LINK TG, IF THE  RECEIVED XID INDICATES THAT THE LINK
   │                 IS NOT TO BE  ASSIGNED TO A MULTIPLE-LINK TG, OR  IF THE MAXIMUM BTU
   │                 LENGTH  SUPPORTED BY  THE LINK  IS NOT  AT  LEAST AS  LARGE AS  THAT
   │                 SUPPORTED BY THE TG.
   │
   │      INPUT:     THE  RECEIVED  XID  IS  THE  CURRENT  MESSAGE  UNIT.   THE  TGCB_PTR
   │                 ADDRESSES  THE CORRECT  TGCB.  THE  LSCB_PTR  ADDRESSES THE  CORRECT
   │                 LSCB.
   │
   │      OUTPUT:    NONE
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 XID_FORMAT_2_RCV                    PAGE 11-67
   │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):
   │                 XID_ERR_SEND                        PAGE 11-75
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF (TGCB.MULTI_LINK_SUPP | XID_2.MULTI_LINK) =  ¬SUPPORTED THEN
    DO;
    . LSCB.XID_SEND.ERROR_STATUS = MULTI_OR_DLC_INCOMPATIBLE;
    . LSCB.CONTACTED_STATUS = INCOMPATIBLE_STATIONS;
    . CALL XID_ERR_SEND;                                  /* PAGE 11-75      */
    END;
  ELSE
    DO;
    . IF (XID_2_SDLC.MAX_RECEIVABLE_I_FIELD < TGCB.MAX_SEND_BTU_LENGTH) THEN
    .   DO;
    .   . LSCB.XID_SEND.ERROR_STATUS = SUBSEQUENT_LINK_PARMS_INCOMPAT;
    .   . LSCB.CONTACTED_STATUS = INCOMPATIBLE_WITH_TG;
    .   . CALL XID_ERR_SEND;                              /* PAGE 11-75      */
    .   END;
    END;

  RETURN;

END MULTI_LINK_TESTS;
```

```
XID_FORMAT_2_BUILD: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  BUILDS XID FORMAT 2 TO BE SENT TO AN ADJACENT LINK STATION.
   │
   │      INPUT:     THE LSCB_PTR ADDRESSES THE CORRECT LSCB.  IF THE STATION IS ASSIGNED
   │                 TO A TG, THE TGCB_PTR ADDRESSES THE CORRECT TGCB.
   │
   │      OUTPUT:    XID FORMAT 2 TO THE CALLING ROUTINE
   │
   │      NOTE:      INPUT TO THE FSM IS (S, CMD_SENDER, ¬ERR) OR (S, RSP_SENDER, ¬ERR).
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 CONTACT_CONFIG                  PAGE 11-44
   │                 SUCCESSFUL_XID_EXCHANGE         PAGE 11-72
   │                 XID_FORMAT_2_RCV                PAGE 11-67
   │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):
   │                 FSM_XID_FORMAT_2                PAGE 11-126
   │                 UPM_BUILD_FORMAT_2_XID          PAGE 11-113
   │                 UPM_PRI_SEC_ROLE                PAGE 11-115
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL XID BASED(ADDR(RU)) LIKE LSCB.XID_SEND;

  CALL UPM_BUILD_FORMAT_2_XID;                          /* PAGE 11-113   */

  SELECT ANYORDER(LSCB.DLC_TYPE);
  . WHEN(SDLC)
  .   DO;
  .   . IF TGCB_PTR = NULL THEN
  .   .   XID_2.TG_STATUS = INACTIVE;
  .   . ELSE
  .   .   DO;
  .   .   . IF EMPTY(TGCB.ASSOC_LSCB_LIST) THEN
  .   .   .   XID_2.TG_STATUS = INACTIVE;
  .   .   . ELSE
  .   .   .   XID_2.TG_STATUS = ACTIVE;
  .   .   . XID_2.MULTI_LINK = TGCB.MULTI_LINK_SUPP;
  .   .   END;
  .   .
  .   . XID_2.TGN = LSCB.XID_SEND.TGN;
  .   . XID_2_SDLC.MAX_RECEIVABLE_I_FIELD = LSCB.LOCAL_STATION.MAX_BTU_LENGTH;
  .   . XID_2_SDLC.STA_XMIT_RCV_CAP = LSCB.LOCAL_STATION.STA_XMT_RCV_CAP;
  .   . XID_2.DLC_TYPE = SDLC;
  .   . MUCB.XID_LENGTH = 43;
  .   END;
  .
  . WHEN(CHAN370)
  .   DO;
  .   . XID_2.TG_STATUS = INACTIVE;
  .   . XID_2.MULTI_LINK =  ¬SUPPORTED;
  .   . XID_2.TGN = LSCB.XID_SEND.TGN;
  .   . XID_2.DLC_TYPE = CHAN370;
  .   . MUCB.XID_LENGTH = 41;
  .   END;
  END;

  XID.PU_TYPE = NCB.PU_TYPE;
  XID_2.FID_4_SUPPORTED = SUPPORTED;
  XID_2.ERROR_STATUS = 0;
  XID_2.CONTACT_OR_LOAD_STAT = LSCB.XID_SEND.CONTACT_OR_LOAD_STAT;
  XID_2_SDLC.STA_ROLE_SEC = YES;
  XID_2_SDLC.STA_ROLE_PRI = YES;
  CALL UPM_PRI_SEC_ROLE;                                /* PAGE 11-115   */
  LSCB.XID_SEND = XID;
  CALL FSM_XID_FORMAT_2;                                /* NOTE, PAGE 11-126 */
  MUCB.DIRECTION = SEND;
  SEND MU TO PU.SVC_MGR.LINK_MGR;

  RETURN;
END XID_FORMAT_2_BUILD;
```

```
SUCCESSFUL_XID_EXCHANGE: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE IS CALLED FROM XID_FORMAT_2_RCV WHEN XID EXCHANGE HAS │
  │                 BEEN SUCCESSFULLY COMPLETED.  IF THE LINK STATION IN  THIS NODE HAS │
  │                 ASSUMED THE SECONDARY  ROLE, AN XID IS SENT TO  THE PRIMARY STATION. │
  │                 IF THE STATION IN THIS NODE HAS  ASSUMED THE PRIMARY ROLE, NO XID IS │
  │                 SENT.  IN EITHER  CASE 'CONTACT' IS SENT TO THE  LINK_MGR TO TRIGGER │
  │                 THE LINK INITIALIZATION PROCEDURE.                   │
  │                                                                      │
  │      INPUT:     THE CURRENT MESSAGE UNIT IS XID FORMAT 2.             │
  │                                                                      │
  │      OUTPUT:    XID MU AND 'CONTACT' TO LINK_MGR                      │
  │                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
  │                 XID_FORMAT_2_RCV                 PAGE 11-67          │
  │                                                                      │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
  │                 FSM_XID_FORMAT_2                 PAGE 11-126         │
  │                 XID_FORMAT_2_BUILD               PAGE 11-71          │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  IF FSM_XID_FORMAT_2 = PEND_ACT_SEC_1 THEN                  /* PAGE 11-126   */
    CALL XID_FORMAT_2_BUILD;                                 /* PAGE 11-71    */

  SEND 'CONTACT' TO PU.SVC_MGR.LINK_MGR;

  RETURN;

END SUCCESSFUL_XID_EXCHANGE;



STATION_CONTACTED: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE IS CALLED FROM NS.DLC_CONFIG  WHEN A LINK STATION HAS │
  │                 BEEN CONTACTED.   IT CALLS SEND_CONTACTED TO SEND CONTACTED REQUESTS │
  │                 TO  ALL CP(S)  THAT ARE  IN THE  LINK STATION'S  CP_LIST.  THE  LINK │
  │                 STATION IS ADDED TO THE LIST OF ACTIVE LINK STATIONS ASSIGNED TO THE │
  │                 TG.                                                  │
  │                                                                      │
  │      INPUT:     THE CURRENT SIGNAL IS 'CONTACTED'  SENT FROM LINK_MGR.  THE LSCB_PTR │
  │                 ADDRESSES THE CORRECT LSCB.                          │
  │                                                                      │
  │      OUTPUT:    TG_OP' TO PC_ROUTE_MGR.RCV; UPDATED TGCB             │
  │                                                                      │
  │      NOTE:      THIS ADDS THE NEWLY ACTIVE LINK STATION TO THE TGCB.  │
  │                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
  │                 NS.DLC_CONFIG                    PAGE 11-66          │
  │                                                                      │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
  │                 FSM_XID_FORMAT_2                 PAGE 11-126         │
  │                 SEND_CONTACTED                   PAGE 11-73          │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  TGCB_PTR = LSCB.TGCBPTR;

  CREATE ASSOC_LSCB_ENTITY;

  ASSOC_LSCB_ENTITY.LSCBPTR = LSCB_PTR;
  INSERT ASSOC_LSCB_ENTITY IN TGCB.ASSOC_LSCB_LIST;

  IF EMPTY(TGCB.ASSOC_LSCB_LIST) THEN
    DO;
    . TGCB.MAX_SEND_BTU_LENGTH = LSCB.XID_RCV.MAX_PIU_LENGTH;
    . SEND 'TG_OP' TO PU.SVC_MGR.PC_ROUTE_MGR.RCV;           /* CHAPTER 12    */
    END;

  NRCB_PTR = LOCATE_NODE_RESOURCE(LSCB.EA);
  IF FSM_XID_FORMAT_2 = PEND_ACT_PRI_2 THEN
    NRCB.LINK_DLC_ROLE = SECONDARY;
  ELSE
    NRCB.LINK_DLC_ROLE = PRIMARY;

  INSERT ASSOC_LSCB_ENTITY IN TGCB.ASSOC_LSCB_LIST;          /* NOTE          */
  MU_PTR = UPM_CREATE_RQ('CONTACTED(LOADED_STA)');           /* APPENDIX B    */
  CALL SEND_CONTACTED;                                       /* PAGE 11-73    */
  CALL FSM_XID_FORMAT_2;                 /* PAGE 11-126, INPUT IS CONTACTED(LOADED_STA) */

  DISCARD MU;                                                /* DISCARD CONTACTED RQ */

  RETURN;

END STATION_CONTACTED;
```

```
SEND_CONTACTED: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE IS  CALLED FROM  STATION_CONTACTED, XID_ERR_RCV  AND │
   │                 XID_ERR_SEND  TO  SEND  CONTACTED  REQUESTS  TO  THE  CP(S)  IN  THE │
   │                 CPCB_LIST FOR  THIS LINK  STATION, I.E.,  THAT HAVE  SENT A  CONTACT │
   │                 REQUEST THAT HAS RESULTED  IN A POSITIVE RESPONSE, BUT NOT  YET IN A │
   │                 CONTACTED REQUEST.                                       │
   │                                                                          │
   │      INPUT:     THE CURRENT MESSAGE  UNIT IS A COPY  OF THE CONTACTED REQUEST  TO BE │
   │                 SENT.  THE LSCB_PTR ADDRESSES THE CORRECT LSCB.          │
   │                                                                          │
   │      OUTPUT:    CONTACTED REQUEST(S) TO SNS.SEND.                        │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 STATION_CONTACTED                PAGE 11-72              │
   │                 XID_ERR_RCV                      PAGE 11-73              │
   │                 XID_ERR_SEND                     PAGE 11-75              │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL CONTACTED_PTR PTR;

  CONTACTED_PTR = MU_PTR;
  FIND NRCB IN NRCB_LIST WHERE(NRCB.ELEMENT_ADDRESS = LSCB.EA);

  SCAN NRCB.CP_INDIRECT_LIST PTR(CP_INDIRECT_PTR);
  . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
  . CREATE MU;
  . MU = CONTACTED_PTR->MU;                  /* COPY THE CONTACTED REQUEST      */
  .
  . SCB_PTR = CPCB.CP_SCB_ID;
  . SEND MU TO SNS.SEND;                                    /* CHAPTER 6      */
  SCANEND;

  MU_PTR = CONTACTED_PTR;

  RETURN;

END SEND_CONTACTED;



XID_ERR_RCV: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE IS CALLED FROM XID_FORMAT_2_RCV  WHEN AN XID WITH THE │
   │                 ERROR_STATUS FIELD SET TO A NONZERO VALUE IS RECEIVED.  THE NON-ZERO │
   │                 VALUE INDICATES  THAT THE LATEST XID  SENT FROM THE LINK  STATION IN │
   │                 THIS NODE WAS IN ERROR.                                  │
   │                                                                          │
   │      INPUT:     RECEIVED XID  IS THE CURRENT  MESSAGE UNIT.  THE  LSCB_PTR ADDRESSES │
   │                 THE CORRECT LSCB.                                        │
   │                                                                          │
   │      OUTPUT:    CONTACTED WITH APPROPRIATE ERROR STATUS TO SNS.SEND       │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 FSM_TGN                          PAGE 11-125             │
   │                 XID_FORMAT_2_RCV                 PAGE 11-67              │
   │                                                                          │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
   │                 FSM_TGN                          PAGE 11-125             │
   │                 FSM_XID_FORMAT_2                 PAGE 11-126             │
   │                 SEND_CONTACTED                   PAGE 11-73              │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL XID BASED(ADDR(RU)) LIKE ISCB.XID_SEND;

  LSCB.XID_RCV = XID;
  CALL FSM_XID_FORMAT_2;                  /* PAGE 11-126, INPUT IS (ERR,R)     */
  DISCARD MU;                                                   /* DISCARD XID */

  SELECT ANYORDER(LSCB.XID_RCV.ERROR_STATUS);
  . WHEN(EXCHANGED_PARMS_INCOMPAT | MULTI_OR_DLC_INCOMPATIBLE)
  .   MU_PTR = UPM_CREATE_RQ('CONTACTED(INCOMPAT_ST)');                 /* APPENDIX B */
  .
  . WHEN(SUBSEQUENT_LINK_PARMS_INCOMPAT)
  .   MU_PTR = UPM_CREATE_RQ('CONTACTED(INCOMPAT_TG)');                 /* APPENDIX B */
  .
  . WHEN(NO_TG)
  .   MU_PTR = UPM_CREATE_RQ('CONTACTED(NO_ROUTE)');                    /* APPENDIX B */
  END;

  CALL SEND_CONTACTED;                                                  /* PAGE 11-73 */
  CALL FSM_TGN;                          /* PAGE 11-125, INPUT IS CONTACTED(NOT_LOADED) */

  DISCARD MU;                                                   /* DISCARD CONTACTED RQ */

  RETURN;

END XID_ERR_RCV;
```

This page intentionally left blank

```
XID_ERR_SEND: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS  PROCEDURE IS  CALLED FROM  XID_FORMAT_2_RCV WHEN  AN ERROR  IS │
  │                 FOUND WHILE  PROCESSING A RECEIVED  XID.  THIS PROCEDURE  BUILDS AND │
  │                 SENDS AN XID AND CONTACTED REQUEST(S) WITH APPROPRIATE ERROR CODES.  │
  │                                                                              │
  │      INPUT:     THE LSCB_PTR ADDRESSES THE CORRECT LSCB.                     │
  │                                                                              │
  │      OUTPUT:    XID TO LINK_MGR                                              │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 FSM_TGN                          PAGE 11-125                 │
  │                 MULTI_LINK_TESTS                 PAGE 11-70                  │
  │                 XID_FORMAT_CHECK_1               PAGE 11-68                  │
  │                 XID_FORMAT_CHECK_2               PAGE 11-69                  │
  │                 XID_FORMAT_2_RCV                 PAGE 11-67                  │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 FSM_TGN                          PAGE 11-125                 │
  │                 FSM_XID_FORMAT_2                 PAGE 11-126                 │
  │                 SEND_CONTACTED                   PAGE 11-73                  │
  │                 UPM_BUILD_ERROR_XID              PAGE 11-113                 │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DISCARD MU;                                                 /* DISCARD XID   */

  CALL UPM_BUILD_ERROR_XID;                                   /* PAGE 11-113   */

  SEND MU TO PU.SVC_MGR.LINK_MGR;

  IF FSM_XID_FORMAT_2 ¬= RESET THEN                           /* PAGE 11-126   */
    DO;
    . SELECT ANYORDER(LSCB.CONTACTED_STATUS);
    . . WHEN(INCOMPATIBLE_STATIONS)
    . .   MU_PTR = UPM_CREATE_RQ('CONTACTED(INCOMPAT_ST)');   /* APPENDIX B    */
    . . WHEN(INCOMPATIBLE_WITH_TG)
    . .   MU_PTR = UPM_CREATE_RQ('CONTACTED(INCOMPAT_TG)');   /* APPENDIX B    */
    . . WHEN(NO_TG)
    . .   MU_PTR = UPM_CREATE_RQ('CONTACTED(NO_TG)');         /* APPENDIX B    */
    . END;
    . CALL SEND_CONTACTED;                                    /* PAGE 11-73    */
    . CALL FSM_XID_FORMAT_2;          /* INPUT IS CONTACTED(NOT_LOADED), PAGE 11-126 */
    . CALL FSM_TGN;                   /* INPUT IS CONTACTED(NOT_LOADED), PAGE 11-125 */
    . DISCARD MU;                                             /* DISCARD CONTACTED RQ */
    END;

  RETURN;

END XID_ERR_SEND;
```

```
NS.DLC_RCV: PROCEDURE;
                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE RECEIVES ALL MESSAGES FROM THE LINK_MGR.  VALID INPUT
│                 IS SENT TO  THE APPROPRIATE PROCEDURE.  INVALID INPUT  CAUSES A CALL
│                 TO NS.INOP_PROC THAT  CREATES AN INOP REQUEST, WHICH IS  SENT TO ALL
│                 APPROPRIATE HALF-SESSIONS.
│
│      INPUT:     REQUESTS, RESPONSES, OR SIGNALS FROM THE LINK_MGR
│
│      OUTPUT:    VALID  NS REQUESTS,  VALID  NS RESPONSES,  VALID  SIGNALS, AND  INOP
│                 REQUESTS
│
│      NOTE:      THE PROTOCOL BOUNDARY  THAT IS MAINTAINED BETWEEN  THE PU.SVC_MGR.NS
│                 AND LINK_MGR IS DEFINED AS FOLLOWS:
│                 1.  TH INFORMATION:
│                     • NONE
│                 2.  RH INFORMATION:
│                     • REQUEST/RESPONSE INDICATOR
│                     • SENSE DATA INCLUDED INDICATOR
│                 3.  RU INFORMATION  (THE RU  INFORMATION IS IN  THE FORMAT  SHOWN IN
│                     APPENDIX E).
│
│                     IN  ADDITION  TO RU'S  AND  RESPONSES  THAT FLOW  THE  FOLLOWING
│                     SIGNALS ALSO ARE SENT BY DLC.PRI:
│                     • 'ALS_RESET_COMPLETE'
│                     • 'CONNECT_IN_SUCCESSFUL'
│                     • 'CONNECT_OUT_SUCCESSFUL'
│                     • 'XID_COMPLETED' INCLUDES RETURNED XID DATA.
│                     • 'LINK_RESET_COMPLETE'
│                     • 'LINK_TEST_COMPLETED'
│                     • 'TEST_COMPLETED'
│
│                     THE FOLLOWING SIGNALS  ARE SENT BY PU.SVC_MGR.NS  TO DLC.PRI:  •
│                     'RESET'
│                     • 'XID' INCLUDES XID DATA TO BE SENT
│
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                 PU.SVC_MGR.NS.RCV                   PAGE 11-28
│
│      REFERS TO THE FOLLOWING PROCEDURE(S):
│                 NS.CONN_RSP                         PAGE 11-82
│                 NS.CONTACT_RSP                      PAGE 11-80
│                 NS.EXECTEST_PROC                    PAGE 11-108
│                 NS.INOP_PROC                        PAGE 11-90
│                 NS.LINK_RSP                         PAGE 11-79
│                 NS.LOAD_RSP                         PAGE 11-84
│                 NS.SIG_RSP_PRI                      PAGE 11-86
│                 NS.SIG_RSP_SEC                      PAGE 11-88
│                 NS.TESTMODE_PROC                    PAGE 11-109
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  SELECT ANYORDER;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │              INPUT IS REQUEST            │
                  └──────────────────────────────────────────┘
                                                                        */
  . WHEN(INPUT(RQ) & NS_RQ_CODE = CONTACTED &
  .      CONTACTED_RQ.STATUS = LOADED &
  .      NRCB.LINK_DLC_ROLE = SECONDARY)
  .   CALL NS.CONTACT_RSP;                          /* PAGE 11-80          */
  .
  . WHEN(INPUT(RQ) & NS_RQ_CODE = CONTACTED &
  .      NRCB.LINK_DLC_ROLE = PRIMARY)
  .   CALL NS.CONTACT_RSP;                          /* PAGE 11-80          */
  .
  . WHEN(INPUT(RQ) & NS_RQ_CODE = INOP)
  .   CALL NS.INOP_PROC(LSCB.EA);                   /* PAGE 11-90          */
  .
  . WHEN(INPUT(RQ) & NS_RQ_CODE = (IPLINIT | IPLTEXT | IPLFINAL |
  .      DUMPINIT | DUMPTEXT | DUMPFINAL | RPO) &
  .      NRCB.LINK_DLC_ROLE = SECONDARY)
  .   SEND MU TO UPM_DLC_SEC_RQ_PROC;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │  THIS  UPM  PERFORMS  THE IPL,  DUMP,  OR  RPO  │
                  │  FUNCTION, CREATES THE RESPONSE,  AND SENDS IT  │
                  │  BACK TO THE LINK_MGR.                          │
                  └──────────────────────────────────────────┘
                                                                        */
  .
  . WHEN(INPUT(RQ) & NS_RQ_CODE = RECTR)
  .   CALL NS.TESTMODE_PROC;                        /* PAGE 11-109         */
  .
  . WHEN(INPUT(RQ) & NS_RQ_CODE = RECTD)
  .   CALL NS.EXECTEST_PROC;                        /* PAGE 11-108         */
```

```
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │          INPUT IS A RESPONSE         │
                    └──────────────────────────────────────┘
                                                                        */
  . WHEN(INPUT(RSP) & NS_RQ_CODE = (ACTCONNIN | DACTCONNIN | ABCONN))
  .   DO;
  .   . IF RTI = POSITIVE THEN
  .   .   CALL NS.CONN_RSP;                          /* PAGE 11-82     */
  .   . ELSE
  .   .   CALL NS.INOP_PROC(LSCB.EA);                /* PAGE 11-90     */
  .   END;
  .
  . WHEN(INPUT(RSP) & NS_RQ_CODE = (ACTLINK | DACTLINK))
  .   CALL NS.LINK_RSP;                              /* PAGE 11-79     */
  .
  . WHEN(INPUT(RSP) & NS_RQ_CODE = (CONNOUT | ABCONNOUT))
  .   CALL NS.CONN_RSP;                              /* PAGE 11-82     */
  .
  . WHEN(INPUT(RSP) & NS_RQ_CODE = DISCONTACT)
  .   CALL NS.CONTACT_RSP;                           /* PAGE 11-80     */
  .
  . WHEN(INPUT(RSP) &
  .      (NS_RQ_CODE = (IPLINIT | IPLTEXT | IPLFINAL |
  .      DUMPINIT | DUMPTEXT | DUMPFINAL | RPO)))
  .   CALL NS.LOAD_RSP;                              /* PAGE 11-84     */
  .
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │          INPUT IS A SIGNAL           │
                    └──────────────────────────────────────┘
                                                                        */
  . WHEN(NRCB.LINK_DLC_ROLE = PRIMARY &
  .      (INPUT('CONNECT_IN_SUCCESSFUL') |
  .      INPUT('CONNECT_OUT_SUCCESSFUL') |
  .      INPUT('XID_COMPLETED') |
  .      INPUT('ALS_RESET_COMPLETE') |
  .      INPUT('LINK_RESET_COMPLETE')))
  .   CALL NS.SIG_RSP_PRI;                           /* PAGE 11-86     */
  .
  . WHEN(NRCB.LINK_DLC_ROLE = SECONDARY &
  .      (INPUT('CONNECT_IN_SUCCESSFUL') |
  .      INPUT('CONNECT_OUT_SUCCESSFUL') |
  .      INPUT('XID') |
  .      INPUT('ALS_RESET_COMPLETE') |
  .      INPUT('LINK_RESET_COMPLETE')))
  .   CALL NS.SIG_RSP_SEC;                           /* PAGE 11-88     */
  .
  . WHEN(INPUT('TEST_COMPLETED'))
  .   CALL NS.TESTMODE_PROC;                         /* PAGE 11-109    */
  .
  . WHEN(INPUT('LINK_TEST_COMPLETED'))
  .   CALL NS.EXECTEST_PROC;                         /* PAGE 11-108    */
  .
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │ ANY SIGNAL, OR RU BUT NOT ONE OF THOSE ABOVE │
                    └──────────────────────────────────────┘
                                                                        */
  . OTHERWISE
  .   CALL NS.INOP_PROC(LSCB.EA);                    /* PAGE 11-90     */
  END;

  RETURN;
END NS.DLC_RCV;
```

This page intentionally left blank

```
NS.LINK_RSP: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THE RESOURCE  FSM FOR THE LINK  ADDRESSED BY ACTLINK OR  DACTLINK IS
  │                 CHECKED.  IF  A RSP(ACTLINK) THEN  THE RESOURCE  FSM MUST BE  IN THE
  │                 PEND_ACTIVE STATE.  IF A RSP(DACTLINK) IS RECEIVED THEN THE RESOURCE
  │                 FSM MUST BE IN  THE PEND_RESET STATE.  IF NOT TRUE  THEN THE LINK IS
  │                 RESET.
  │
  │      INPUT:     POSITIVE  AND  NEGATIVE  RESPONSES  TO  ACTLINK  AND  DACTLINK  FROM
  │                 NS.DLC_RCV  (PAGE  11-76).   THE   LSCB_PTR  IDENTIFIES  THE  SOURCE
  │                 ADJACENT LINK STATION.
  │
  │      OUTPUT:    THE  RESPONSES   TO  ACTLINK  OR   DACTLINK  TO  THE   RESOURCE  FSM
  │                 LINK_EA.LINK_ACT_RES AND TO THE CORRESPONDING HALF-SESSIONS
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 NS.DLC_RCV                          PAGE 11-76
  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):
  │                 FSM_LINK_ACT_RES                    PAGE 11-119
  │                 NS.INOP_PROC                        PAGE 11-90
  │                 UPM_RESTORE_SNF                     PAGE 11-115
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL LINK_EA BIT(16);

  NRCB_PTR = FIND_LINK_FOR_RESOURCE(LSCB.EA);                  /* APPENDIX B    */
  LINK_EA = NRCB.ELEMENT_ADDRESS;
  SELECT ANYORDER(NS_RQ_CODE);
  .
  . WHEN(ACTLINK)
  .   DO;
  .   . IF FSM_LINK_ACT_RES ¬= PEND_ACTIVE THEN               /* PAGE 11-119   */
  .   .   CALL NS.INOP_PROC(LINK_EA);                        /* PAGE 11-90    */
  .   . ELSE
  .   .   DO;
  .   .   . CALL FSM_LINK_ACT_RES;                           /* PAGE 11-119   */
  .   .   . SCAN NRCB.CP_INDIRECT_LIST PTR(CP_INDIRECT_PTR);
  .   .   . . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
  .   .   . . SCB_PTR = CPCB.CP_SCB_ID;
  .   .   . . MU_PTR = UPM_CREATE_RSP('ACTLINK');             /* APPENDIX B    */
  .   .   . . CALL UPM_RESTORE_SNF;                          /* PAGE 11-115   */
  .   .   . . SEND MU TO SNS.SEND;                           /* CHAPTER 6     */
  .   .   . SCANEND;
  .   .   END;
  .   END;
  .
  . WHEN(DACTLINK)
  .   DO;
  .   . IF FSM_LINK_ACT_RES ¬= PEND_RESET THEN               /* PAGE 11-119   */
  .   .   CALL NS.INOP_PROC(LINK_EA);                        /* PAGE 11-90    */
  .   . ELSE
  .   .   DO;
  .   .   . CALL FSM_LINK_ACT_RES;                           /* PAGE 11-119   */
  .   .   . SCAN NRCB.CP_INDIRECT_LIST PTR(CP_INDIRECT_PTR);
  .   .   . . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
  .   .   . . SCB_PTR = CPCB.CP_SCB_ID;
  .   .   . . MU_PTR = UPM_CREATE_RSP('DACTLINK');            /* APPENDIX B    */
  .   .   . . CALL UPM_RESTORE_SNF;                          /* PAGE 11-115   */
  .   .   . . SEND MU TO SNS.SEND;                           /* CHAPTER 6     */
  .   .   . SCANEND;
  .   .   . CALL DEQUEUE_RUS_FROM_RESOURCE(LINK_EA);         /* APPENDIX B    */
  .   .   END;
  .   END;
  END;
  RETURN;
END NS.LINK_RSP;
```

```
NS.CONTACT_RSP: PROCEDURE;

                                                                                /*
┌──────────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THE  PERTINENT RESOURCE  FSM  IS CHECKED  TO  SEE IF  IT  IS IN  THE │
│                 APPROPRIATE  STATE TO  RECEIVE  THE  RESPONSE/REQUEST.  IF  SO,  THE │
│                 RESOURCE FSM'S  ARE UPDATED AND  THE RESPONSE/REQUEST  IS FORWARDED. │
│                 IF NOT, THE RESPONSE/REQUEST IS DISCARDED AND NS.INOP_PROC IS CALLED │
│                 TO CREATE  AN INOP REQUEST AND  SEND THE REQUEST TO  THE APPROPRIATE │
│                 HALF-SESSIONS.                                               │
│                                                                              │
│      INPUT:     CONTACTED(LOADED),    CONTACTED(ERROR),     CONTACTED(LOAD_REQUIRED) │
│                 REQUESTS, ±RSP(DISCONTACT,0822) FROM NS.DLC_RCV (PAGE 11-76) │
│                                                                              │
│      OUTPUT:    THE REQUEST/RESPONSE  TO THE FIRST  CP IN THE  RESOURCE'S CPCB_LIST. │
│                 RESET SIGNAL TO THE CONTACT RESOURCE FSM                     │
│                                                                              │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
│                 NS.DLC_RCV                      PAGE 11-76                   │
│                                                                              │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
│                 FSM_ALS_CONTACT_DISCONTACT_RES  PAGE 11-122                  │
│                 NS.INOP_PROC                    PAGE 11-90                   │
└──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
DCL ALS_EA BIT(16);

NRCB_PTR = LOCATE_NODE_RESOURCE(LSCB.EA);                       /* APPENDIX B   */
ALS_EA = NRCB.ELEMENT_ADDRESS;
SELECT ANYORDER(NS_RQ_CODE);
                                                                                /*
                ┌──────────────────────────────────────────┐
                │        CONTACTED(LOADED) FROM NS.DLC_RCV │
                └──────────────────────────────────────────┘
                                                                                */
. WHEN(CONTACTED)
.   DO;
.   . IF CONTACTED_RQ.STATUS = LOADED THEN
.   .   DO;
.   .   . IF FSM_ALS_CONTACT_DISCONTACT_RES = PEND_ACTIVE THEN /* PAGE 11-122   */
.   .   .   DO;
.   .   .   . CALL FSM_ALS_CONTACT_DISCONTACT_RES; /* CONTACTED(LOADED) PAGE 11-122 */
.   .   .   . SCAN NRCB.CP_INDIRECT_LIST PTR(CP_INDIRECT_PTR);
.   .   .   . . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
.   .   .   . . SCB_PTR = CPCB.CP_SCB_ID;
.   .   .   . . MU_PTR = UPM_CREATE_RQ('CONTACTED(LOADED)');
.   .   .   . . SEND MU TO SNS.SEND;                            /* CHAPTER 6    */
.   .   .   . SCANEND;
.   .   .   . IF LSCB.TGCBPTR ¬= NULL THEN
.   .   .   .   DO;
.   .   .   .   . TGCB_PTR = LSCB.TGCBPTR;
.   .   .   .   . CREATE ASSOC_LSCB_ENTITY;
.   .   .   .   . ASSOC_LSCB_ENTITY.LSCBPTR = LSCB_PTR;
.   .   .   .   . INSERT ASSOC_LSCB_ENTITY IN TGCB.ASSOC_LSCB_LIST;
.   .   .   .   . SEND 'TG_OP' TO PU.SVC_MGR.PC_ROUTE_MGR.RCV; /* CHAPTER 12   */
.   .   .   .   END;
.   .   .   END;
.   .   . IF FSM_ALS_CONTACT_DISCONTACT_RES = PEND_RESET |
.   .   .      FSM_ALS_CONTACT_DISCONTACT_RES = RESET_IN_PROGRESS THEN
.   .   .                                                       /* PAGE 11-122  */
.   .   .   DISCARD MU;
.   .   . ELSE
.   .   .                               /* RESET, ACTIVE, OR TEST IN PROGRESS   */
.   .   .   CALL NS.INOP_PROC(ALS_EA);                          /* PAGE 11-90   */
.   .   END;
                                                                                /*
                ┌──────────────────────────────────────────┐
                │  CONTACTED(ERROR) OR  CONTACTED(LOAD REQUIRED)  │
                │  FROM NS.DLC_RCV                         │
                └──────────────────────────────────────────┘
                                                                                */
.   . ELSE
.   .   IF CONTACTED_RQ.STATUS = (LOAD_REQUIRED | ERROR) THEN
.   .     DO;
.   .     . IF FSM_ALS_CONTACT_DISCONTACT_RES = PEND_ACTIVE THEN /* PAGE 11-122  */
.   .     .   DO;
.   .     .   . CALL FSM_ALS_CONTACT_DISCONTACT_RES; /* CONTACTED(ERROR|LOAD_REQUIRED) */
.   .     .   .                                                 /* PAGE 11-122  */
.   .     .   . SCAN NRCB.CP_INDIRECT_LIST PTR(CP_INDIRECT_PTR);
.   .     .   . . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
.   .     .   . . SCB_PTR = CPCB.CP_SCB_ID;
.   .     .   . . IF CONTACTED_RQ.STATUS = ERROR THEN
.   .     .   . .   MU_PTR = UPM_CREATE_RQ('CONTACTED(ERROR)');
.   .     .   . . ELSE
.   .     .   . .   MU_PTR = UPM_CREATE_RQ('CONTACTED(LOAD_REQUIRED)');
.   .     .   . . SEND MU TO SNS.SEND;                          /* CHAPTER 6    */
.   .     .   . SCANEND;
.   .     .   END;
.   .     END;
.   . IF FSM_ALS_CONTACT_DISCONTACT_RES = PEND_RESET |
.   .      FSM_ALS_CONTACT_DISCONTACT_RES = RESET_IN_PROGRESS THEN
.   .                                                         /* PAGE 11-122  */
.   .   DISCARD MU;
.   . ELSE
.   .                               /* RESET, ACTIVE, OR TEST IN PROGRESS   */
.   .   CALL NS.INOP_PROC(ALS_EA);                            /* PAGE 11-90   */
.   END;
```

```
                                                                                /*
                    ┌──────────────────────────────────────────────┐
                    │ POSITIVE OR NEGATIVE RESPONSE TO DISCONTACT  │
                    │ FROM NS.DLC_RCV                              │
                    └──────────────────────────────────────────────┘
                                                                                */
  . WHEN(DISCONTACT)
  .   DO;
  .   . CP_INDIRECT_PTR = FIRST_ENTRY(NRCB.CP_INDIRECT_LIST);
  .   . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
  .   . SCB_PTR = CPCB.CP_SCB_ID;
  .   . IF FSM_ALS_CONTACT_DISCONTACT_RES = PEND_RESET THEN      /* PAGE 11-122      */
  .   .   DO;
  .   .   . SEND MU TO SNS.SEND;                                /* CHAPTER 6        */
  .   .   . CALL FSM_ALS_CONTACT_DISCONTACT_RES('RESET');       /* PAGE 11-122      */
  .   .   . TGCB_PTR = FIND_TGCB_FOR_ALS_EA(ALS_EA);           /* APPENDIX B       */
  .   .   . IF TGCB_PTR ¬= NULL THEN
  .   .   .   DO;
  .   .   .   . CALL DELETE_ALS_FROM_TGCB(ALS_EA);             /* APPENDIX B       */
  .   .   .   . IF EMPTY(TGCB.ASSOC_LSCB_LIST) THEN
  .   .   .   .   SEND 'TG_INOP_NORMAL' TO PU.SVC_MGR.PC_ROUTE_MGR.RCV; /* CHAPTER 12 */
  .   .   .   END;
  .   .   . ELSE
  .   .   .   SEND 'REX_INOP' TO PU.SVC_MGR.CSC_MGR.SON;       /* CHAPTER 13       */
  .   .   . CALL DEQUEUE_RUS_FROM_RESOURCE(ALS_EA);            /* APPENDIX B       */
  .   .   END;
  .   . IF FSM_ALS_CONTACT_DISCONTACT_RES = RESET_IN_PROGRESS THEN
  .   .                                                        /* PAGE 11-122      */
  .   .   DISCARD MU;
  .   . ELSE
  .   .           /* RESET, ACTIVE, PEND_ACTIVE, OR TEST IN PROGRESS              */
  .   .   CALL NS.INOP_PROC(ALS_EA);                           /* PAGE 11-90       */
  .   END;
  END;
  RETURN;
END NS.CONTACT_RSP;
```

```
NS.CONN_RSP: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THE  PERTINENT RESOURCE  FSM  IS CHECKED  TO  SEE IF  IT  IS IN  THE │
│                 APPROPRIATE STATE TO  RECEIVE THE RESPONSE.  IF SO,  THE RESPONSE IS │
│                 SENT TO THE CONTROL POINT IN THE RESOURCE'S CP_LIST AND THE RESOURCE │
│                 FSM IS CALLED.  OTHERWISE, THE RESPONSE  IS DISCARDED AND AN INOP IS │
│                 GENERATED FOR THE LINK.                                  │
│                                                                          │
│      INPUT:     POSITIVE  AND  NEGATIVE  RESPONSES TO  CONNOUT  AND  ABCONNOUT,  AND │
│                 POSITIVE  RESPONSES  TO  ABCONN,   ACTCONNIN,  AND  DACTCONNIN  FROM │
│                 NS.DLC_RCV (PAGE 11-76)                                  │
│                                                                          │
│      OUTPUT:    RESPONSES THAT  WERE RECEIVED  AS INPUT  TO THE  APPROPRIATE CONTROL │
│                 POINT                                                    │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 NS.DLC_RCV                      PAGE 11-76               │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 FSM_ALS_CONNECTED_RES           PAGE 11-121              │
│                 FSM_LINK_CONNIN_RES             PAGE 11-120              │
│                 FSM_LINK_CONNOUT_RES            PAGE 11-121              │
│                 NS.INOP_PROC                    PAGE 11-90               │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL ERROR_SWITCH BIT(1);
  DCL LINK_EA BIT(16);

  NRCB_PTR = FIND_LINK_FOR_RESOURCE(LSCB.EA);                  /* APPENDIX B      */
  LINK_EA = NRCB.ELEMENT_ADDRESS;
  ERROR_SWITCH = ON;
  CP_INDIRECT_PTR = FIRST_ENTRY(NRCB.CP_INDIRECT_LIST);
  CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
  SCB_PTR = CPCB.CP_SCB_ID;
  IF SCB_PTR ¬= NULL THEN
    SELECT ANYORDER(NS_RQ_CODE);
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │          POSITIVE RESPONSE TO ACTCONNIN          │
                  └──────────────────────────────────────────┘
                                                                        */
    . WHEN(ACTCONNIN)
    .   DO;
    .   . IF FSM_LINK_CONNIN_RES = PEND_ACTIVE THEN            /* PAGE 11-120     */
    .   .   DO;
    .   .   . SEND MU TO SNS.SEND;                             /* CHAPTER 6       */
    .   .   . CALL FSM_LINK_CONNIN_RES;                        /* PAGE 11-120     */
    .   .   . ERROR_SWITCH = OFF;
    .   .   END;
    .   END;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │          POSITIVE RESPONSE TO DACTCONNIN         │
                  └──────────────────────────────────────────┘
                                                                        */
    . WHEN(DACTCONNIN)
    .   DO;
    .   . IF FSM_LINK_CONNIN_RES = PEND_RESET THEN             /* PAGE 11-120     */
    .   .   DO;
    .   .   . SEND MU TO SNS.SEND;                             /* CHAPTER 6       */
    .   .   . CALL FSM_LINK_CONNIN_RES;                        /* PAGE 11-120     */
    .   .   . ERROR_SWITCH = OFF;
    .   .   END;
    .   END;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │     POSITIVE OR NEGATIVE RESPONSE TO CONNOUT     │
                  └──────────────────────────────────────────┘
                                                                        */
    . WHEN(CONNOUT)
    .   DO;
    .   . IF FSM_LINK_CONNOUT_RES = PEND_ACTIVE THEN           /* PAGE 11-121     */
    .   .   DO;
    .   .   . SEND MU TO SNS.SEND;                             /* CHAPTER 6       */
    .   .   . CALL FSM_LINK_CONNOUT_RES;                       /* PAGE 11-121     */
    .   .   . ERROR_SWITCH = OFF;
    .   .   END;
    .   END;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │    POSITIVE OR NEGATIVE RESPONSE TO ABCONNOUT    │
                  └──────────────────────────────────────────┘
                                                                        */
    . WHEN(ABCONNOUT)
    .   DO;
    .   . IF FSM_LINK_CONNOUT_RES = PEND_RESET THEN            /* PAGE 11-121     */
    .   .   DO;
    .   .   . SEND MU TO SNS.SEND;                             /* CHAPTER 6       */
    .   .   . CALL FSM_LINK_CONNOUT_RES;                       /* PAGE 11-121     */
    .   .   . ERROR_SWITCH = OFF;
    .   .   END;
    .   END;
```

```
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │      POSITIVE RESPONSE TO ABCONN     │
                    └──────────────────────────────────────┘
                                                                        */
   . WHEN(ABCONN)
   .   DO;
   .   . NRCB_PTR = FIND_ALS_FOR_RESOURCE(LSCB.EA);          /* APPENDIX B     */
   .   . IF FSM_ALS_CONNECTED_RES = PEND_RESET THEN          /* PAGE 11-121    */
   .   .   DO;
   .   .   . SEND MU TO SNS.SEND;                           /* CHAPTER 6      */
   .   .   . CALL FSM_ALS_CONNECTED_RES;                    /* PAGE 11-121    */
   .   .   . ERROR_SWITCH = OFF;
   .   .   END;
   .   END;
   END;
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │ COMMON PROCESSING TO DISCARD THE RESPONSE │
                    │ (RESPONSE IS NOT CURRENTLY APPROPRIATE).  │
                    └──────────────────────────────────────┘
                                                                        */
 IF ERROR_SWITCH = ON THEN
   DO;
   . DISCARD MU;
   . CALL NS.INOP_PROC(LINK_EA);                             /* PAGE 11-90     */
   END;
 RETURN;
END NS.CONN_RSP;
```

```
NS.LOAD_RSP: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  WHEN A  RESPONSE IS  RECEIVED, THE  APPROPRIATE RESOURCE  FSM (DUMP,  │
│                 IPL,  OR RPO)  IS  CHECKED.  THE  RESOURCE FSM  IS  UPDATED AND  THE  │
│                 RESPONSE IS  SENT TO THE CONTROL  POINT.  IF THE  APPROPRIATE (DUMP,  │
│                 IPL,  OR RPO)  RESOURCE FSM  IS NOT  FOUND  IN A  STATE PENDING  THE  │
│                 RECEIPT OF THIS RESPONSE,  AN INOP IS SENT TO ANY  CONTROL POINTS IN  │
│                 THE ALS RESOURCE'S CP_LIST.                                           │
│                                                                                       │
│      INPUT:     POSITIVE  AND  NEGATIVE  RESPONSES TO  IPLINIT,  IPLTEXT,  IPLFINAL,  │
│                 DUMPINIT, DUMPTEXT, DUMPFINAL, AND RPO FROM NS.DLC_RCV (PAGE 11-76)   │
│                                                                                       │
│      OUTPUT:    IPLINIT, IPLTEXT, AND IPLFINAL  RESPONSES TO FSM_ALS_SEC_IPL_RES AND  │
│                 TO  SNS.SEND;   DUMPINIT,  DUMPTEXT,  AND  DUMPFINAL   RESPONSES  TO  │
│                 FSM_ALS_SEC_DUMP_FSM   AND   TO   SNS.SEND;     RPO   RESPONSES   TO  │
│                 FSM_ALS_SEC_RPO_FSM   AND   TO   THE   SNS.SEND;   INOP(ALS_EA)   TO  │
│                 NS.INOP_PROC                                                          │
│                                                                                       │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                        │
│                 NS.DLC_RCV                          PAGE 11-76                        │
│                                                                                       │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                                            │
│                 FSM_ALS_SEC_DUMP_RES                PAGE 11-122                       │
│                 FSM_ALS_SEC_IPL_RES                 PAGE 11-123                       │
│                 FSM_ALS_SEC_RPO_RES                 PAGE 11-123                       │
│                 NS.INOP_PROC                        PAGE 11-90                        │
│                 UPM_RESTORE_SNF                     PAGE 11-115                       │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL ALS_EA BIT(16);

NRCB_PTR = FIND_ALS_FOR_RESOURCE(LSCB.EA);                  /* APPENDIX B      */
ALS_EA = NRCB.ELEMENT_ADDRESS;
CP_INDIRECT_PTR = FIRST_ENTRY(NRCB.CP_INDIRECT_LIST);
CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
SCB_PTR = CPCB.CP_SCB_ID;
SELECT ANYORDER(NS_RQ_CODE);
                                                                        /*
                ┌──────────────────────────────────────────┐
                │    POSITIVE OR NEGATIVE RESPONSE TO IPLINIT    │
                └──────────────────────────────────────────┘
                                                                        */
. WHEN(IPLINIT)
.   DO;
.   . IF FSM_ALS_SEC_IPL_RES = PEND_INIPL_INIT THEN          /* PAGE 11-123     */
.   .   DO;
.   .   . CALL UPM_RESTORE_SNF;                             /* PAGE 11-115     */
.   .   . SEND MU TO SNS.SEND;                              /* CHAPTER 6       */
.   .   . CALL FSM_ALS_SEC_IPL_RES;                         /* PAGE 11-123     */
.   .   END;
.   . ELSE
.   .   DO;
.   .   . DISCARD MU;
.   .   . CALL NS.INOP_PROC(ALS_EA);                        /* PAGE 11-90      */
.   .   END;
.   END;
                                                                        /*
                ┌──────────────────────────────────────────┐
                │    POSITIVE OR NEGATIVE RESPONSE TO IPLTEXT    │
                └──────────────────────────────────────────┘
                                                                        */
. WHEN(IPLTEXT)
.   DO;
.   . IF FSM_ALS_SEC_IPL_RES = PEND_INIPL_TEXT THEN          /* PAGE 11-123     */
.   .   DO;
.   .   . CALL UPM_RESTORE_SNF;                             /* PAGE 11-115     */
.   .   . SEND MU TO SNS.SEND;                              /* CHAPTER 6       */
.   .   . CALL FSM_ALS_SEC_IPL_RES;                         /* PAGE 11-123     */
.   .   END;
.   . ELSE
.   .   DO;
.   .   . DISCARD MU;
.   .   . CALL NS.INOP_PROC(ALS_EA);                        /* PAGE 11-90      */
.   .   END;
.   END;
                                                                        /*
                ┌──────────────────────────────────────────┐
                │    POSITIVE OR NEGATIVE RESPONSE TO IPLFINAL   │
                └──────────────────────────────────────────┘
                                                                        */
. WHEN(IPLFINAL)
.   DO;
.   . IF FSM_ALS_SEC_IPL_RES = PEND_RESET THEN               /* PAGE 11-123     */
.   .   DO;
.   .   . CALL UPM_RESTORE_SNF;                             /* PAGE 11-115     */
.   .   . SEND MU TO SNS.SEND;                              /* CHAPTER 6       */
.   .   . CALL FSM_ALS_SEC_IPL_RES;                         /* PAGE 11-123     */
.   .   . CALL DELETE_ALL_CP_ENTRIES(ALS_EA);               /* APPENDIX B      */
.   .   END;
.   . ELSE
.   .   DO;
.   .   . DISCARD MU;
.   .   . CALL NS.INOP_PROC(ALS_EA);                        /* PAGE 11-90      */
.   .   END;
.   END;
```

```
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │  POSITIVE AND NEGATIVE RESPONSES TO DUMPINIT  │
                    └──────────────────────────────────────────┘
                                                                        */
. WHEN(DUMPINIT)
.   DO;
.   . IF FSM_ALS_SEC_DUMP_RES = PEND_INDUMP_INIT THEN          /* PAGE 11-122          */
.   .   DO;
.   .   . CALL UPM_RESTORE_SNF;                                /* PAGE 11-115          */
.   .   . SEND MU TO SNS.SEND;                                 /* CHAPTER 6            */
.   .   . CALL FSM_ALS_SEC_DUMP_RES;                           /* PAGE 11-122          */
.   .   END;
.   . ELSE
.   .   DO;
.   .   . DISCARD MU;
.   .   . CALL NS.INOP_PROC(ALS_EA);                           /* PAGE 11-90           */
.   .   END;
.   END;
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │  POSITIVE OR NEGATIVE RESPONSES TO DUMPTEXT  │
                    └──────────────────────────────────────────┘
                                                                        */
. WHEN(DUMPTEXT)
.   DO;
.   . IF FSM_ALS_SEC_DUMP_RES = PEND_INDUMP_TEXT THEN          /* PAGE 11-122          */
.   .   DO;
.   .   . CALL UPM_RESTORE_SNF;                                /* PAGE 11-115          */
.   .   . SEND MU TO SNS.SEND;                                 /* CHAPTER 6            */
.   .   . CALL FSM_ALS_SEC_DUMP_RES;                           /* PAGE 11-122          */
.   .   END;
.   . ELSE
.   .   DO;
.   .   . DISCARD MU;
.   .   . CALL NS.INOP_PROC(ALS_EA);                           /* PAGE 11-90           */
.   .   END;
.   END;
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │  POSITIVE OR NEGATIVE RESPONSES TO DUMPFINAL  │
                    └──────────────────────────────────────────┘
                                                                        */
. WHEN(DUMPFINAL)
.   DO;
.   . IF FSM_ALS_SEC_DUMP_RES = PEND_RESET THEN                /* PAGE 11-122          */
.   .   DO;
.   .   . CALL UPM_RESTORE_SNF;                                /* PAGE 11-115          */
.   .   . SEND MU TO SNS.SEND;                                 /* CHAPTER 6            */
.   .   . CALL FSM_ALS_SEC_DUMP_RES;                           /* PAGE 11-122          */
.   .   . CALL DELETE_ALL_CP_ENTRIES(ALS_EA);                  /* APPENDIX B           */
.   .   END;
.   . ELSE
.   .   DO;
.   .   . DISCARD MU;
.   .   . CALL NS.INOP_PROC(ALS_EA);                           /* PAGE 11-90           */
.   .   END;
.   END;
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │  POSITIVE OR NEGATIVE RESPONSES TO RPO  │
                    └──────────────────────────────────────────┘
                                                                        */
. WHEN(RPO)
.   DO;
.   . IF FSM_ALS_SEC_RPO_RES = PEND THEN                       /* PAGE 11-123          */
.   .   DO;
.   .   . CALL UPM_RESTORE_SNF;                                /* PAGE 11-115          */
.   .   . SEND MU TO SNS.SEND;                                 /* CHAPTER 6            */
.   .   . CALL FSM_ALS_SEC_RPO_RES;                            /* RPO      PAGE 11-123 */
.   .   . CALL DELETE_ALL_CP_ENTRIES(ALS_EA);                  /* APPENDIX B           */
.   .   END;
.   . ELSE
.   .   DO;
.   .   . DISCARD MU;
.   .   . CALL NS.INOP_PROC(ALS_EA);                           /* PAGE 11-90           */
.   .   END;
.   END;
  END;
  RETURN;
END NS.LOAD_RSP;
```

```
NS.SIG_RSP_PRI: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  DEPENDING  ON THE  INPUT  SIGNAL THE  PERTINENT  RESOURCE FSM'S  ARE │
│                 CHECKED TO  SEE IF THE  INPUT IS EXPECTED.   IF IT IS  EXPECTED, THE │
│                 INPUT IS PROCESSED; OTHERWISE, IT IS IGNORED.            │
│                                                                          │
│      INPUT:     CONNECT_IN_SUCCESSFUL, CONNECT_OUT_SUCCESSFUL,           │
│                 LINK_RESET_COMPLETE, ALS_RESET_COMPLETE,  AND XID_COMPLETED  SIGNALS │
│                 FROM NS.DLC_RCV (PAGE 11-76)                             │
│                                                                          │
│      OUTPUT:    XID TO DLC, REQCONT TO SNS.SEND                          │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 NS.DLC_RCV                      PAGE 11-76               │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 FSM_ALS_CONNECTED_RES           PAGE 11-121              │
│                 FSM_ALS_CONTACT_DISCONTACT_RES  PAGE 11-122              │
│                 FSM_ALS_SEC_XID_RES             PAGE 11-124              │
│                 FSM_LINK_ACT_RES                PAGE 11-119              │
│                 FSM_LINK_CONNIN_RES             PAGE 11-120              │
│                 FSM_LINK_CONNOUT_RES            PAGE 11-121              │
│                 NS.INOP_PROC                    PAGE 11-90               │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL ALS_EA BIT(16);
DCL LINK_EA BIT(16);
DCL SAVE_ALS_PTR PTR;

NRCB_PTR = FIND_ALS_FOR_RESOURCE(LSCB.EA);                  /* APPENDIX B      */
ALS_EA = NRCB.ELEMENT_ADDRESS;
SAVE_ALS_PTR = NRCB_PTR;
NRCB_PTR = FIND_LINK_FOR_RESOURCE(LSCB.EA);                 /* APPENDIX B      */
LINK_EA = NRCB.ELEMENT_ADDRESS;
CP_INDIRECT_PTR = FIRST_ENTRY(NRCB.CP_INDIRECT_LIST);
CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
SCB_PTR = CPCB.CP_SCB_ID;

SELECT ANYORDER;
.
. WHEN(INPUT('CONNECT_IN_SUCCESSFUL'))
.   DO;
.   . IF FSM_LINK_CONNIN_RES = ACTIVE THEN                  /* PAGE 11-120     */
.   .   DO;
.   .   . SEND 'XID' TO PU.SVC_MGR.LINK_MGR;
.   .   . NRCB_PTR = SAVE_ALS_PTR;
.   .   . CALL FSM_ALS_SEC_XID_RES('XID');                  /* PAGE 11-124     */
.   .   . CALL FSM_ALS_CONNECTED_RES('CONNECTED');          /* PAGE 11-121     */
.   .   END;
.   . ELSE
.   .   CALL NS.INOP_PROC(LINK_EA);                         /* PAGE 11-90      */
.   END;
.
. WHEN(INPUT('CONNECT_OUT_SUCCESSFUL'))
.   DO;
.   . IF FSM_LINK_CONNOUT_RES = ACTIVE THEN                 /* PAGE 11-121     */
.   .   DO;
.   .   . CALL FSM_LINK_CONNOUT_RES('CONNECT_OUT_SUCCESSFUL');  /* PAGE 11-121 */
.   .   . SEND 'XID' TO PU.SVC_MGR.LINK_MGR;
.   .   . NRCB_PTR = SAVE_ALS_PTR;
.   .   . CALL FSM_ALS_SEC_XID_RES('XID');                  /* PAGE 11-124     */
.   .   . CALL FSM_ALS_CONNECTED_RES('CONNECTED');          /* PAGE 11-121     */
.   .   END;
.   . ELSE
.   .   CALL NS.INOP_PROC(LINK_EA);                         /* PAGE 11-90      */
.   END;
.
. WHEN(INPUT('XID_COMPLETED'))
.   DO;
.   . NRCB_PTR = SAVE_ALS_PTR;
.   . CALL FSM_ALS_SEC_XID_RES('XID_COMPLETED');            /* PAGE 11-124     */
.   . CP_INDIRECT_PTR = FIRST_ENTRY(NRCB.CP_INDIRECT_LIST);
.   . IF CP_INDIRECT_PTR ¬= NULL THEN
.   .   DO;
.   .   . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
.   .   . SCB_PTR = CPCB.CP_SCB_ID;
.   .   . IF SCB_PTR ¬= NULL THEN
.   .   .   DO;
.   .   .   . MU_PTR = UPM_CREATE_RQ('REQCONT');           /* APPENDIX B      */
.   .   .   . SEND MU TO SNS.SEND;                         /* CHAPTER 6       */
.   .   .   END;
.   .   . ELSE
.   .   .   DISCARD MU;
.   .   END;
.   . ELSE
.   .   DISCARD MU;
.   END;
.
```

```
   . WHEN(INPUT('LINK_RESET_COMPLETE'))
   .   DO;
   .   . CALL PSM_LINK_ACT_RES('LINK_RESET_COMPLETE');          /* PAGE 11-119          */
   .   . CALL DEQUEUE_BUS_FROM_RESOURCE(LINK_EA);              /* APPENDIX B           */
   .   END;
   .
   . WHEN(INPUT('ALS_RESET_COMPLETE'))
   .   DO;
   .   . MRCB_PTR = SAVE_ALS_PTR;
   .   . CALL PSM_ALS_CONTACT_DISCONTACT_RES('ALS_RESET_COMPLETE'); /* PAGE 11-122          */
   .   . SEND 'REX_INOP' TO PU.SVC_MGR.CSC_MGR.SON;            /* CHAPTER 13           */
   .   . CALL DEQUEUE_BUS_FROM_RESOURCE(ALS_EA);               /* APPENDIX B           */
   .   END;
   END;

   RETURN;
END MS.SIG_RSP_PRI;
```

```
NS.SIG_RSP_SEC: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  FOR   'CONNECT_OUT_SUCCESSFUL'   AND   'CONNECT_IN_SUCCESSFUL'   THE │
│                 APPROPRIATE (CONNOUT OR CONNIN) LINK_EA RESOURCE FSM IS CHECKED.  IF │
│                 THE  STATE  IS  CORRECT,  THE  SIGNAL  IS  SENT  TO  THAT  FSM,  AND │
│                 'CONNECTED' IS SENT TO THE CONNECTED RESOURCE FSM FOR THE ASSOCIATED │
│                 ADJACENT LINK STATION.  OTHERWISE NS.INOP_PROC IS CALLED TO GENERATE │
│                 AN  INOP   REQUEST  AND  ROUTE   THE  REQUEST  TO   THE  APPROPRIATE │
│                 HALF-SESSIONS.  WHEN 'XID' IS RECEIVED,  THE CONNOUT RECEIVE FSM FOR │
│                 THIS SECONDARY  LINK STATION  IS CHECKED.  IF  NOT ACTIVE,  THEN THE │
│                 CONNIN  FSM IS  CHECKED.  IF  EITHER THE  CONNIN OR  CONNOUT FSM  IS │
│                 ACTIVE, THEN REQCONT IS IS GENERATED  AND SENT TO SNS.SEND.  FOR THE │
│                 RESET-COMPLETE  SIGNALS  THE  RESOURCE  FSM  IS  RESET,  AND  QUEUED │
│                 REQUESTS ARE  RETRIED.  A  ROUTE EXTENSION  INOP SIGNAL  IS SENT  TO │
│                 COMMON SESSION CONTROL FOR SESSION OUTAGE NOTIFICATION.              │
│                                                                                      │
│      INPUT:     CONNECT_OUT_SUCCESSFUL, CONNECT_IN_SUCCESSFUL, XID,                  │
│                 LINK_RESET_COMPLETE, OR  ALS_RESET_COMPLETE SIGNALS  FROM NS.DLC_RCV │
│                 (PAGE 11-76)                                                         │
│                                                                                      │
│      OUTPUT:    REQCONT TO SNS.SEND; XID TO DLC                                      │
│                                                                                      │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                       │
│                 NS.DLC_RCV                          PAGE 11-76                       │
│                                                                                      │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                                           │
│                 FSM_ALS_CONNECTED_RES               PAGE 11-121                      │
│                 FSM_ALS_CONTACT_DISCONTACT_RES      PAGE 11-122                      │
│                 FSM_LINK_ACT_RES                    PAGE 11-119                      │
│                 FSM_LINK_CONNIN_RES                 PAGE 11-120                      │
│                 FSM_LINK_CONNOUT_RES                PAGE 11-121                      │
│                 NS.INOP_PROC                        PAGE 11-90                       │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL ALS_EA BIT(16);
DCL LINK_EA BIT(16);
DCL SAVE_ALS_PTR PTR;

NRCB_PTR = FIND_ALS_FOR_RESOURCE(LSCB.EA);                  /* APPENDIX B      */
SAVE_ALS_PTR = NRCB_PTR;
ALS_EA = NRCB.ELEMENT_ADDRESS;
NRCB_PTR = FIND_LINK_FOR_RESOURCE(LSCB.EA);                 /* APPENDIX B      */
LINK_EA = NRCB.ELEMENT_ADDRESS;

SELECT ANYORDER;
.
. WHEN(INPUT('CONNECT_IN_SUCCESSFUL'))
.   DO;
.   . IF FSM_LINK_CONNIN_RES = ACTIVE THEN                  /* PAGE 11-120     */
.   .   DO;
.   .   . NRCB_PTR = SAVE_ALS_PTR;
.   .   . CALL FSM_ALS_CONNECTED_RES('CONNECTED');          /* PAGE 11-121     */
.   .   END;
.   . ELSE
.   .   CALL NS.INOP_PROC(LINK_EA);
.   END;
.
. WHEN(INPUT('CONNECT_OUT_SUCCESSFUL'))
.   DO;
.   . IF FSM_LINK_CONNOUT_RES = ACTIVE THEN                 /* PAGE 11-121     */
.   .   DO;
.   .   . CALL FSM_LINK_CONNOUT_RES('CONNECT_OUT_SUCCESSFUL'); /* PAGE 11-121   */
.   .   . NRCB_PTR = SAVE_ALS_PTR;
.   .   . CALL FSM_ALS_CONNECTED_RES('CONNECTED');          /* PAGE 11-121     */
.   .   END;
.   . ELSE
.   .   CALL NS.INOP_PROC(LINK_EA);                         /* PAGE 11-90      */
.   END;
.
. WHEN(INPUT('XID'))
.   DO;
.   . NRCB_PTR = SAVE_ALS_PTR;
.   . IF FSM_ALS_CONNECTED_RES = ACTIVE THEN                /* PAGE 11-121     */
.   .   DO;
.   .   . MU_PTR = UPM_CREATE_RQ('REQCONT');                /* APPENDIX B      */
.   .   . CP_INDIRECT_PTR = FIRST_ENTRY(NRCB.CP_INDIRECT_LIST);
.   .   . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
.   .   . SCB_PTR = CPCB.CP_SCB_ID;
.   .   . SEND MU TO SNS.SEND;                              /* CHAPTER 6       */
.   .   . SEND 'XID' TO PU.SVC_MGR.LINK_MGR;
.   .   END;
.   . ELSE
.   .   CALL NS.INOP_PROC(LINK_EA);                         /* PAGE 11-90      */
.   END;
```

```
   . WHEN(INPUT('LINK_RESET_COMPLETE'))
   .   DO;
   .   . CALL FSM_LINK_ACT_RES('LINK_RESET_COMPLETE');          /* PAGE 11-119          */
   .   . CALL DEQUEUE_RUS_FROM_RESOURCE(LINK_EA);              /* APPENDIX B           */
   .   END;
   .
   . WHEN(INPUT('ALS_RESET_COMPLETE'))
   .   DO;
   .   . NRCB_PTR = SAVE_ALS_PTR;
   .   . CALL FSM_ALS_CONTACT_DISCONTACT_RES('ALS_RESET_COMPLETE');  /* PAGE 11-122      */
   .   . SEND 'REX_INOP' TO PU.SVC_MGR.CSC_MGR.SON;            /* CHAPTER 13           */
   .   . CALL DEQUEUE_RUS_FROM_RESOURCE(ALS_EA);               /* APPENDIX B           */
   .   END;
   END;

   RETURN;
END NS.SIG_RSP_SEC;
```

```
NS.INOP_PROC: PROCEDURE(LINK_STA_EA);
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE RESETS THE APPROPRIATE RESOURCE FSM'S FOR THE TYPE OF  │
   │                 INOP RECEIVED.  IT THEN SENDS A COPY OF THE INOP REQUEST TO ALL CP'S  │
   │                 IN THE RESOURCE'S CP_LIST.                           │
   │                                                                      │
   │      INPUT:     LINK OR ADJACENT LINK STATION ADDRESS TO BE "INOP'ED" FROM  │
   │                 NS.CONN_RSP,     NS.SIG_RSP_PRI,     NS.DLC_RCV,     NS.SIG_RSP_SEC,  │
   │                 NS.CONTACT_RSP_PRI,   NS.LINK_RSP,    NS.CONTACT_RSP_REPLY_SEC,  OR  │
   │                 NS.LOAD_RSP                                          │
   │                                                                      │
   │      OUTPUT:    INOP IS SENT TO SNS.SEND FOR THE HALF SESSION OF EACH CP IN THE  │
   │                 CP_LIST.   REX_INOP (ROUTE EXTENSION INOPERATIVE) SIGNAL FOR EACH  │
   │                 ADJACENT LINK STATION AFFECTED IS SENT TO CSC_MGR.SON.  │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 FSM_XID_FORMAT_2                 PAGE 11-126         │
   │                 NS.CONN_RSP                      PAGE 11-82          │
   │                 NS.CONTACT_RSP                   PAGE 11-80          │
   │                 NS.DLC_CONFIG                    PAGE 11-66          │
   │                 NS.DLC_RCV                       PAGE 11-76          │
   │                 NS.LINK_RSP                      PAGE 11-79          │
   │                 NS.LOAD_RSP                      PAGE 11-84          │
   │                 NS.SIG_RSP_PRI                   PAGE 11-86          │
   │                 NS.SIG_RSP_SEC                   PAGE 11-88          │
   │                                                                      │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
   │                 INOP_TO_HALF_SESSIONS            PAGE 11-91          │
   │                 NS.ALS_RESET                     PAGE 11-95          │
   │                 NS.LINK_RESET                    PAGE 11-94          │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL LINK_STA_EA BIT(16);
  DCL ALS_EA BIT(16);
  DCL LINK_EA BIT(16);

  NRCB_PTR = LOCATE_NODE_RESOURCE(LINK_STA_EA);              /* APPENDIX B      */
  SELECT ANYORDER(NRCB.RESOURCE_CATEGORY);
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │              INOP LINK               │
                  └──────────────────────────────────────┘
                                                                        */
  .
  .
  . WHEN(LINK)
  .
  .   DO;
  .   . LINK_EA = LINK_STA_EA;
  .   .
  .   . CALL INOP_TO_HALF_SESSIONS(LINK_EA);                 /* PAGE 11-91      */
  .   .
  .   . CALL NS.LINK_RESET(LINK_EA,INOP_RQ.INOP_REASON);     /* PAGE 11-94      */
  .   .
  .   . SCAN NRCB_LIST PTR(NRCB_PTR);
  .   . .
  .   . . IF NRCB.ASSOCIATED_RESOURCE = LINK_EA THEN
  .   . .    DO;
  .   . .    . FIND LSCB IN LSCB_LIST
  .   . .    .      WHERE(LSCB.EA = NRCB.ELEMENT_ADDRESS);
  .   . .    . SEND 'REX_INOP' TO PU.SVC_MGR.CSC_MGR.SON;     /* CHAPTER 13      */
  .   . .    END;
  .   . .
  .   . SCANEND;
  .   . NRCB_PTR = FIND_ALS_FOR_RESOURCE(LINK_EA);            /* APPENDIX B      */
  .   . ALS_EA = NRCB.ELEMENT_ADDRESS;
  .   . TGCB_PTR = FIND_TGCB_FOR_ALS_EA(ALS_EA);             /* APPENDIX B      */
  .   . IF TGCB_PTR ¬= NULL THEN
  .   .   DO;
  .   .   . CALL DELETE_ALS_FROM_TGCB(ALS_EA);               /* APPENDIX B      */
  .   .   . IF EMPTY(TGCB.ASSOC_LSCB_LIST) THEN
  .   .   .   SEND 'TG_INOP_ERROR' TO PU.SVC_MGR.PC_ROUTE_MGR.RCV;/* CHAPTER 12  */
  .   .   END;
  .   .
  .   END;
```

```
                                                                        /*
                    ┌──────────────────────────────────┐
                    │  INOP ADJACENT LINK STATION      │
                    └──────────────────────────────────┘
                                                                        */
  .
  .
  . WHEN(ALS)
  .
  .   DO;
  .   . ALS_EA = LINK_STA_EA;
  .   . CALL INOP_TO_HALF_SESSIONS(ALS_EA);                  /* PAGE 11-91   */
  .   .
  .   . CALL NS.ALS_RESET(ALS_EA);                           /* PAGE 11-95   */
  .   .
  .   . TGCB_PTR = FIND_TGCB_FOR_ALS_EA(ALS_EA);             /* APPENDIX B   */
  .   . IF TGCB_PTR ¬= NULL THEN
  .   .   DO;
  .   .   . CALL DELETE_ALS_FROM_TGCB(ALS_EA);               /* APPENDIX B   */
  .   .   . IF EMPTY(TGCB.ASSOC_LSCB_LIST) THEN
  .   .   .   SEND 'TG_INOP_ERROR' TO PU.SVC_MGR.PC_ROUTE_MGR.RCV;/* CHAPTER 12 */
  .   .   END;
  .   . SEND 'REX_INOP' TO PU.SVC_MGR.CSC_MGR.SON;           /* CHAPTER 13   */
  .   .
  .   END;
  END;
  RETURN;
END NS.INOP_PROC;



INOP_TO_HALF_SESSIONS: PROCEDURE(LINK_STA_EA);
                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE SENDS A COPY OF THE INOP TO EACH CP IN THE RESOURCE'S
  │                 CPCB_LIST.                                               │
  │                                                                          │
  │      INPUT:     THE INOP TO BE SENT                                      │
  │                                                                          │
  │      OUTPUT:    THE INOP TO ALL CP'S THAT ARE IN THE RESOURCE'S CPCB_LIST │
  │                                                                          │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
  │                 NS.INOP_PROC                          PAGE 11-90         │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL LINK_STA_EA BIT(16);

  NRCB_PTR = LOCATE_NODE_RESOURCE(LINK_STA_EA);              /* APPENDIX B   */

  SCAN NRCB.CP_INDIRECT_LIST PTR(CP_INDIRECT_PTR);
  . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
  . MU_PTR = UPM_CREATE_RQ('INOP');                          /* APPENDIX B   */
  . SCB_PTR = CPCB.CP_SCB_ID;
  . SEND MU TO SNS.SEND;                                     /* CHAPTER 6    */
  SCANEND;

  RETURN;
END INOP_TO_HALF_SESSIONS;
```

```
NS.REQFNA_PROC: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE RECEIVES CONTROL FROM A UPM.  THIS PROCEDURE VERIFIES │
  │                 THAT THE ADDRESS IN  THE REQUEST IS VALID AND THAT  THE SESSIONS ARE │
  │                 NOT ACTIVE PRIOR TO SENDING THE  REQUEST TO SNS.SEND.  IF THE CHECKS │
  │                 FAIL A  -RSP IS  RETURNED TO  THE SENDER.   ALL APPROPRIATE  RH SEND │
  │                 CHECKS ARE MADE BY THE UPM.                                  │
  │                                                                              │
  │      INPUT:     THE REQFNA, WHICH IS THE CURRENT MESSAGE UNIT, FROM A UPM    │
  │                                                                              │
  │      OUTPUT:    THE REQFNA IS SENT TO SNS.SEND  IF THE CHECKS ARE PASSED; OTHERWISE, │
  │                 THE REQFNA IS RETURNED TO THE SENDER.                        │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 FSM_CP_SESS_SDT                  PAGE 11-119                 │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL P POINTER;
  DCL RC BIT(1);

  FIND CPCB IN CPCB_LIST WHERE(CPCB.CP_SCB_ID = SCB_PTR);
  IF CPCB_PTR = NULL THEN
    SEND SEND_CHECK(X'8005') TO SENDING_PROCEDURE;              /* NO SESSION          */
  ELSE
    IF FSM_CP_SESS_SDT ¬= ACTIVE THEN
      SEND SEND_CHECK(X'2005') TO SENDING_PROCEDURE;            /* DATA TRAFFIC RESET  */
    ELSE
      DO;
      . NRCB_PTR = LOCATE_NODE_RESOURCE(REQFNA_RQ.LU_ADDRESS);  /* APPENDIX B          */
      .
      . IF NRCB_PTR = NULL THEN
      .   SEND SEND_CHECK(X'0806') TO SENDING_PROCEDURE;        /* RESOURCE UNKNOWN    */
      .
      . ELSE
      .   DO;
      .   .
      .   . IF RRI = RQ THEN
      .   .   SEND MU TO SNS.SEND;                              /* CHAPTER 6           */
      .   . ELSE
      .   .   SEND SEND_CHECK(X'0809') TO SENDING_PROCEDURE;    /* MODE INCONSISTENCY  */
      .   END;
      END;
  RETURN;

END NS.REQFNA_PROC;
```

```
NS.REQACTLU_PROC: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE RECEIVES CONTROL FROM  A UPM.  THIS PROCEDURE OBTAINS │
  │                 A NETWORK ADDRESS FOR  THE LU THAT IS TO BE  ACTIVATED AND SENDS THE │
  │                 REQUEST TO SNS.SEND.  ALL APPROPRIATE RH SEND CHECKS ARE MADE BY THE │
  │                 UPM.                                                         │
  │                                                                              │
  │      INPUT:     REQACTLU, WHICH IS THE CURRENT MESSAGE, FROM A UPM           │
  │                                                                              │
  │      OUTPUT:    THE REQACTLU  LU IS  ADDED TO THE  NRCB_LIST AND THE  RU IS  SENT TO │
  │                 SNS.SEND.                                                    │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 FSM_CP_SESS_SDT                    PAGE 11-119               │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  FIND CPCB IN CPCB_LIST WHERE(CPCB.CP_SCB_ID = SCB_PTR);
  IF CPCB_PTR = NULL THEN
    SEND SEND_CHECK(X'8005') TO SENDING_PROCEDURE;              /* NO SESSION      */
  ELSE
    IF FSM_CP_SESS_SDT ¬= ACTIVE THEN
      SEND SEND_CHECK(X'2005') TO SENDING_PROCEDURE;            /* DATA TRAFFIC RESET */
    ELSE
      DO;
      . CREATE NRCB PTR(NRCB_PTR);
      .
      . IF NRCB_PTR = NULL THEN
      .   SEND SEND_CHECK(X'0812') TO SENDING_PROCEDURE;        /* INSUFFICIENT RESOURCE */
      .
      . ELSE
      .   DO;
      .   . NRCB.RESOURCE_CATEGORY = LU;
      .   . NRCB.ELEMENT_ADDRESS = REQACTLU_RQ.LU_ADDRESS & NCB.NODE_ELEMENT_MASK;
      .   . NRCB.ASSIGNING_CP_SCB_ID = SCB_PTR;
      .   . INSERT NRCB IN NRCB_LIST;
      .   .
      .   . SEND RU TO SNS.SEND;                                /* CHAPTER 6       */
      .   END;
      END;

  RETURN;

END NS.REQACTLU_PROC;
```

```
NS.LINK_RESET: PROCEDURE(LINK_EA,RESET_REASON);

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  RESETS THE LINK_EA.LINK_(PRI|SEC)_SUBTREE SHOWN IN FIGURE 11-4.
  │
  │      INPUT:     LINK_EA  CARRIES THE  ELEMENT  ADDRESS OF  THE  LINK  FOR WHICH  THE
  │                 SUBTREE IS TO  BE RESET.  THE RESET_REASON  SPECIFIES THE PARTICULAR
  │                 CAUSE OF THE RESET.
  │
  │      OUTPUT:    FSM'S ASSOCIATED  WITH THE LINK AND  ITS ADJACENT LINK  STATIONS ARE
  │                 RESET.
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 NS.INOP_PROC                    PAGE 11-90
  │                 NS.LCP_RESET_PROC               PAGE 11-33
  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):
  │                 FSM_LINK_ACT_RES                PAGE 11-119
  │                 FSM_LINK_CONNIN_RES             PAGE 11-120
  │                 FSM_LINK_CONNOUT_RES            PAGE 11-121
  │                 FSM_LINK_TRACE_RES              PAGE 11-120
  │                 NS.ALS_RESET                    PAGE 11-95
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL LINK_EA BIT(16);
  DCL RESET_REASON BIT(4);
  DCL SAVE_NRCB_PTR PTR;

  SAVE_NRCB_PTR = NRCB_PTR;
  NRCB_PTR = LOCATE_NODE_RESOURCE(LINK_EA);                  /* APPENDIX B          */

  IF FSM_LINK_ACT_RES = RESET_IN_PROGRESS THEN               /* PAGE 11-90          */
    DO;
    . NRCB_PTR = SAVE_NRCB_PTR;
    . RETURN;
    END;

  SCAN NRCB_LIST PTR(NRCB_PTR);
  .
  . IF NRCB.ASSOCIATED_RESOURCE = LINK_EA THEN
  .   DO;
  .   .
  .   . CALL NS.ALS_RESET(NRCB.ELEMENT_ADDRESS);            /* PAGE 11-95          */
  .   .
  .   END;
  .
  SCAN END;

  NRCB_PTR = LOCATE_NODE_RESOURCE(LINK_EA);                  /* APPENDIX B          */


  IF RESET_REASON = LINK_FAILURE THEN
    DO;
    . CALL FSM_LINK_ACT_RES('LINK_RESET');                   /* PAGE 11-119         */
    .
    . CALL PURGE_RUS_FROM_RESOURCE(LINK_EA);                 /* APPENDIX B          */
    .
    . SEND 'LINK_RESET' TO PU.SVC_MGR.LINK_MGR;
    .
    . CALL DELETE_ALL_CP_ENTRIES(LINK_EA);                   /* APPENDIX B          */
    END;

  CALL FSM_LINK_TRACE_RES('RESET');                          /* PAGE 11-120         */
  CALL FSM_LINK_CONNOUT_RES('RESET');                        /* PAGE 11-121         */
  CALL FSM_LINK_CONNIN_RES('RESET');                         /* PAGE 11-120         */

  NRCB_PTR = SAVE_NRCB_PTR;
  RETURN;
END NS.LINK_RESET;
```

```
NS.ALS_RESET: PROCEDURE(ALS_EA);

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  SEARCHES THE NRCB_LIST TO FIND THE ADJACENT LINK     │
 │                 STATIONS SPECIFIED. THE ALS_TEST_RES FSM AND THE     │
 │                 ALS_CONNECTED_RES FSM ARE RESET FOR EACH ADJACENT    │
 │                 STATION FOUND. OTHER ADJACENT LINK STATION FSM'S     │
 │                 ARE RESET BY A CALL TO NS.ALS_PROC_RESET. IF IT IS   │
 │                 A CONFIGURABLE STATION, THE FSM_TGN AND              │
 │                 FSM_XID_FORMAT_2 FSM'S ARE RESET. A LINK-LEVEL       │
 │                 RESET IS INITIATED IF A RESET IS NOT ALREADY IN      │
 │                 PROGRESS.                                            │
 │                                                                      │
 │      INPUT:     ALS_EA CONTAINS THE ELEMENT ADDRESS OF THE ALS TO    │
 │                 BE RESET.                                            │
 │                                                                      │
 │      OUTPUT:    RESET TO FSM'S OF THE ADJACENT LINK STATIONS         │
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 NS.CONN_PROC                    PAGE 11-40           │
 │                 NS.INOP_PROC                    PAGE 11-90           │
 │                 NS.LCP_RESET_PROC               PAGE 11-33           │
 │                 NS.LINK_RESET                   PAGE 11-94           │
 │                                                                      │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
 │                 FSM_ADJ_PU_LOAD                 PAGE 11-124          │
 │                 FSM_ALS_CONNECTED_RES           PAGE 11-121          │
 │                 FSM_ALS_CONTACT_DISCONTACT_RES  PAGE 11-122          │
 │                 FSM_ALS_TEST_RES                PAGE 11-124          │
 │                 FSM_TGN                         PAGE 11-125          │
 │                 FSM_XID_FORMAT_2                PAGE 11-126          │
 │                 NS.ALS_PROC_RESET               PAGE 11-96           │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL ALS_EA BIT(16);
  DCL SAVE_NRCB_PTR PTR;

  SAVE_NRCB_PTR = NRCB_PTR;
  NRCB_PTR = FIND_ALS_FOR_RESOURCE(ALS_EA);                  /* APPENDIX B      */
  FIND LSCB IN LSCB_LIST WHERE(LSCB.EA = ALS_EA);

  CALL FSM_ALS_CONNECTED_RES('RESET');                       /* PAGE 11-121     */
  CALL FSM_ALS_TEST_RES('RESET');                            /* PAGE 11-120     */

  CALL FSM_ADJ_PU_LOAD('RESET');                             /* PAGE 11-124     */

  CALL NS.ALS_PROC_RESET(ALS_EA);                            /* PAGE 11-96      */

  IF NRCB.PRI_SEC_ROLE = CONFIGURABLE THEN
    DO;
    . CALL FSM_TGN('RESET');                                 /* PAGE 11-125     */
    . CALL FSM_XID_FORMAT_2('RESET');                        /* PAGE 11-126     */
    END;

  IF FSM_ALS_CONTACT_DISCONTACT_RES ¬= RESET_IN_PROGRESS THEN /* PAGE 11-122    */
    DO;
    . CALL FSM_ALS_CONTACT_DISCONTACT_RES('ALS_RESET');      /* PAGE 11-122     */
    . SEND 'ALS_RESET' TO PU.SVC_MGR.LINK_MGR;
    END;

  NRCB_PTR = SAVE_NRCB_PTR;
END NS.ALS_RESET;
```

```
NS.ALS_PROC_RESET: PROCEDURE(ALS_EA);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  SEARCHES THE NRCB_LIST TO FIND THE ADJACENT LINK STATIONS SPECIFIED.  │
│                 THE  ALS_CONTACT_DISCONTACT_RES  FSM  IS   RESET;  IN  ADDITION  FOR  │
│                 SECONDARY STATIONS, THE FSM'S FOR XID, IPL, DUMP, AND RPO ARE RESET. │
│                                                                          │
│      INPUT:     ALS_EA CONTAINS THE ELEMENT ADDRESS OF THE ALS.           │
│                                                                          │
│      OUTPUT:    RESET TO FSM'S OF THE ADJACENT LINK STATION               │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 NS.ALS_RESET                        PAGE 11-95           │
│                 NS.DUMP_PROC                        PAGE 11-48           │
│                 NS.LOAD_PROC                        PAGE 11-46           │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 FSM_ALS_CONTACT_DISCONTACT_RES      PAGE 11-122          │
│                 FSM_ALS_SEC_DUMP_RES                PAGE 11-122          │
│                 FSM_ALS_SEC_IPL_RES                 PAGE 11-123          │
│                 FSM_ALS_SEC_RPO_RES                 PAGE 11-123          │
│                 FSM_ALS_SEC_XID_RES                 PAGE 11-124          │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL ALS_EA BIT(16);
  DCL SAVE_NRCB_PTR PTR;

  SAVE_NRCB_PTR = NRCB_PTR;
  NRCB_PTR = FIND_ALS_FOR_RESOURCE(ALS_EA);                  /* APPENDIX B      */

  CALL FSM_ALS_CONTACT_DISCONTACT_RES('RESET');              /* PAGE 11-122     */

  IF NRCB.LINK_DLC_ROLE = SECONDARY THEN
    DO;
    . CALL FSM_ALS_SEC_XID_RES('RESET');                     /* PAGE 11-124     */
    .
    . CALL FSM_ALS_SEC_IPL_RES('RESET');                     /* PAGE 11-123     */
    .
    . CALL FSM_ALS_SEC_DUMP_RES('RESET');                    /* PAGE 11-122     */
    .
    . CALL FSM_ALS_SEC_RPO_RES('RESET');                     /* PAGE 11-123     */
    .
    END;

  CALL DELETE_ALL_CP_ENTRIES(ALS_EA);                        /* APPENDIX B      */

  NRCB_PTR = SAVE_NRCB_PTR;

  RETURN;

END NS.ALS_PROC_RESET;
```

```
ALS_SEC_SUBTREE_CHECK: PROCEDURE(ALS_EA) RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  CHECKS THAT THE  SEC_SUBTREE ASSOCIATED WITH ELEMENT  ADDRESS PASSED
  │                 IS IN THE RESET STATE.
  │
  │      INPUT:     THE ELEMENT ADDRESS OF THE ADJACENT LINK STATION TO BE CHECKED.
  │
  │      OUTPUT:    OK, IF THE ALS_SEC_SUBTREE IS RESET; NG, IF IT IS IN ANY OTHER STATE
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 DACTLINK_RCV_CHECKS                 PAGE 11-39
  │                 FNA_VALIDITY_CHECK                  PAGE 11-56
  │                 LOAD_CHECKS                         PAGE 11-104
  │                 NS.DELETENR_PROC                    PAGE 11-63
  │                 NS.RPO_PROC                         PAGE 11-50
  │                 NS.SETCV_PROC                       PAGE 11-64
  │                 NS.TESTMODE_PROC                    PAGE 11-109
  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):
  │                 FSM_ALS_CONTACT_DISCONTACT_RES      PAGE 11-122
  │                 FSM_ALS_SEC_DUMP_RES                PAGE 11-122
  │                 FSM_ALS_SEC_IPL_RES                 PAGE 11-123
  │                 FSM_ALS_SEC_RPO_RES                 PAGE 11-123
  │                 FSM_ALS_SEC_XID_RES                 PAGE 11-124
  │                 FSM_ALS_TEST_RES                    PAGE 11-124
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL ALS_EA BIT(16);
  DCL RC BIT(1);
  DCL SAVE_NRCB_PTR PTR;

  RC = OK;
  SAVE_NRCB_PTR = NRCB_PTR;
  NRCB_PTR = FIND_ALS_FOR_RESOURCE(ALS_EA);                  /* APPENDIX B      */

  IF ((FSM_ALS_CONTACT_DISCONTACT_RES ¬= RESET) |             /* PAGE 11-122     */
      (FSM_ALS_SEC_XID_RES ¬= RESET) |                       /* PAGE 11-124     */
      (FSM_ALS_SEC_IPL_RES ¬= RESET) |                       /* PAGE 11-123     */
      (FSM_ALS_SEC_DUMP_RES ¬= RESET) |                      /* PAGE 11-122     */
      (FSM_ALS_SEC_RPO_RES ¬= RESET) |                       /* PAGE 11-123     */
      (FSM_ALS_TEST_RES ¬= RESET)) THEN                      /* PAGE 11-124     */

    RC = NG;

  NRCB_PTR = SAVE_NRCB_PTR;
  RETURN(RC);
END ALS_SEC_SUBTREE_CHECK;
```

This page intentionally left blank

```
ALS_SEC_SUBTREE_INTERRUPT: PROCEDURE(ALS_EA) RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  CHECKS THAT THE  SEC_SUBTREE ASSOCIATED WITH ELEMENT  ADDRESS PASSED
  │                 IS IN AN INTERRUPTIBLE STATE.
  │
  │      INPUT:     THE ELEMENT ADDRESS OF THE ADJACENT LINK STATION TO BE CHECKED
  │
  │      OUTPUT:    OK, IF THE  ALS_SEC_SUBTREE IS IN AN INTERRUPTIBLE STATE;  NG, IF IT
  │                 IS NOT
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 NS.DUMP_PROC                        PAGE 11-48
  │                 NS.LOAD_PROC                        PAGE 11-46
  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):
  │                 PSM_ALS_SEC_DUMP_RES                PAGE 11-122
  │                 PSM_ALS_SEC_IPL_RES                 PAGE 11-123
  │                 PSM_ALS_SEC_RPO_RES                 PAGE 11-123
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL ALS_EA BIT(16);
  DCL RC BIT(1);
  DCL SAVE_NRCB_PTR PTR;

  RC = OK;
  SAVE_NRCB_PTR = NRCB_PTR;
  NRCB_PTR = FIND_ALS_FOR_RESOURCE(ALS_EA);                    /* APPENDIX B      */

  IF PSM_ALS_SEC_DUMP_RES = PEND_INDUMP_INIT |
       PSM_ALS_SEC_DUMP_RES = PEND_INDUMP_TEXT |
       PSM_ALS_SEC_DUMP_RES = PEND_RESET THEN                  /* PAGE 11-122     */
    RC = NG;

  IF PSM_ALS_SEC_IPL_RES = PEND_INIPL_INIT |
       PSM_ALS_SEC_IPL_RES = PEND_INIPL_TEXT |
       PSM_ALS_SEC_IPL_RES = PEND_RESET THEN                  /* PAGE 11-123     */
    RC = NG;

  IF PSM_ALS_SEC_RPO_RES = PEND THEN                           /* PAGE 11-123     */
    RC = NG;

  NRCB_PTR = SAVE_NRCB_PTR;
  RETURN(RC);
END ALS_SEC_SUBTREE_INTERRUPT;
```

```
PU_T2_LOAD_PROC: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS FUNCTION  TRACKS THE  LOAD OPERATION FOR  THE PU_T2.   THE LOAD │
 │                 OPERATION CAN BE ACCEPTED FROM THE SUBAREA PU OR FROM THE SSCP.  THE │
 │                 PU_T2_LOAD  FSM  MUST BE  IN  THE  RESET  STATE FOR  NC_IPL_INIT  OR │
 │                 NS_IPL_INIT TO BE ACCEPTED.  MANY IPL_TEXT REQUESTS MAY BE RECEIVED. │
 │                 AT LEAST  ONE IPL_TEXT  REQUEST MUST  BE RECEIVED  FOR THE  PU_T2 TO │
 │                 BEGIN EXECUTION OF THE LOAD MODULE.                  │
 │                                                                      │
 │      INPUT:     THE  CURRENT  MU  IS   AN  NC_IPL_INIT,  NC_IPL_TEXT,  NC_IPL_FINAL, │
 │                 NC_IPL_ABORT    NS_IPL_INIT,    NS_IPL_TEXT,     NS_IPL_FINAL,    OR │
 │                 NS_IPL_ABORT.                                        │
 │                                                                      │
 │      OUTPUT:    POSITIVE OR  NEGATIVE RESPONSE  TO PC_T2.SEND  OR SNS.SEND.   IN THE │
 │                 CASE THAT  AN NS_IPL_ABORT  OR AN NC_IPL_ABORT  IS RECEIVED  AND THE │
 │                 LOAD WAS  INITIATED BY  THE PU_T2  SENDING LDREQD,  A REQDISCONT  OR │
 │                 LDREQD IS SENT.                                      │
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 NS.CS_RCV                       PAGE 11-34           │
 │                 PU.SVC_MGR.NS.RCV               PAGE 11-28           │
 │                                                                      │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
 │                 FSM_PU_ACT_RES                  PAGE 11-118          │
 │                 FSM_PU_T2_LOAD                  PAGE 11-118          │
 │                 UPM_IPL_RQ_VALIDITY_CHECK       PAGE 11-115          │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL ADDRPRIME BIT(8);                   /* TO SWAP OAFPRIME AND DAFPRIME */
DCL RECEIVE_CHECK_SENSE BIT(32);

IF NCB.PU_TYPE ¬= T2 THEN
  IF RRI = RQ THEN
    DO;
    . CALL CHANGE_MU_TO_NEG_RSP(X'1003'); /* APPENDIX B, FUNCTION NOT SUPPORTED */
    . SEND MU TO SNS.SEND;                          /* CHAPTER 6          */
    END;
  ELSE
    DISCARD MU;
ELSE
  DO;
  . NRCB_PTR = LOCATE_NODE_RESOURCE(LSCB.EA);       /* APPENDIX B         */
  . IF DISPATCHED_BY(PC*) THEN
  .   DO;
  .   . ADDPPRIME = OAFPRIME;
  .   . OAFPRIME = DAFPRIME;
  .   . DAFPRIME = ADDRPRIME;
  .   END;
  .
  . SELECT ANYORDER;
                                                                        /*
                 ┌──────────────────────────────────────────────┐
                 │ NC_IPL_INIT FROM THE SUBAREA PU              │
                 └──────────────────────────────────────────────┘
                                                                        */
  . . WHEN(RQ_CODE = NC_IPL_INIT)
  . .   DO;
  . .   . IF FSM_PU_T2_LOAD ¬= RESET THEN           /* PAGES 11-118       */
  . .   .   RECEIVE_CHECK_SENSE = X'0809';          /* MODE INCONSISTENCY */
  . .   . ELSE
  . .   .   RECEIVE_CHECK_SENSE = UPM_IPL_RQ_VALIDITY_CHECK; /* PAGE 11-115 */
  . .   . IF RECEIVE_CHECK_SENSE ¬= 0 THEN
  . .   .   CALL CHANGE_MU_TO_NEG_RSP(RECEIVE_CHECK_SENSE); /* APPENDIX B   */
  . .   . ELSE
  . .   .   CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);  /* APPENDIX B         */
  . .   . CALL FSM_PU_T2_LOAD;                    /* PAGE 11-118        */
  . .   . SEND MU TO PC_T2.SEND;                  /* CHAPTER 3          */
  . .   END;
                                                                        /*
                 ┌──────────────────────────────────────────────┐
                 │ NC_IPL_TEXT OR NC_IPL_FINAL  FROM THE SUBAREA │
                 │ PU                                           │
                 └──────────────────────────────────────────────┘
                                                                        */
  . . WHEN(RQ_CODE = (NC_IPL_TEXT | NC_IPL_FINAL))
  . .   DO;
  . .   . IF FSM_PU_T2_LOAD ¬= NC_TEXT THEN        /* PAGE 11-118        */
  . .   .   RECEIVE_CHECK_SENSE = X'0809';          /* MODE INCONSISTENCY */
  . .   . ELSE
  . .   .   RECEIVE_CHECK_SENSE = UPM_IPL_RQ_VALIDITY_CHECK; /* PAGE 11-115 */
  . .   . IF RECEIVE_CHECK_SENSE ¬= 0 THEN
  . .   .   CALL CHANGE_MU_TO_NEG_RSP(RECEIVE_CHECK_SENSE); /* APPENDIX B   */
  . .   . ELSE
  . .   .   CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);  /* APPENDIX B         */
  . .   . CALL FSM_PU_T2_LOAD;                    /* PAGE 11-118        */
  . .   . IF RQ_CODE = NC_IPL_FINAL THEN
  . .   .   CALL FSM_PU_ACT_RES;                  /* PAGE 11-54         */
  . .   . SEND MU TO PC_T2.SEND;                  /* CHAPTER 3          */
  . .   END;
```

```
                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │ NC_IPL_ABORT FROM THE SUBAREA PU          │
                    └──────────────────────────────────────────┘
                                                                                */
   . . WHEN(RQ_CODE = NC_IPL_ABORT)
   . .   DO;
   . .   . IF FSM_PU_T2_LOAD = RESET THEN                    /* PAGE 11-118     */
   . .   .   DO;
   . .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'0809');           /* APPENDIX B      */
   . .   .   . SEND MU TO PC_T2.SEND;                        /* CHAPTER 3       */
   . .   .   END;
   . .   . ELSE
   . .   .   DO;
   . .   .   . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);          /* APPENDIX B      */
   . .   .   . CALL FSM_PU_T2_LOAD;                          /* PAGE 11-118     */
   . .   .   . SEND MU TO PC_T2.SEND;                        /* CHAPTER 3       */
   . .   .   END;
   . .   END;
                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │ NS_IPL_INIT FROM THE SSCP                 │
                    └──────────────────────────────────────────┘
                                                                                */
   . . WHEN(NS_RQ_CODE = NS_IPL_INIT)
   . .   DO;
   . .   . IF FSM_PU_T2_LOAD ¬= RESET THEN                   /* PAGE 11-118     */
   . .   .   RECEIVE_CHECK_SENSE = X'0809';                  /* MODE INCONSISTENCY */
   . .   . ELSE
   . .   .   RECEIVE_CHECK_SENSE = UPM_IPL_RQ_VALIDITY_CHECK; /* PAGE 11-115     */
   . .   . IF RECEIVE_CHECK_SENSE ¬= 0 THEN
   . .   .   CALL CHANGE_MU_TO_NEG_RSP(RECEIVE_CHECK_SENSE); /* APPENDIX B      */
   . .   . ELSE
   . .   .   CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);           /* APPENDIX B      */
   . .   . CALL FSM_PU_T2_LOAD;                             /* PAGE 11-118     */
   . .   . SEND MU TO SNS.SEND;                             /* CHAPTER 6       */
   . .   END;
                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │ NS_IPL_TEXT OR NS_IPL_FINAL FROM THE SSCP │
                    └──────────────────────────────────────────┘
                                                                                */
   . . WHEN(NS_RQ_CODE = (NS_IPL_TEXT | NS_IPL_FINAL))
   . .   DO;
   . .   . IF FSM_PU_T2_LOAD ¬= NS_TEXT THEN                 /* PAGE 11-118     */
   . .   .   RECEIVE_CHECK_SENSE = X'0809';                  /* MODE INCONSISTENCY */
   . .   . ELSE
   . .   .   RECEIVE_CHECK_SENSE = UPM_IPL_RQ_VALIDITY_CHECK; /* PAGE 11-115     */
   . .   . IF RECEIVE_CHECK_SENSE ¬= 0 THEN
   . .   .   CALL CHANGE_MU_TO_NEG_RSP(RECEIVE_CHECK_SENSE); /* APPENDIX B      */
   . .   . ELSE
   . .   .   CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);           /* APPENDIX B      */
   . .   . CALL FSM_PU_T2_LOAD;                             /* PAGE 11-118     */
   . .   . IF RQ_CODE = NS_IPL_FINAL THEN
   . .   .   CALL FSM_PU_ACT_RES;                           /* PAGE 11-54      */
   . .   . SEND MU TO SNS.SEND;                             /* CHAPTER 6       */
   . .   END;
                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │ NS_IPL_ABORT FROM THE SSCP                │
                    └──────────────────────────────────────────┘
                                                                                */
   . . WHEN(NS_RQ_CODE = NS_IPL_ABORT)
   . .   DO;
   . .   . IF FSM_PU_T2_LOAD = RESET THEN                    /* PAGE 11-118     */
   . .   .   DO;
   . .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'0809'); /* APPENDIX B, MODE INCONSISTENCY */
   . .   .   . SEND MU TO SNS.SEND;                          /* CHAPTER 6       */
   . .   .   END;
   . .   . ELSE
   . .   .   DO;
   . .   .   . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);          /* APPENDIX B      */
   . .   .   . CALL FSM_PU_T2_LOAD;                          /* PAGE 11-118     */
   . .   .   . SEND MU TO SNS.SEND;                          /* CHAPTER 6       */
   . .   .   END;
   . .   END;
   . END;
   END;
END PU_T2_LOAD_PROC;
```

```
ADJ_PU_LOAD_PROC: PROCEDURE;

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS ROUTINE PERFORMS  A PU-PU LOAD OPERATION FOR  AN ADJACENT PU_T2 │
 │                 NODE.  THE SSCP DIRECTS THE SUBAREA PU TO PERFORM THE LOAD OPERATION │
 │                 BY SENDING INITPROC.  THE SUBAREA PU SENDS PROCSTAT TO THE SSCP UPON │
 │                 COMPLETION OF THE LOAD MODULE TRANSFER.                      │
 │                                                                              │
 │      INPUT:     INITPROC    FROM  THE   SSCP,   RSP(NC_IPL_INIT   |  NC_IPL_TEXT   | │
 │                 NC_IPL_FINAL | NC_IPL_ABORT) FROM THE PU_T2 NODE              │
 │                                                                              │
 │      OUTPUT:    NC_IPL_INIT, NC_IPL_TEXT, NC_IPL_FINAL, NC_IPL_ABORT TO THE PU_T2 │
 │                                                                              │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
 │                 NS.CS_RCV                       PAGE 11-34                   │
 │                 PU.SVC_MGR.NS.RCV               PAGE 11-28                   │
 │                                                                              │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
 │                 ADJ_PU_IPL_ABORT                PAGE 11-105                  │
 │                 FSM_ADJ_PU_LOAD                 PAGE 11-124                  │
 │                 LOAD_CHECKS                     PAGE 11-104                  │
 │                 SEND_NC_MU_TO_BF_FOR_PU_T2      PAGE 11-106                  │
 │                 UPM_BUILD_TEXT_OR_FINAL         PAGE 11-113                  │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RECEIVE_CHECK_SENSE BIT(32);

  IF NCB.PU_TYPE ¬= (T4 | T5) THEN
    IF RRI = RQ THEN
      DO;
      . CALL CHANGE_MU_TO_NEG_RSP(X'1003'); /* APPENDIX B, FUNCTION NOT SUPPORTED      */
      . SEND MU TO SNS.SEND;                /* CHAPTER 6                               */
      END;
    ELSE
      DISCARD MU;
  ELSE
    DO;
    . IF RRI = RSP THEN
    .   NRCB_PTR = LOCATE_NODE_RESOURCE(LSCB.EA);              /* APPENDIX B          */
    . ELSE
    .   NRCB_PTR = LOCATE_NODE_RESOURCE(NSC_RQ.TARGET_ADDRESS); /* APPENDIX B        */
    . CP_INDIRECT_PTR = FIRST_ENTRY(NRCB.CP_INDIRECT_LIST);    /* APPENDIX B          */
    . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
    .
    . SELECT ANYORDER;
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │  INITPROC FROM THE SSCP                  │
                  └──────────────────────────────────────────┘
                                                                                */
    . . WHEN(RQ_CODE = INITPROC)
    . .   DO;
    . .   . RECEIVE_CHECK_SENSE = LOAD_CHECKS;                 /* PAGE 11-104         */
    . .   . IF RECEIVE_CHECK_SENSE ¬= 0 THEN
    . .   .   DO;
    . .   .   . CALL CHANGE_MU_TO_NEG_RSP(RECEIVE_CHECK_SENSE); /* APPENDIX B          */
    . .   .   . SEND MU TO SNS.SEND;                           /* CHAPTER 6           */
    . .   .   END;
    . .   . ELSE
    . .   .   DO;
    . .   .   . FIND LSCB IN LSCB_LIST WHERE(LSCB.EA = NRCB.ELEMENT_ADDRESS);
    . .   .   . CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);           /* APPENDIX B          */
    . .   .   . SEND MU TO SNS.SEND;                           /* CHAPTER 6           */
    . .   .   . MU_PTR = UPM_CREATE_RQ('NC_IPL_INIT');         /* APPENDIX B          */
    . .   .   . CALL FSM_ADJ_PU_LOAD;                          /* PAGE 11-124         */
    . .   .   . CALL SEND_NC_MU_TO_BF_FOR_PU_T2;               /* PAGE 11-106         */
    . .   .   END;
    . .   END;
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │  RESPONSE TO NC_IPL_INIT FROM THE PU_T2  │
                  └──────────────────────────────────────────┘
                                                                                */
    . . WHEN(RQ_CODE = NC_IPL_INIT & RRI = RSP)
    . .   DO;
    . .   . IF RTI = NEG THEN
    . .   .   CALL ADJ_PU_IPL_ABORT(SNC);  /* SENSE CODE OF -RSP FROM PU_T2          */
    . .   . ELSE
    . .   .   IF FSM_ADJ_PU_LOAD ¬= TEXT THEN                  /* PAGE 11-124         */
    . .   .     CALL ADJ_PU_IPL_ABORT(X'0809'); /* PAGE 11-105, MODE INCONSISTENCY    */
    . .   .   ELSE
    . .   .     DO;
    . .   .     . MU_PTR = UPM_BUILD_TEXT_OR_FINAL;            /* PAGE 11-113         */
    . .   .     . CALL FSM_ADJ_PU_LOAD;                        /* PAGE 11-124         */
    . .   .     . CALL SEND_NC_MU_TO_BF_FOR_PU_T2;             /* PAGE 11-106         */
    . .   .     .
    . .   .     END;
    . .   END;
```

```
                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │ RESPONSE TO NC_IPL_TEXT FROM THE PU_T2   │
                    └──────────────────────────────────────────┘
                                                                                */
    . . WHEN(RQ_CODE = NC_IPL_TEXT & RRI = BSP)
    . .   DO;
    . .   . IF RTI = NEG THEN
    . .   .   CALL ADJ_PU_IPL_ABORT(SNC);  /* SENSE CODE OF -PSP FROM PU_T2          */
    . .   . ELSE
    . .   .   IF FSM_ADJ_PU_LOAD ¬= TEXT THEN
    . .   .     CALL ADJ_PU_IPL_ABORT(X'0809'); /* MODE INCONSISTENCY, PAGE 11-105   */
    . .   .   ELSE
    . .   .     DO;
    . .   .     . MU_PTR = UPM_BUILD_TEXT_OR_FINAL;          /* PAGE 11-113          */
    . .   .     . CALL FSM_ADJ_PU_LOAD;                      /* PAGE 11-124          */
    . .   .     . CALL SEND_NC_MU_TO_BF_FOR_PU_T2;           /* PAGE 11-106          */
    . .   .     END;
    . .   END;
                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │ RESPONSE TO NC_IPL_FINAL FROM THE PU_T2  │
                    └──────────────────────────────────────────┘
                                                                                */
    . . WHEN(RQ_CODE = NC_IPL_FINAL & RRI = BSP)
    . .   DO;
    . .   . IF RTI = NEG THEN
    . .   .   CALL ADJ_PU_IPL_ABORT(SNC);  /* SENSE CODE OF -PSP FROM PU_T2          */
    . .   . ELSE
    . .   .   IF FSM_ADJ_PU_LOAD ¬= FINAL THEN               /* PAGE 11-124          */
    . .   .     CALL ADJ_PU_IPL_ABORT(X'0809'); /* MODE INCONSISTENCY, PAGE 11-105   */
    . .   .   ELSE
    . .   .     DO;
    . .   .     . SCB_PTR = CPCB.CP_SCB_ID;
    . .   .     . MU_PTR = UPM_CREATE_RQ('PROCSTAT'); /* IPL SUCCESSFUL, APPENDIX B   */
    . .   .     . PROCSTAT_RQ.PROCEDURE_STATUS = IPL_SUCCESSFUL;
    . .   .     . CALL FSM_ADJ_PU_LOAD;                      /* PAGE 11-124          */
    . .   .     . SEND MU TO SNS.SEND;                       /* CHAPTER 6            */
    . .   .     END;
    . .   END;
                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │ RESPONSE TO NC_IPL_ABORT FROM THE PU_T2  │
                    └──────────────────────────────────────────┘
                                                                                */
    . . WHEN(RQ_CODE = NC_IPL_ABORT & RRI = BSP)
    . .   DO;
    . .   . IF RTI = NEG THEN
    . .   .   CALL UPM_LOG('NEGATIVE NC_IPL_ABORT');          /* APPENDIX B           */
    . .   . DISCARD MU;
    . .   END;
    . END;
    END;
END ADJ_PU_LOAD_PROC;
```

```
LOAD_CHECKS: PROCEDURE RETURNS(BIT(32));
                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  PERFORMS CERTAIN VALIDITY  CHECKS BEFORE A PU-PU  LOAD OPERATION CAN │
  │                 TAKE PLACE.                                                  │
  │                                                                              │
  │      INPUT:     CURRENT MU IS INITPROC                                       │
  │                                                                              │
  │      OUTPUT:    RETURNS A SENSE CODE OF 0 IF NO ERRORS ARE DETECTED AND THE LOAD CAN │
  │                 BE PERFORMED; OTHERWISE, THE APPROPRIATE SENSE CODE.         │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 ADJ_PU_LOAD_PROC                 PAGE 11-102                 │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 ALS_SEC_SUBTREE_CHECK            PAGE 11-97                  │
  │                 FSM_ADJ_PU_LOAD                  PAGE 11-124                 │
  │                 UPM_CHECK_MODULE_ID              PAGE 11-114                 │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RETURN_VALUE BIT(32) INIT(0);
  DCL P PTR;

  FIND NRCB IN NRCB_LIST WHERE(NRCB.RESOURCE_CATEGORY = BF.PU &
       NRCB.ELEMENT_ADDRESS = INITPROC_RQ.TARGET_ADDRESS);
  IF NRCB_PTR = NULL | NRCB.RESOURCE_TYPE ¬= T2 THEN
    RETURN_VALUE = X'0849';                 /* INVALID REQUESTED RESOURCE       */
  ELSE
    DO;
    . FIND P->SCB IN SCB_LIST
    .      WHERE(P->SCB.PARTNER_SA = SCB.PARTNER_SA &
    .      P->SCB.PARTNER_EA = SCB.PARTNER_EA &
    .      P->SCB.THIS_SA = NCB.NODE_SUBAREA_ADDRESS &
    .      P->SCB.THIS_EA = NRCB.ELEMENT_ADDRESS);
    . IF P = NULL THEN
    .   RETURN_VALUE = X'8005';                                 /* NO SESSION   */
    . ELSE
    .   DO;
    .   . IF FSM_ADJ_PU_LOAD ¬= RESET THEN                     /* PAGE 11-124  */
    .   .   RETURN_VALUE = X'0809';                            /* MODE INCONSISTENCY */
    .   . ELSE
    .   .   DO;
    .   .   . IF UPM_CHECK_MODULE_ID = NG THEN                 /* PAGE 11-114  */
    .   .   .   RETURN_VALUE = X'084B';    /* REQUESTED RESOURCE NOT AVAILABLE */
    .   .   . ELSE
    .   .   .   IF ALS_SEC_SUBTREE_CHECK(INITPROC_RQ.TARGET_ADDRESS) = NG THEN
    .   .   .                                                  /* PAGE 11-97   */
    .   .   .     RETURN_VALUE = X'0809';                      /* MODE INCONSISTENCY */
    .   .   END;
    .   END;
    END;
  RETURN(RETURN_VALUE);
END LOAD_CHECKS;
```

```
ADJ_PU_IPL_ABORT: PROCEDURE(SENSE);
                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE  IS INVOKED WHEN THE  SUBAREA PU CANNOT  COMPLETE THE │
   │                 LOAD OPERATION TO A PU_T2 NODE.  THIS CAN HAPPEN BECAUSE OF AN ERROR │
   │                 AT THE SUBAREA PU, OR BECAUSE  THE PU_T2 RESPONDED NEGATIVELY TO ONE │
   │                 OF THE NC_IPL COMMANDS.  THIS PROCEDURE RESETS THE ADJ_PU_LOAD FSM.  │
   │                                                                          │
   │      INPUT:     SENSE  CODE  FOR  RESPONSE;  MU_PTR  POINTS  TO  -RSP(NC_IPL_INIT  | │
   │                 NC_IPL_TEXT  |  NC_IPL_FINAL);  NRCB_PTR AND  CPCB_PTR  ARE  SET  IN │
   │                 CALLING PROCEDURE                                        │
   │                                                                          │
   │      OUTPUT:    NC_IPL_ABORT, INCLUDING  THE APPROPRIATE  SENSE DATA  TO THE  PU_T2,  │
   │                 PROCSTAT, INCLUDING  PROCEDURE STATUS SET  TO IPL  PROCEDURE FAILURE │
   │                 AND THE APPROPRIATE SENSE CODE TO THE SSCP               │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 ADJ_PU_LOAD_PROC                    PAGE 11-102          │
   │                                                                          │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
   │                 FSM_ADJ_PU_LOAD                     PAGE 11-124          │
   │                 SEND_NC_MU_TO_BF_FOR_PU_T2          PAGE 11-106          │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL SAVE_MU_PTR PTR;
  DCL SENSE BIT(32);

  SAVE_MU_PTR = MU_PTR;                    /* POINTS TO THE RESPONSE FROM THE PU_T2      */
                                           /* OR TO THE FAILING REQUEST FROM THE ADJ PU  */
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │  SEND NC_IPL_ABORT TO PU_T2 NODE         │
                  └──────────────────────────────────────────┘
                                                                                */
  MU_PTR = UPM_CREATE_RQ('NC_IPL_ABORT');                  /* APPENDIX B        */
  SNC = SENSE;
  CALL SEND_NC_MU_TO_BF_FOR_PU_T2;                         /* PAGE 11-106       */
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │  SEND  PROCSTAT(PROCEDURE  FAILURE)  TO  SSCP;  │
                  │  RESET FSM                               │
                  └──────────────────────────────────────────┘
                                                                                */
  SCB_PTR = CPCB.CP_SCB_ID;
  MU_PTR = UPM_CREATE_RQ('PROCSTAT');                      /* APPENDIX B        */
  IF SENSE = 0 THEN                        /* -RSP FROM PU_T2, SET SENSE TO SNC OF -RSP  */
    SENSE = SAVE_MU_PTR->SNC;
  PROCSTAT_RQ.PROCEDURE_STATUS = PROCEDURE_FAILURE;
  PROCSTAT_RQ.FAILING_NC_RQ_CODE = SAVE_MU_PTR->RQ_CODE;
  PROCSTAT_RQ.SENSE_DATA = SENSE;
  CALL FSM_ADJ_PU_LOAD;                                    /* PAGE 11-124       */
  SEND MU TO SNS.SEND;                                     /* CHAPTER 6         */
END ADJ_PU_IPL_ABORT;
```

```
SEND_NC_MU_TO_BF_FOR_PU_T2: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE FILLS  IN THE CANONICAL TH  AND THE RH FIELDS  FOR NC  │
   │                 MESSAGES SENT TO BOUNDARY FUNCTION FOR A PU_T2.      │
   │                                                                      │
   │      INPUT:     A NC MESSAGE  UNIT AND A POINTER  TO THE NRCB FOR  THE ADJACENT LINK  │
   │                 STATION THAT THE PU_T2 IS ATTACHED.                  │
   │                                                                      │
   │      OUTPUT:    THE CANONICAL TH AND RH IS ADDED TO THE MESSAGE.     │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 ADJ_PU_IPL_ABORT                  PAGE 11-105        │
   │                 ADJ_PU_LOAD_PROC                  PAGE 11-102        │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                                        /*
                    ┌──────────────────────────────────┐
                    │            TH FIELDS             │
                    └──────────────────────────────────┘
                                                                        */
  OAFPRIME = X'FF';                                  /* THE PU_T4|5      */
  DAFPRIME = X'00';                                  /* THE PU_T2        */
  EFI = EF;                                          /* DEFINED TO BE EXPEDITED */
                                                                        /*
                    ┌──────────────────────────────────┐
                    │            RH FIELDS             │
                    └──────────────────────────────────┘
                                                                        */
  RRI = RQ;                                          /* ONLY REQUESTS SENT */
  RU_CTGY = NC;                                      /* ONLY NC_IPL REQUESTS */
  FI = B'1';                                         /* FORMATTED        */
  SDI =  ¬SD;                                        /* NO SENSE DATA    */
  BCI = BC;                                          /* BEGIN CHAIN      */
  ECI = EC;                                          /* END CHAIN        */
  DR1I = B'1';                                       /*  RQD             */
  DR2I = B'0';
  ERI = B'0';
  QRI =  ¬QR;                                        /* RSP NOT QUEUED   */
  PI =  ¬PAC;                                        /* NO PACING REQUESTED */
  BBI =  ¬BB;                                        /* NOT BEGIN BRACKETS */
  EBI =  ¬EB;                                        /* NOT END BRACKETS */
  CDI =  ¬CD;                                        /* NO CHANGE DIRECTION */
  CSI = CODE0;                                       /* NO ALTERNATE CODE */
  EDI =  ¬ED;                                        /* NOT ENCIPHERED   */
  PDI =  ¬PD;                                        /* DATA NOT PADDED  */

  SEND MU TO BF.PC;                                  /* CHAPTER B        */

  RETURN;
END SEND_NC_MU_TO_BF_FOR_PU_T2;
```

```
NS.MS_PROC: PROCEDURE;

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  ROUTES MAINTENANCE SERVICES REQUESTS THAT ARE ADDRESSED TO THE PU OR │
 │                 THAT ORIGINATE  IN THE PU.   ALL RESPONSES ARE  DISCARDED.  VERIFIES │
 │                 THAT A  HALF-SESSION EXISTS FOR REQUEST  FROM LINK_MGR AND  THAT THE │
 │                 TARGET ADDRESS IS GOOD FOR REQUEST FROM A CONTROL POINT.     │
 │                                                                              │
 │      INPUT:     MAINTENANCE  SERVICES   REQUESTS  AND   RESPONSES  FROM   A  SSCP-PU │
 │                 HALF-SESSION OR  FROM UPM_CNMS TO BE  SENT TO A CONTROL  POINT; RU'S │
 │                 FROM LINK_MGR TO BE FORWARDED TO A CONTROL POINT             │
 │                                                                              │
 │      OUTPUT:    REQUESTS AND RESPONSES TO SNS.SEND,  UPM_CNMS, OR DLC; -RSP(0801) TO │
 │                 A CONTROL POINT HALF-SESSION                                 │
 │                                                                              │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
 │                 PU.SVC_MGR.NS.RCV                PAGE 11-28                  │
 │                                                                              │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
 │                 NS.EXECTEST_PROC                 PAGE 11-108                 │
 │                 NS.MAINT_INFO_PROC               PAGE 11-111                 │
 │                 NS.SETCV_PROC                    PAGE 11-64                  │
 │                 NS.TESTMODE_PROC                 PAGE 11-109                 │
 │                 NS.TRACE_PROC                    PAGE 11-110                 │
 │                 UPM_DISPSTOR                     PAGE 11-114                 │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF DISPATCHED_BY(DLC*) THEN
    DO;
    . NRCB_PTR = LOCATE_NODE_RESOURCE(LSCB.EA);                  /* APPENDIX B      */
    . CP_INDIRECT_PTR = FIRST_ENTRY(NRCB.CP_INDIRECT_LIST);
    . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;
    . SCB_PTR = CPCB.CP_SCB_ID;
    .
    . IF SCB_PTR = NULL THEN
    .   DO;
    .   . DISCARD MU;
    .   . RETURN;
    .   END;
    END;
  ELSE
    IF RRI = RQ THEN
      DO;
      . NRCB_PTR = LOCATE_NODE_RESOURCE(NSC_RQ.TARGET_ADDRESS);  /* APPENDIX B      */
      .                                 /* WHEN THE TARGET ADDRESS IS ZERO THIS WILL
      .                                    RETURN THE ADDRESS OF THE PU RESOURCE ENTRY */
      . IF NRCB_PTR = NULL THEN
      .   DO;
      .   . CALL CHANGE_MU_TO_NEG_RSP(X'0801');  /* APPENDIX B, RESOURCE NOT AVAILABLE */
      .   . SEND MU TO SNS.SEND;                                 /* CHAPTER 6       */
      .   . RETURN;
      .   END;
      END;
    ELSE
      IF DISPATCHED_BY(SNS.RCV) THEN                             /* CHAPTER 6       */
        DO;
        . DISCARD MU;
        . RETURN;
        END;

  SELECT ANYORDER;
  .
  . WHEN(NS_RQ_CODE = ROUTE_TEST)
  .   SEND MU TO PU.SVC_MGR.PC_ROUTE_MGR.RCV;                    /* CHAPTER 12      */
  .
  . WHEN(NS_RQ_CODE = (EXECTEST | RECTD))
  .   CALL NS.EXECTEST_PROC;                                     /* PAGE 11-108     */
  .
  . WHEN(NS_RQ_CODE = (TESTMODE | RECTR | REQTEST))
  .   CALL NS.TESTMODE_PROC;                                     /* PAGE 11-109     */
  .
  . WHEN(NS_RQ_CODE = (ACTTRACE | DACTTRACE | RECTRD))
  .   CALL NS.TRACE_PROC;                                        /* PAGE 11-110     */
  .
  . WHEN(NS_RQ_CODE = (DISPSTOR | RECSTOR))
  .   CALL UPM_DISPSTOR;                                         /* PAGE 11-114     */
  .
  . WHEN(NS_RQ_CODE = (REQMS | RECMS | RECFMS))
  .   CALL NS.MAINT_INFO_PROC;                                   /* PAGE 11-111     */
  .
  . WHEN(NS_RQ_CODE = SETCV)
  .   CALL NS.SETCV_PROC;                                        /* PAGE 11-64      */
  .
  END;
  RETURN;
END NS.MS_PROC;
```

```
NS.EXECTEST_PROC: PROCEDURE;

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  COORDINATE EXECTEST FUNCTIONS WITH OTHER LINK-LEVEL PROCEDURES.
 │
 │      INPUT:     EXECTEST FROM A CONTROL POINT, RECTD FROM DLC
 │
 │      OUTPUT:    EXECTEST TO THE LINK RESOURCE FSM; RESET SIGNAL TO THE LINK RESOURCE
 │                 FSM; EXECTEST RU TO  DLC; -RSP(EXECTEST,0801|0818|1003) TO SNS.SEND;
 │                 RECTD RU'S TO SNS.SEND
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 NS.DLC_RCV                          PAGE 11-76
 │                 NS.MS_PROC                          PAGE 11-107
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 FSM_LINK_ACT_RES                    PAGE 11-119
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL LINK_EA BIT(16);

  LINK_EA = NRCB.ELEMENT_ADDRESS;
  SELECT ANYORDER;
  .
  . WHEN(INPUT('LINK_TEST_COMPLETED'))
  .   DO;
  .   . CALL FSM_LINK_ACT_RES;                              /* PAGE 11-119        */
  .   . CALL DELETE_ALL_CP_ENTRIES(LINK_EA);                /* APPENDIX B         */
  .   END;
  .
  . WHEN(INPUT(RQ) & NS_RQ_CODE = RECTD)
  .   SEND MU TO SNS.SEND;                                  /* CHAPTER 6          */
  .
  . WHEN(INPUT(RQ) & NS_RQ_CODE = EXECTEST)
  .   IF NCB.PU_TYPE = (1 | 2) THEN
  .     DO;
  .     . CALL CHANGE_MU_TO_NEG_RSP(X'1003');  /* APPENDIX B, FUNCTION NOT SUPPORTED */
  .     . SEND MU TO SNS.SEND;                              /* CHAPTER 6          */
  .     END;
  .   ELSE
  .     DO;
  .     . IF RESOURCE_TOTAL_SHARE_CNT(LINK_EA) > 0 THEN     /* APPENDIX B         */
  .     .   DO;
  .     .   . CALL CHANGE_MU_TO_NEG_RSP(X'0801');  /* APPENDIX B, RESOURCE NOT AVAILABLE */
  .     .   . SEND MU TO SNS.SEND;                          /* CHAPTER 6          */
  .     .   END;
  .     .
  .     . ELSE
  .     .   IF FSM_LINK_ACT_RES ¬= RESET THEN               /* PAGE 11-119        */
  .     .     DO;
  .     .     . CALL CHANGE_MU_TO_NEG_RSP(X'0818');
  .     .     .                       /* APPENDIX B, LINK PROCEDURE IN PROGRESS   */
  .     .     . SEND MU TO SNS.SEND;                        /* CHAPTER 6          */
  .     .     END;
  .     .
  .     .   ELSE
  .     .     DO;
  .     .     . CALL FSM_LINK_ACT_RES;                      /* PAGE 11-119        */
  .     .     . CALL ADD_CP_ENTRY(LINK_EA,SCB_PTR);         /* APPENDIX B         */
  .     .     . FIND LSCB IN LSCB_LIST WHERE(LSCB.EA = LINK_EA);
  .     .     . SEND MU TO PU.SVC_MGR.LINK_MGR;
  .     .     . MU_PTR = UPM_CREATE_RSP('EXECTEST');        /* APPENDIX B         */
  .     .     . SEND MU TO SNS.SEND;                        /* CHAPTER 6          */
  .     .     END;
  .     END;
  END;

  RETURN;
END NS.EXECTEST_PROC;
```

```
NS.TESTMODE_PROC: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  COORDINATE TESTMODE FUNCTIONS  WITH OTHER LINK LEVEL  PROCEDURES AND │
  │                 INTERFACE UPM_REQTEST WITH AN SSCP.                          │
  │                                                                              │
  │      INPUT:     TESTMODE FROM A CONTROL POINT,  REQTEST FROM UPM_REQTEST, RECTR FROM │
  │                 DLC                                                          │
  │                                                                              │
  │      OUTPUT:    TESTMODE TO THE TEST RESOURCE FSM; RESET SIGNAL TO THE TEST RESOURCE │
  │                 FSM; TESTMODE RU TO  DLC; -RSP(TESTMODE,0801|0809|0815|0817|1003) TO │
  │                 SNS.SEND; RECTR AND REQTEST RU'S TO SNS.SEND                 │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 NS.DLC_RCV                          PAGE 11-76               │
  │                 NS.MS_PROC                          PAGE 11-107              │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 ALS_SEC_SUBTREE_CHECK               PAGE 11-97               │
  │                 FSM_ALS_CONTACT_DISCONTACT_RES      PAGE 11-122              │
  │                 FSM_ALS_TEST_RES                    PAGE 11-124              │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL ALS_EA BIT(16);
  DCL LINK_EA BIT(16);

  ALS_EA = NRCB.ELEMENT_ADDRESS;
  LINK_EA = NRCB.ASSOCIATED_RESOURCE;

  SELECT ANYORDER;
  .
  . WHEN(INPUT('TEST_COMPLETED'))                                   /* FROM DLC          */
  .   DO;
  .   . CALL DELETE_ALL_CP_ENTRIES(ALS_EA);                         /* APPENDIX B        */
  .   . CALL FSM_ALS_TEST_RES('RESET');                             /* PAGE 11-120       */
  .   END;
  .
  . WHEN(INPUT(RQ) & NS_RQ_CODE = RECTR)                            /* FROM DLC          */
  .   SEND MU TO SNS.SEND;                                          /* CHAPTER 6         */
  .
  . WHEN(INPUT(RQ) & NS_RQ_CODE = REQTEST)                          /* FROM UPM_REQTEST  */
  .   SEND MU TO SNS.SEND;                                          /* CHAPTER 6         */
  .
  . WHEN(INPUT(RQ) & NS_RQ_CODE = TESTMODE)                         /* FROM DLC          */
  .   IF NCB.PU_TYPE = (1 | 2) |
  .        TESTMODE_RQ.TARGET_ID(2:3) ¬= B'01' THEN
  .     DO;
  .     . CALL CHANGE_MU_TO_NEG_RSP(X'1003');  /* APPENDIX B, FUNCTION NOT SUPPORTED */
  .     . SEND MU TO SNS.SEND;                                      /* CHAPTER 6         */
  .     END;
  .
  .   ELSE
  .     DO;
  .     . IF FSM_ALS_CONTACT_DISCONTACT_RES = PEND_RESET |          /* PAGE 11-122       */
  .     .      FSM_ALS_CONTACT_DISCONTACT_RES = RESET_IN_PROGRESS THEN
  .     .   CALL ENQUEUE_RU_FOR_RESOURCE(ALS_EA);                   /* APPENDIX B        */
  .     . ELSE
  .     .   IF FIND_CP_ENTRY(LINK_EA,SCB_PTR) = NG THEN             /* APPENDIX B        */
  .     .     DO;
  .     .     . CALL CHANGE_MU_TO_NEG_RSP(X'0817');  /* APPENDIX B, LINK INACTIVE */
  .     .     . SEND MU TO SNS.SEND;                                /* CHAPTER 6         */
  .     .     END;
  .     .
  .     .   ELSE
  .     .     IF FIND_CP_ENTRY(ALS_EA,SCB_PTR) = NG |               /* APPENDIX B        */
  .     .          (FSM_ALS_TEST_RES = RESET &                      /* PAGE 11-124       */
  .     .          ALS_SEC_SUBTREE_CHECK(ALS_EA) = NG) THEN         /* PAGE 11-97        */
  .     .       DO;
  .     .       . CALL CHANGE_MU_TO_NEG_RSP(X'0801');  /* APPENDIX B, RESOURCE NOT AVAILABLE */
  .     .       . SEND MU TO SNS.SEND;                              /* CHAPTER 6         */
  .     .       END;
  .     .
  .     .     ELSE
  .     .       IF FIND_CP_ENTRY(ALS_EA,SCB_PTR) = OK &             /* APPENDIX B        */
  .     .            FSM_ALS_TEST_RES = TEST_IN_PROGRESS THEN       /* PAGE 11-124       */
  .     .         DO;
  .     .         . CALL CHANGE_MU_TO_NEG_RSP(X'0815');  /* APPENDIX B, FUNCTION ACTIVE */
  .     .         . SEND MU TO SNS.SEND;                            /* CHAPTER 6         */
  .     .         END;
  .     .
  .     .       ELSE
  .     .         DO;
  .     .         . CALL ADD_CP_ENTRY(ALS_EA,SCB_PTR);              /* APPENDIX B        */
  .     .         . CALL FSM_ALS_TEST_RES;                          /* PAGE 11-124       */
  .     .         . FIND LSCB IN LSCB_LIST WHERE(LSCB.EA = ALS_EA);
  .     .         . SEND MU TO PU.SVC_MGR.LINK_MGR;
  .     .         . MU_PTR = UPM_CREATE_RSP('TESTMODE');            /* APPENDIX B        */
  .     .         . SEND MU TO SNS.SEND;                            /* CHAPTER 6         */
  .     .         END;
  .     END;
  END;
  RETURN;
END NS.TESTMODE_PROC;
```

```
NS.TRACE_PROC: PROCEDURE;

                                                                                /*
┌──────────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  COORDINATE REQUEST FOR A LINK LEVEL TRACE.                   │
│                                                                              │
│      INPUT:     ACTTRACE AND DACTTRACE FROM A CONTROL POINT; RECTRD FROM DLC  │
│                                                                              │
│      OUTPUT:    +RSP TO ACTTRACE AND DACTTRACE;                              │
│                 -RSP(ACTTRACE,08330501|08330580|0815|0817); -RSP(DACTTRACE,0816)
│                                                                              │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
│                 NS.MS_PROC                      PAGE 11-107                  │
│                                                                              │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
│                 FSM_LINK_TRACE_RES              PAGE 11-120                  │
└──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL LINK_EA BIT(16);
  DCL ALS_EA BIT(16);
  DCL P PTR;

  LINK_EA = NRCB.ELEMENT_ADDRESS;
  FIND P->NRCB IN NRCB_LIST
       WHERE(P->NRCB.ASSOCIATED_RESOURCE = LINK_EA &
       P->NRCB.RESOURCE_CATEGORY = ALS);
  ALS_EA = P->NRCB.ELEMENT_ADDRESS;
  FIND LSCB IN LSCB_LIST WHERE(LSCB.EA = ALS_EA);
  TGCB_PTR = FIND_TGCB_FOR_ALS_EA(ALS_EA);                      /* APPENDIX B      */

  SELECT ANYORDER;
  .
  . WHEN(NS_RQ_CODE = RECTRD)                                   /* FROM DLC        */
  .   DO;
  .   . IF FSM_LINK_TRACE_RES = RESET THEN                      /* PAGE 11-120     */
  .   .   DISCARD MU;
  .   . ELSE
  .   .   SEND MU TO SNS.SEND;                                  /* CHAPTER 6       */
  .   END;
  .
  . WHEN(NS_RQ_CODE = DACTTRACE)                                /* FROM SSCP       */
  .   DO;
  .   . IF FIND_CP_ENTRY(LINK_EA,SCB_PTR) = NG THEN            /* APPENDIX B      */
  .   .   DO;
  .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'0817');  /* APPENDIX B, LINK INACTIVE  */
  .   .   . SEND MU TO SNS.SEND;                                /* CHAPTER 6       */
  .   .   END;
  .   . ELSE
  .   .   IF FSM_LINK_TRACE_RES = RESET THEN                   /* PAGE 11-120     */
  .   .
  .   .     DO;
  .   .     . CALL CHANGE_MU_TO_NEG_RSP(X'0816');  /* APPENDIX B, FUNCTION INACTIVE */
  .   .     . SEND MU TO SNS.SEND;                              /* CHAPTER 6       */
  .   .     END;
  .   .   ELSE
  .   .     DO;
  .   .     . IF FSM_LINK_TRACE_RES = TG_TRACE THEN             /* PAGE 11-120     */
  .   .     .   TGCB.TG_TRACE = OFF;
  .   .     . CALL FSM_LINK_TRACE_RES;                          /* PAGE 11-120     */
  .   .     . SEND MU TO PU.SVC_MGR.LINK_MGR;
  .   .     . MU_PTR = UPM_CREATE_RSP('DACTTRACE');             /* APPENDIX B      */
  .   .     . SEND MU TO SNS.SEND;                              /* CHAPTER 6       */
  .   .     END;
  .   END;
```

```
  . WHEN(NS_RQ_CODE = ACTTRACE)                                  /* FROM SSCP              */
  .
  .   IF FIND_CP_ENTRY(LINK_EA,SCB_PTR) = NG THEN               /* APPENDIX B             */
  .     DO;
  .     . CALL CHANGE_MU_TO_NEG_RSP(X'0817');  /* APPENDIX B, LINK INACTIVE                */
  .     . SEND MU TO SNS.SEND;                                   /* CHAPTER 6              */
  .     END;
  .   ELSE
  .     IF RESOURCE_TOTAL_SHARE_CNT(LINK_EA) > 1 THEN           /* APPENDIX B             */
  .       DO;
  .       . CALL CHANGE_MU_TO_NEG_RSP(X'0809');  /* APPENDIX B, MODE INCONSISTENCY         */
  .       . SEND MU TO SNS.SEND;                                 /* CHAPTER 6              */
  .       END;
  .     ELSE
  .       IF FSM_LINK_TRACE_RES = LINE_TRACE &                 /* PAGE 11-120            */
  .            ACTTRACE_RQ.TRACE_TYPE = LINK_TRACE_WITH_TG THEN
  .         DO;
  .         . CALL CHANGE_MU_TO_NEG_RSP(X'08330580');  /* APPENDIX B, INVALID PARAMETER  */
  .         .                                                    /* LINE TRACE ACTIVE      */
  .         . SEND MU TO SNS.SEND;                               /* CHAPTER 6              */
  .         END;
  .       ELSE
  .         IF FSM_LINK_TRACE_RES ¬= RESET THEN                /* PAGE 11-120            */
  .           DO;
  .           . CALL CHANGE_MU_TO_NEG_RSP(X'0815');  /* APPENDIX B, FUNCTION ACTIVE        */
  .           . SEND MU TO SNS.SEND;                             /* CHAPTER 6              */
  .           END;
  .         ELSE
  .           IF TGCB_PTR = NULL &
  .                ACTTRACE_RQ.TRACE_TYPE = LINK_TRACE_WITH_TG THEN
  .             DO;
  .             . CALL CHANGE_MU_TO_NEG_RSP(X'0801');            /* APPENDIX B             */
  .             . SEND MU TO SNS.SEND;                           /* CHAPTER 6              */
  .             END;
  .           ELSE
  .             IF FSM_LINK_TRACE_RES = RESET &                  /* PAGE 11-120            */
  .                  ACTTRACE_RQ.TRACE_TYPE = LINK_TRACE_WITH_TG &
  .                  TGCB.TG_TRACE = ON THEN
  .               DO;
  .               . CALL CHANGE_MU_TO_NEG_RSP(X'08330501');    /* APPENDIX B             */
  .                                  /* TG TRACE ALREADY ACTIVE ON ANOTHER LINK          */
  .               . SEND MU TO SNS.SEND;                       /* CHAPTER 6              */
  .               END;
  .             ELSE
  .               DO;
  .               . CALL FSM_LINK_TRACE_RES;                   /* PAGE 11-120            */
  .               . IF ACTTRACE_RQ.TRACE_TYPE = LINK_TRACE_WITH_TG THEN /* TG TRACE OPTION */
  .               .   TGCB.TG_TRACE = ON;
  .               . SEND MU TO PU.SVC_MGR.LINK_MGR;
  .               . MU_PTR = UPM_CREATE_RSP('ACTTRACE');       /* APPENDIX B             */
  .               . SEND MU TO SNS.SEND;                       /* CHAPTER 6              */
  .               END;
  END;
  RETURN;
END NS.TRACE_PROC;
```

```
NS.MAINT_INFO_PROC: PROCEDURE;
                                                                                          /*
  ┌──────────────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE PROCESSES REQUESTS  FOR MAINTENANCE INFORMATION.  THE │
  │                 PARTICULAR HALF SESSION IS PASSED WITH  THE REQMS RU TO UPM_CNMS AND │
  │                 IS RETURNED WITH  THE RESPONSE.  THE PARTICULAR  SSCP DESTINATION IS │
  │                 SPECIFIED BY UPM_CNMS FOR ALL RU'S AND RESPONSES THAT IT GENERATES.  │
  │                                                                                      │
  │      INPUT:     REQMS FROM A CONTROL POINT, RECFMS AND RECMS RU'S FROM UPM_CNMS       │
  │                                                                                      │
  │      OUTPUT:    REQMS TO UPM_CNMS, RECFMS AND RECMS TO A CONTROL POINT               │
  │                                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                       │
  │                 NS.MS_PROC                          PAGE 11-107                      │
  └──────────────────────────────────────────────────────────────────────────────────────┘
                                                                                          */
  SELECT ANYORDER;
  .
  . WHEN(INPUT(RQ) & NS_RQ_CODE = REQMS)                         /* FROM SSCP              */
  .   SEND MU TO UPM_CNMS;                                       /* PAGE 11-114            */
  .
  . WHEN(INPUT(RQ) & NS_RQ_CODE = (RECFMS | RECMS))              /* FROM UPM_CNMS          */
  .   SEND MU TO SNS.SEND;                                       /* CHAPTER 6              */
  .
  . WHEN(INPUT(RSP) & NS_RQ_CODE = REQMS)                        /* FROM UPM_CNMS          */
  .   SEND MU TO SNS.SEND;                                       /* CHAPTER 6              */
  .
  END;
  RETURN;
END NS.MAINT_INFO_PROC;
```

```
UPM_ACTPU_CPID_CHECK: PROCEDURE RETURNS(BIT(1));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE VALIDATES THE CP ID.                      │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 NS.SC_PROC                        PAGE 11-30             │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                        /*    SEE FUNCTION    */
  RETURN(OK);
END UPM_ACTPU_CPID_CHECK;



UPM_ADDLINK: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE CHECKS  TO  SEE IF  SUFFICIENT  STORAGE AND  ELEMENT     │
   │                 ADDRESSES ARE AVAILABLE AND  THE LOCAL LINK ID IS VALID.   IF OK, IT     │
   │                 GENERATES A +RSP; IF NOT, IT GENERATES A -RSP(0806|0812).                │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 NS.ADDLINK_ADDLINKSTA_PROC        PAGE 11-62             │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                        /*    SEE FUNCTION    */
  RETURN;
END UPM_ADDLINK;



UPM_ADDLINKSTA: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE CHECKS  TO  SEE IF  SUFFICIENT  STORAGE AND  ELEMENT     │
   │                 ADDRESSES ARE AVAILABLE AND THE FID TYPE IS CORRECTLY SPECIFIED.  IF      │
   │                 OK,   IT   GENERATES   A   +RSP;    IF   NOT,   IT   GENERATES   A      │
   │                 -RSP(0806|0812|0835);                                    │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 NS.ADDLINK_ADDLINKSTA_PROC        PAGE 11-62             │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                        /*    SEE FUNCTION    */
  RETURN;
END UPM_ADDLINKSTA;



UPM_ANA_PROC: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  PROCESSES AN ANA REQUEST AND GENERATES THE RESPONSE.  THESE REQUESTS      │
   │                 ARE REQUESTED FROM  A MUTUALLY PREDETERMINED POOL  OF ADDRESSES.  IN      │
   │                 THE SHARED  CONTROL ENVIRONMENT EACH  POTENTIAL SSCP HAS  A SEPARATE     │
   │                 POOL OF ADDRESSES FOR ASSIGNMENT BY ANA.  THIS IS NOT CHECKED BY THE     │
   │                 PU.SVC_MGR; HOWEVER, IF A REQUEST IS  MADE TO ASSIGN AN ADDRESS THAT     │
   │                 IS ALREADY PREVIOUSLY ASSIGNED THE ENTIRE REQUEST IS REJECTED WITH A     │
   │                 NEGATIVE RESPONSE (0815--FUNCTION ACTIVE).                               │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 NS.CS_RCV                         PAGE 11-34             │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                        /*    NOT ARCHITECTED */
  RETURN;
END UPM_ANA_PROC;
```

```
UPM_BUILD_ERROR_XID: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE  CREATES AN XID AND  INITIALIZES IT ACCORDING  TO THE │
   │                 VALUES IN LSCB.XID_SEND.                                 │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 XID_ERR_SEND                        PAGE 11-75           │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                          /*    SEE FUNCTION    */
  RETURN;
END UPM_BUILD_ERROR_XID;



UPM_BUILD_FORMAT_2_XID: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE CREATES AN XID AND INITIALIZES ALL VALUES. │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 XID_FORMAT_2_BUILD                  PAGE 11-71           │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  CREATE MU;
  MUCB.XID = ON;
                                                          /*    SEE FUNCTION    */
  RETURN;
END UPM_BUILD_FORMAT_2_XID;



UPM_BUILD_TEXT_OR_FINAL: PROCEDURE RETURNS(PTR);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE  CREATES  AN  NC_IPL_TEXT  OR  NC_IPL_FINAL.  AN │
   │                 NC_IPL_TEXT IS CREATED  IF THERE IS MORE IPL TEXT  TO BE TRANSMITTED │
   │                 TO  THE PU_T2  NODE.  IF  ALL  OF THE  IPL  TEXT HAS  BEEN SENT,  AN │
   │                 NC_IPL_FINAL IS CREATED.                                 │
   │                                                                          │
   │      INPUT:     NONE                                                     │
   │                                                                          │
   │      OUTPUT:    MU_PTR POINTS TO AN NC_IPL_TEXT OR NC_IPL_FINAL          │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 ADJ_PU_LOAD_PROC                    PAGE 11-102          │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  CREATE MU;
                                                          /* FUNCTION AS DESCRIBED */
  RETURN(MU_PTR);
END UPM_BUILD_TEXT_OR_FINAL;



UPM_CHAN370_CHECK: PROCEDURE RETURNS(BIT(1));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE  PERFORMS  IMPLEMENTATION-DEPENDENT  CHECKS  ON  THE │
   │                 RECEIVED XID.                                            │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 XID_FORMAT_CHECK_2                  PAGE 11-69           │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                          /*    SEE FUNCTION    */
  RETURN(OK);
END UPM_CHAN370_CHECK;
```

```
UPM_CHECK_MODULE_ID: PROCEDURE RETURNS(BIT(1));

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM DETERMINES IF  THE IPL CODE REFERENCED BY THE  MODULE ID IN  │
   │                 INITPROC  CAN  BE  ACCESSED.   IF  THE   MODULE  ID  IS  ALL  SPACES  │
   │                 (X'4040...'),  THIS  UPM  DETERMINES  IF A  DEFAULT  MODULE  CAN  BE  │
   │                 ACCESSED.                                            │
   │                                                                      │
   │      INPUT:     THE CURRENT MU IS INITPROC                           │
   │                                                                      │
   │      OUTPUT:    OK, IF A LOAD MODULE CAN BE ACCESSED;                │
   │                 NG, IF A LOAD MODULE CANNOT BE ACCESSED              │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 LOAD_CHECKS                          PAGE 11-104     │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RC BIT(1);
                                                   /* FUNCTION AS DEFINED  */
  RC = OK;
  RETURN(RC);
END UPM_CHECK_MODULE_ID;



UPM_CNMS: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE  RECEIVES  REQMS AND  GENERATES  RECMS  AND  RECFMS.  │
   │                 REQUESTS AND  RESPONSES ARE  RECEIVED FROM  NS.MAINT_INFO_PROC.  ALL  │
   │                 REQUESTS AND RESPONSES ARE SENT TO PU.SVC_MGR.NS.RCV, WHICH FORWARDS  │
   │                 THEM TO NS.MAINT_INFO_PROC TO BE FORWARDED TO THE CORRECT SSCP.      │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                   /*    SEE FUNCTION    */
  RETURN;
END UPM_CNMS;



UPM_DISPSTOR: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE PROCESSES DISPSTOR REQUESTS.  IT CHECKS THE BEGINNING  │
   │                 LOCATION.   IF CORRECT,  IT SENDS  A  +RSP(DISPSTOR); OTHERWISE,  IT  │
   │                 SENDS -RSP(08350009).  IF THE REQUEST IS  VALID, ONE OR MORE RECSTOR  │
   │                 RU'S ARE SENT TO SSCP REQUESTING IT.                 │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 NS.MS_PROC                           PAGE 11-107     │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                   /*    SEE FUNCTION    */
  RETURN;
END UPM_DISPSTOR;



UPM_EXTRACT_NS_LSA_RQD: PROCEDURE RETURNS(BIT(1));

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE  LOCATES CONTROL VECTOR  X'0B' IN THE  ACTPU REQUEST.  │
   │                 IF FOUND,  THE NS_LSA_RQD  BIT IS  RETURNED.  IF  NOT FOUND,  THEN A  │
   │                 VALUE OF YES IS RETURNED INDICATING THAT NS_LSA IS REQUIRED.         │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 NS.SC_PROC                           PAGE 11-30      │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                   /*    NOT ARCHITECTED  */
  RETURN(NO);
END UPM_EXTRACT_NS_LSA_RQD;
```

```
UPM_IPL_RQ_VALIDITY_CHECK: PROCEDURE RETURNS(BIT(32));

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  THIS FUNCTION CHECKS  THE VALIDITY OF THE IPL COMMAND  RECEIVED BY A │
   │                PU_T2 NODE.  SPECIFICALLY, THIS FUNCTION RETURNS A SENSE CODE OF:    │
   │                     X'1001'  RU DATA ERROR--IF THE TEXT  PORTION OF THE NC_IPL_TEXT  │
   │                              OR THE NS_IPL_TEXT REQUEST IS IN THE WRONG FORMAT.      │
   │                     X'1005'  PARAMETER  ERROR--IF THE  WRONG LOAD  MODULE NAME  WAS  │
   │                              RECEIVED IN AN NC_IPL_INIT  OR NS_IPL_INIT REQUEST, OR  │
   │                              THE   ENTRY  POINT   LOCATION   ON  THE   NC_IPL_FINAL  │
   │                              NS_IPL_FINAL REQUEST WAS OUT OF RANGE.                  │
   │                     X'0815'  FUNCTION  ACTIVE--IF A  PU_T2  NODE  IS ALREADY  FULLY  │
   │                              LOADED OR DIDN'T REQUEST TO BE LOADED.                  │
   │                THIS FUNCTION RETURNS A SENSE CODE OF 0 IF THE IPL COMMAND IS VALID.  │
   │                                                                      │
   │     INPUT:     THE  CURRENT  MU  IS   AN  NC_IPL_INIT,  NC_IPL_TEXT,  NC_IPL_FINAL,  │
   │                NS_IPL_INIT, NS_IPL_TEXT,OR NS_IPL_FINAL REQUEST.     │
   │                                                                      │
   │     OUTPUT:    A SENSE CODE OF  0 IF THE IPL COMMAND IS  VALID; OTHERWISE, X'1001',  │
   │                X'1005', OR X'0815' AS APPROPRIATE                    │
   │                                                                      │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
   │                PU_T2_LOAD_PROC                  PAGE 11-100          │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RETURN_VALUE BIT(32);
                                                    /* FUNCTION AS DESCRIBED  */
  RETURN_VALUE = 0;
  RETURN(RETURN_VALUE);
END UPM_IPL_RQ_VALIDITY_CHECK;



UPM_PRI_SEC_ROLE: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  THIS PROCEDURE DETERMINES  IF THE LOCAL STATION CAN  ASSUME ONLY THE │
   │                PRIMARY STATION  OR ONLY  THE SECONDARY  STATION ROLE.  IF SO,  THE  │
   │                SETTING OF THE  XID_2.CONTACT_OR_LOAD_STAT, XID_2_SDLC.STA_ROLE_PRI,  │
   │                AND XID_2_SDLC.STA_ROLE_SEC  FIELDS ARE  OVERRIDDEN TO  INDICATE THE  │
   │                ONE ROLE THAT THE LOCAL STATION CAN ASSUME.            │
   │                                                                      │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
   │                FSM_XID_FORMAT_2                 PAGE 11-126          │
   │                XID_FORMAT_2_BUILD               PAGE 11-71           │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                    /*    SEE FUNCTION     */
  RETURN;
END UPM_PRI_SEC_ROLE;



UPM_REQTEST: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  THIS PROCEDURE GENERATES A REQTEST REQUEST.  IT FORWARDS THE REQUEST │
   │                TO PU.SVC_MGR.NS.RCV TO BE FORWARDED TO THE SPECIFIED SSCP.  HOW THE  │
   │                SSCP IS SELECTED IS NOT ARCHITECTED.                   │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                    /*    SEE FUNCTION     */
  RETURN;
END UPM_REQTEST;



UPM_RESTORE_SNF: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  RESTORES  SEQUENCE NUMBER  FIELD FROM  A PREVIOUS  REQUEST INTO  THE  │
   │                CURRENT RESPONSE.                                     │
   │                                                                      │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
   │                NS.LINK_RSP                      PAGE 11-79           │
   │                NS.LOAD_RSP                      PAGE 11-84           │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                    /*    NOT ARCHITECTED  */
  RETURN;
END UPM_RESTORE_SNF;
```

```
UPM_RNAA_RESOURCE_CHECK: PROCEDURE RETURNS(BIT(1));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE CHECKS  TO  SEE IF  SUFFICIENT  STORAGE AND  NETWORK  │
   │                 ADDRESSES ARE AVAILABLE TO ALLOW ALL THE REQUESTED NETWORK ADDRESSES  │
   │                 TO BE  ASSIGNED.  IF  THERE ARE  SUFFICIENT RESOURCES,  IT SETS  THE  │
   │                 RETURN CODE TO OK; IF THERE ARE NOT, IT SETS THE RETURN CODE TO NG.   │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 NS.RNAA_PROC                    PAGE 11-52               │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                        /*    SEE FUNCTION    */
  RETURN(OK);
END UPM_RNAA_RESOURCE_CHECK;



UPM_SAVE_SNF: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  SAVES SEQUENCE NUMBER FIELD FROM CURRENT  REQUEST FOR USE IN A LATER  │
   │                 RESPONSE.                                                │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 NS.DUMP_PROC                    PAGE 11-48               │
   │                 NS.LOAD_PROC                    PAGE 11-46               │
   │                 NS.RPO_PROC                     PAGE 11-50               │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                        /*    NOT ARCHITECTED */
  RETURN;
END UPM_SAVE_SNF;



UPM_SETCV_KEY1: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE PROCESSES DATE AND TIME CONTROL VECTOR.   │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 NS.SETCV_PROC                   PAGE 11-64               │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                        /*    NOT ARCHITECTED */
  RETURN;
END UPM_SETCV_KEY1;



UPM_SETCV_KEY3: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE PROCESSES SDLC SEC STATION CONTROL VECTOR. │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 NS.SETCV_PROC                   PAGE 11-64               │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                        /*    NOT ARCHITECTED */
  RETURN;
END UPM_SETCV_KEY3;



UPM_SETCV_KEY4: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE PROCESSES LU CONTROL VECTOR.              │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 NS.SETCV_PROC                   PAGE 11-64               │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                        /*    NOT ARCHITECTED */
  RETURN;
END UPM_SETCV_KEY4;
```

```
UPM_SETCV_KEY8: PROCEDURE RETURNS(BIT(16));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE PROCESSES INTENSIVE MODE CONTROL VECTOR.       │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 NS.SETCV_PROC                        PAGE 11-64              │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL ERROR_CODE BIT(16);

  ERROR_CODE = 0;                                                   /*    NOT ARCHITECTED       */

  RETURN(ERROR_CODE);
END UPM_SETCV_KEY8;



UPM_2_BYTE_NA_ASSIGN: PROCEDURE RETURNS(BIT(16));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  SELECTS AN UNUSED ELEMENT ADDRESS AND FORMS A 2 BYTE NETWORK ADDRESS
  │                 BY ADDING THE NODE SUBAREA.                                  │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 NS.BF_LU_ADD                         PAGE 11-60              │
  │                 NS.BF_PU_AND_ALS_ADD                 PAGE 11-61              │
  │                 NS.LU_ADD                            PAGE 11-62              │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL TWO_BYTE_ADDRESS BIT(16);
                                                                    /*    NOT ARCHITECTED       */

  RETURN(TWO_BYTE_ADDRESS);
END UPM_2_BYTE_NA_ASSIGN;
```

```
FSM_PU_ACT_RES: FSM_DEFINITION CONTEXT(NRCB);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO REMEMBER THE STATUS OF A PHYSICAL  UNIT WITH RESPECT TO ACTPU AND │
   │                 DACTPU COMMANDS AND THE IPL STATUS OF THE NODE IF A PU_T2.  │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 NS.LCP_RESET_PROC                  PAGE 11-33            │
   │                 NS.SC_PROC                         PAGE 11-30            │
   │                 PU_T2_LOAD_PROC                    PAGE 11-100           │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
```

STATE NAMES----->	RESET	PEND IPL	ACTIVE
INPUT STATE NUMBERS----->	01	02	03
ACTPU,IPL_REQUIRED	2	/	/
ACTPU,¬IPL_REQUIRED	3	/	/
S,+RSP(NC_IPL_FINAL)	/	3	-
S,+RSP(NS_IPL_FINAL)	/	3	-
'RESET'	/	1	1

```
END FSM_PU_ACT_RES;


FSM_PU_T2_LOAD: FSM_DEFINITION CONTEXT(NRCB);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO REMEMBER THE STATUS OF THE PU_T2 WHILE ITS NODE IS BEING LOADED │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 NS.LCP_RESET_PROC                  PAGE 11-33            │
   │                 PU_T2_LOAD_PROC                    PAGE 11-100           │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
```

STATE NAMES------>	RESET	NS TEXT	NC TEXT	ABORT
INPUT STATE NUMBERS---->	01	02	03	04
S,+RSP(NS_IPL_INIT)	2	/	/	/
S,-RSP(NS_IPL_INIT)	4	/	/	/
S,+RSP(NS_IPL_TEXT)	/	-	/	/
S,-RSP(NS_IPL_TEXT)	/	4	/	/
S,+RSP(NS_IPL_FINAL)	/	1	/	/
S,-RSP(NS_IPL_FINAL)	/	4	/	/
S,+RSP(NS_IPL_ABORT)	/	1	/	1
S,+RSP(NC_IPL_INIT)	3	/	/	/
S,-RSP(NC_IPL_INIT)	4	/	/	/
S,+RSP(NC_IPL_TEXT)	/	/	-	/
S,-RSP(NC_IPL_TEXT)	/	/	4	-
S,+RSP(NC_IPL_FINAL)	/	/	1	/
S,-RSP(NC_IPL_FINAL)	/	/	4	/
S,+RSP(NC_IPL_ABORT)	/	/	1	1
'RESET'	-	1	1	1

```
END FSM_PU_T2_LOAD;
```

```
FSM_CP_SESS_SDT: FSM_DEFINITION CONTEXT(CPCB);
```

```
/*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO  REMEMBER THE  STATUS  OF A  SESSION WITH  A  CONTROL POINT  WITH │
│                 RESPECT  TO SDT  REQUEST.   THE FSM  IS RESET  WHEN  THE SESSION  IS │
│                 INITIALIZED.  IT IS UNDEFINED AFTER THE CPCB IS DELETED.  │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 NS.REQACTLU_PROC                  PAGE 11-93             │
│                 NS.REQFNA_PROC                    PAGE 11-92             │
│                 NS.SC_PROC                        PAGE 11-30             │
│                 PU.SVC_MGR.NS.RCV                 PAGE 11-28             │
└──────────────────────────────────────────────────────────────────────────┘
*/
```

STATE NAMES----->	RESET	ACTIVE
INPUT STATE NUMBERS----->	01	02
SDT	2	-
'ACTIVE'	2	-

```
END FSM_CP_SESS_SDT;
```

```
FSM_LINK_ACT_RES: FSM_DEFINITION CONTEXT(NRCB);
```

```
/*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO REMEMBER THE STATUS OF A  LINK WITH RESPECT TO ACTLINK, DACTLINK, │
│                 AND EXECTEST REQUESTS.                                   │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 LINK_STATUS_CHECKS                PAGE 11-51             │
│                 NS.ACTLINK_PROC                   PAGE 11-36             │
│                 NS.DACTLINK_PROC                  PAGE 11-37             │
│                 NS.DELETENR_PROC                  PAGE 11-63             │
│                 NS.EXECTEST_PROC                  PAGE 11-108            │
│                 NS.LINK_RESET                     PAGE 11-94             │
│                 NS.LINK_RSP                       PAGE 11-79             │
│                 NS.SIG_RSP_PRI                    PAGE 11-86             │
│                 NS.SIG_RSP_SEC                    PAGE 11-88             │
└──────────────────────────────────────────────────────────────────────────┘
*/
```

STATE NAMES----->	RESET	PEND ACTIVE	ACTIVE	PEND RESET	TEST IN PROGRESS	RESET IN PROGRESS
INPUT STATE NUMBERS----->	01	02	03	04	05	06
ACTLINK	2	/	/	/	/	/
+RSP(ACTLINK)	/	3	/	/	/	-
-RSP(ACTLINK)	/	1	/	/	/	-
DACTLINK	/	/	4	/	/	/
+RSP(DACTLINK)	/	/	/	1	/	-
EXECTEST	5	/	/	/	/	/
'LINK_TEST_COMPLETED'	>	>	>	>	1	-
'LINK_RESET'	>	6	6	6	6	-
'LINK_RESET_COMPLETE'	>	>	>	>	>	1

```
END FSM_LINK_ACT_RES;
```

```
FSM_LINK_TRACE_RES: FSM_DEFINITION CONTEXT(NRCB);

                                                                                /*
┌──────────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO REMEMBER THE STATUS OF A LINK WITH RESPECT TO TRACE PROCEDURE.
│
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                 NS.DACTLINK_PROC                    PAGE 11-37
│                 NS.LINK_RESET                       PAGE 11-94
│                 NS.TRACE_PROC                       PAGE 11-110
└──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
```

STATE NAMES----->	RESET	LINE TRACE	TG TRACE
INPUT STATE NUMBERS----->	01	02	03
ACTTRACE,LINE_TRACE ACTTRACE,TG_TRACE	2 3	/ /	/ /
DACTTRACE	/	1	1
'RESET'	-	1	1

```
END FSM_LINK_TRACE_RES;


FSM_LINK_CONNIN_RES: FSM_DEFINITION CONTEXT(NRCB);

                                                                                /*
┌──────────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO REMEMBER THE STATUS OF A LINK ON A SWITCHED FACILITY WITH RESPECT
│                 TO ACTCONNIN AND DACTCONNIN.
│
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                 DACTLINK_RCV_CHECKS                 PAGE 11-39
│                 NS.CONN_PROC                        PAGE 11-40
│                 NS.CONN_RSP                         PAGE 11-82
│                 NS.LINK_RESET                       PAGE 11-94
│                 NS.SIG_RSP_PRI                      PAGE 11-86
│                 NS.SIG_RSP_SEC                      PAGE 11-88
└──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
```

STATE NAMES----->	RESET	PEND ACTIVE	ACTIVE	PEND RESET
INPUT STATE NUMBERS----->	01	02	03	04
ACTCONNIN +RSP(ACTCONNIN)	2 /	/ 3	/ /	/ /
DACTCONNIN +RSP(DACTCONNIN)	/ /	/ /	4 /	/ 1
'RESET'	-	1	1	1

```
END FSM_LINK_CONNIN_RES;
```

```
FSM_LINK_CONNOUT_RES: FSM_DEFINITION CONTEXT(NRCB);
                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO REMEMBER THE STATUS OF A LINK ON A SWITCHED FACILITY WITH RESPECT
│                 TO CONNOUT AND ABCONNOUT.
│
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                 DACTLINK_RCV_CHECKS                 PAGE 11-39
│                 NS.CONN_PROC                        PAGE 11-40
│                 NS.CONN_RSP                         PAGE 11-82
│                 NS.LINK_RESET                       PAGE 11-94
│                 NS.SIG_RSP_PRI                      PAGE 11-86
│                 NS.SIG_RSP_SEC                      PAGE 11-88
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----->	RESET	PEND ACTIVE	ACTIVE	PEND RESET
INPUT STATE NUMBERS----->	01	02	03	04
CONNOUT	2	/	/	/
+RSP(CONNOUT)	/	3	/	/
-RSP(CONNOUT)	/	1	/	/
ABCONNOUT	/	/	4	/
+RSP(ABCONNOUT)	/	/	/	1
-RSP(ABCONNOUT)	/	/	/	3
'CONNECT_OUT_SUCCESSFUL'	/	/	1	/
'RESET'	-	1	1	1

```
END FSM_LINK_CONNOUT_RES;


FSM_ALS_CONNECTED_RES: FSM_DEFINITION CONTEXT(NRCB);
                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO REMEMBER THE CONNECTED STATUS OF AN ADJACENT LINK STATION THAT IS
│                 ON A SWITCHED FACILITY.
│
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                 DACTLINK_RCV_CHECKS                 PAGE 11-39
│                 LINK_STATUS_CHECKS                  PAGE 11-51
│                 NS.ALS_RESET                        PAGE 11-95
│                 NS.CONN_PROC                        PAGE 11-40
│                 NS.CONN_RSP                         PAGE 11-82
│                 NS.SIG_RSP_PRI                      PAGE 11-86
│                 NS.SIG_RSP_SEC                      PAGE 11-88
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----->	RESET	ACTIVE	PEND RESET
INPUT STATE NUMBERS----->	01	02	03
'CONNECTED'	2	>	>
ABCONN	3	3	/
+RSP(ABCONN)	/	/	1
'RESET'	-	1	1

```
END FSM_ALS_CONNECTED_RES;
```

```
FSM_ALS_CONTACT_DISCONTACT_RES: FSM_DEFINITION CONTEXT(NRCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO REMEMBER THE  STATUS OF AN ADJACENT LINK STATION  WITH  │
│                RESPECT TO THE CONTACT AND DISCONTACT REQUESTS SENT TO IT. │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                ALS_SEC_SUBTREE_CHECK              PAGE 11-97             │
│                CONTACT_CONFIG                     PAGE 11-44             │
│                DACTLINK_RCV_CHECKS                PAGE 11-39             │
│                NS.ALS_PROC_RESET                  PAGE 11-96             │
│                NS.ALS_RESET                       PAGE 11-95             │
│                NS.CONTACT_PROC                    PAGE 11-42             │
│                NS.CONTACT_RSP                     PAGE 11-80             │
│                NS.DISCONTACT_PROC                 PAGE 11-45             │
│                NS.DLC_CONFIG                      PAGE 11-66             │
│                NS.DUMP_PROC                       PAGE 11-48             │
│                NS.LOAD_PROC                       PAGE 11-46             │
│                NS.RPO_PROC                        PAGE 11-50             │
│                NS.SETCV_PROC                      PAGE 11-64             │
│                NS.SIG_RSP_PRI                     PAGE 11-86             │
│                NS.SIG_RSP_SEC                     PAGE 11-88             │
│                NS.TESTMODE_PROC                   PAGE 11-109            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----->	RESET	PEND ACTIVE	ACTIVE	PEND RESET	RESET IN PROGRESS
INPUT STATE NUMBERS----->	01	02	03	04	05
CONTACT	2	/	/	/	/
DISCONTACT	/	4	4	/	/
±RSP(DISCONTACT)	/	/	/	1	-
CONTACTED(LOADED)	/	3	/	/	-
CONTACTED(LOAD_REQUIRED)	/	1	/	/	-
CONTACTED(ERROR)	/	1	/	/	-
'RESET'	-	1	1	-	-
'ALS_RESET'	-	5	5	5	/
'ALS_RESET_COMPLETE'	-	/	/	/	1

```
END FSM_ALS_CONTACT_DISCONTACT_RES;


FSM_ALS_SEC_DUMP_RES: FSM_DEFINITION CONTEXT(NRCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO REMEMBER  THE STATUS  OF A SECONDARY  ADJACENT LINK  STATION WITH │
│                RESPECT TO DUMPINIT, DUMPTEXT, AND DUMPFINAL REQUESTS SENT TO IT. │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                ALS_SEC_SUBTREE_CHECK              PAGE 11-97             │
│                ALS_SEC_SUBTREE_INTERRUPT          PAGE 11-99             │
│                CONTACT_CONFIG                     PAGE 11-44             │
│                NS.ALS_PROC_RESET                  PAGE 11-96             │
│                NS.CONTACT_PROC                    PAGE 11-42             │
│                NS.DISCONTACT_PROC                 PAGE 11-45             │
│                NS.DUMP_PROC                       PAGE 11-48             │
│                NS.LOAD_RSP                        PAGE 11-84             │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----->	RESET	PEND INDUMP INIT	INDUMP	PEND INDUMP TEXT	PEND RESET
INPUT STATE NUMBERS----->	01	02	03	04	05
DUMPINIT	2	/	/	/	/
+RSP(DUMPINIT)	/	3	/	/	/
-RSP(DUMPINIT)	/	1	/	/	/
DUMPTEXT	/	/	5	/	/
±RSP(DUMPTEXT)	/	/	/	3	/
DUMPFINAL	/	/	6	/	/
+RSP(DUMPFINAL)	/	/	/	/	1
-RSP(DUMPFINAL)	/	/	/	/	3
'RESET'	-	1	1	1	1

```
END FSM_ALS_SEC_DUMP_RES;
```

```
FSM_ALS_SEC_IPL_RES: FSM_DEFINITION CONTEXT(NRCB);
```

```
/*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO REMEMBER  THE STATUS  OF A SECONDARY  ADJACENT LINK  STATION WITH │
│                 RESPECT TO IPLINIT, IPLTEXT, AND IPLFINAL REQUESTS SENT TO IT.       │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 ALS_SEC_SUBTREE_CHECK              PAGE 11-97            │
│                 ALS_SEC_SUBTREE_INTERRUPT          PAGE 11-99            │
│                 CONTACT_CONFIG                     PAGE 11-44            │
│                 NS.ALS_PROC_RESET                  PAGE 11-96            │
│                 NS.CONTACT_PROC                    PAGE 11-42            │
│                 NS.DISCONTACT_PROC                 PAGE 11-45            │
│                 NS.LOAD_PROC                       PAGE 11-46            │
│                 NS.LOAD_RSP                        PAGE 11-84            │
└──────────────────────────────────────────────────────────────────────────┘
*/
```

STATE NAMES----->	RESET	PEND INIPL INIT	INIPL	PEND INIPL TEXT	PEND RESET
INPUT STATE NUMBERS----->	01	02	03	04	05
IPLINIT	2	/	/	/	/
+RSP(IPLINIT)	/	3	/	/	/
-RSP(IPLINIT)	/	1	/	/	/
IPLTEXT	/	/	5	/	/
±RSP(IPLTEXT)	/	/	/	3	/
IPLFINAL	/	/	6	/	/
+RSP(IPLFINAL)	/	/	/	/	1
-RSP(IPLFINAL)	/	/	/	/	3
'RESET'	-	1	1	1	1

```
END FSM_ALS_SEC_IPL_RES;
```

```
FSM_ALS_SEC_RPO_RES: FSM_DEFINITION CONTEXT(NPCB);
```

```
/*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO REMEMBER  THE STATUS  OF A SECONDARY  ADJACENT LINK  STATION WITH │
│                 RESPECT TO RPO REQUEST SENT TO IT.                       │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 ALS_SEC_SUBTREE_CHECK              PAGE 11-97            │
│                 ALS_SEC_SUBTREE_INTERRUPT          PAGE 11-99            │
│                 CONTACT_CONFIG                     PAGE 11-44            │
│                 NS.ALS_PROC_RESET                  PAGE 11-96            │
│                 NS.CONTACT_PROC                    PAGE 11-42            │
│                 NS.DISCONTACT_PROC                 PAGE 11-45            │
│                 NS.LOAD_RSP                        PAGE 11-84            │
│                 NS.RPO_PROC                        PAGE 11-50            │
└──────────────────────────────────────────────────────────────────────────┘
*/
```

STATE NAMES----->	RESET	PEND
INPUT STATE NUMBERS----->	01	02
RPO	2	/
±RSP(RPO)	/	1
'RESET'	-	1

```
END FSM_ALS_SEC_RPO_RES;
```

```
FSM_ALS_SEC_XID_RES: FSM_DEFINITION CONTEXT(NRCB);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO REMEMBER  THE STATUS  OF A SECONDARY  ADJACENT LINK  STATION WITH
   │                 RESPECT TO XID EXCHANGE.
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 ALS_SEC_SUBTREE_CHECK              PAGE 11-97
   │                 NS.ALS_PROC_RESET                  PAGE 11-96
   │                 NS.SIG_RSP_PRI                     PAGE 11-86
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
```

STATE NAMES----->	RESET	PEND
INPUT — STATE NUMBERS----->	01	02
'XID'	2	-
'XID_COMPLETED'	-	1
'RESET'	-	1

```
END FSM_ALS_SEC_XID_RES;


FSM_ADJ_PU_LOAD: FSM_DEFINITION CONTEXT(NRCB);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO REMEMBER THE STATUS OF THE SUBAREA  PU WHEN IT IS LOADING A PU_T2
   │                 NODE
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 ADJ_PU_IPL_ABORT                   PAGE 11-105
   │                 ADJ_PU_LOAD_PROC                   PAGE 11-102
   │                 LOAD_CHECKS                        PAGE 11-104
   │                 NS.ALS_RESET                       PAGE 11-95
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
```

STATE NAMES----------> / INPUT STATE NUMBERS--------->	RESET 01	TEXT 02	FINAL 03
S,NC_IPL_INIT	2	/	/
S,NC_IPL_TEXT	/	-	/
S,NC_IPL_FINAL	/	3	/
S,PROCSTAT	/	1	1
'RESET'	/	1	1

```
END FSM_ADJ_PU_LOAD;


FSM_ALS_TEST_RES: FSM_DEFINITION CONTEXT(NRCB);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO REMEMBER  THE STATUS  OF A SECONDARY  ADJACENT LINK  STATION WITH
   │                 RESPECT TO TESTMODE PROCEDURE.
   │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
   │                 ALS_SEC_SUBTREE_CHECK              PAGE 11-97
   │                 LINK_STATUS_CHECKS                 PAGE 11-51
   │                 NS.ALS_RESET                       PAGE 11-95
   │                 NS.TESTMODE_PROC                   PAGE 11-109
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
```

STATE NAMES----->	RESET	TEST IN PROGRESS
INPUT — STATE NUMBERS----->	01	02
TESTMODE	2	/
'RESET'	-	1

```
END FSM_ALS_TEST_RES;
```

```
FSM_TGN: FSM_DEFINITION CONTEXT(LSCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  THIS FINITE-STATE MACHINE  TRACKS THE MATCHING OF TGN  IN XID'S SENT │
│                AND RECEIVED.  IT  IS CALLED BY XID_FORMAT_2_RCV,  XID_ERR_SEND, AND │
│                XID_ERR_RCV.                                              │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                NS.ALS_RESET                        PAGE 11-95            │
│                XID_ERR_RCV                         PAGE 11-73            │
│                XID_ERR_SEND                        PAGE 11-75            │
│                XID_FORMAT_2_RCV                    PAGE 11-67            │
│                                                                          │
│     REFERS TO THE FOLLOWING PROCEDURE(S):                                │
│                XID_ERR_RCV                         PAGE 11-73            │
│                XID_ERR_SEND                        PAGE 11-75            │
│                XID_FORMAT_2_RCV                    PAGE 11-67            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

INPUT	STATE NAMES----> / STATE NUMBERS--> RESET 01	NOT_MATCH 02	MATCH 03
RCV.TGN=0,SEND.TGN=0	-(A1)	/	/
RCV.TGN=0,¬SEND.TGN=0	2	-	2
¬RCV.TGN=0,SEND.TGN=0	2(A2)	/	/
¬RCV.TGN=0,¬SEND.TGN=0, ¬RCV.TGN=SEND.TGN	-(A1)	1(A1)	1(A1)
¬RCV.TGN=0,¬SEND.TGN=0, RCV.TGN=SEND.TGN	3	3	-
CONTACTED(NOT_LOADED)	-	1	1
'RESET'	-	1	1

OUTPUT CODE	FUNCTION
A1	LSCB.CONTACTED_STATUS = INCOMPATIBLE_STATIONS; LSCB.XID_SEND.ERROR_STATUS = EXCHANGED_PARMS_INCOMPAT;

A2	LSCB.XID_SEND.TGN = LSCB.XID_RCV.TGN;
	FIND TGCB IN TGCB_LIST WHERE
	((TGCB.TGN = LSCB.XID_SEND.TGN) & (TGCB.ADJ_SA = LSCB.XID_SEND.SA));
	IF TGCB_PTR = NULL THEN
	DO;
	. LSCB.XID_SEND.ERROR_STATUS = NO_TG;
	. LSCB.CONTACTED_STATUS = NO_ROUTE;
	END;
	ELSE
	LSCB.TGCBPTR = TGCB_PTR;

```
END FSM_TGN;
```

```
FSM_XID_FORMAT_2: FSM_DEFINITION CONTEXT(LSCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  THIS FINITE-STATE MACHINE TRACKS THE PROCESS FOR CHOOSING PRIMARY OR │
│                SECONDARY ROLE DURING THE XID EXCHANGE.                   │
│                                                                          │
│                RESET STATE IS EXITED ONLY UPON THE RECEIPT OF A CONTACT REQUEST FOR │
│                THIS ADJACENT LINK STATION.  PEND_ACT_PRI_SEC  STATE IS ENTERED ONLY │
│                BY A  TRANSITION FROM  RESET WHEN  THE CONTACT  IS RECEIVED,  AND IS │
│                EXITED ONLY BY SENDING THE FIRST XID.                     │
│                                                                          │
│                PEND_ACT_PRI_1  AND  PEND_ACT_PRI_2  ARE STATES  THAT  INDICATE  THE │
│                STATION IN THIS NODE WILL BE  A CMD_SENDER.  PEND_ACT_PRI_1 STATE IS │
│                ENTERED  UPON  SENDING  AN  XID THAT  SPECIFIES  THIS  NODE  AS  THE │
│                CMD_SENDER, AND IS EXITED UPON  RECEIVING AN XID.  PEND_ACT_PRI_2 IS │
│                ENTERED WHEN AN XID SPECIFYING THE  ADJACENT STATION AS A RSP_SENDER │
│                IS RECEIVED, AND  IS EXITED WHEN SENDING AN XID  SPECIFYING THAT THE │
│                STATION IN  THIS NODE  IS TO  BE THE  CMD_SENDER, OR,  WHEN THE  XID │
│                EXCHANGE IS COMPLETE, BY SENDING A CONTACTED REQUEST.     │
│                                                                          │
│                PEND_ACT_SEC_1  AND  PEND_ACT_SEC_2  ARE STATES  THAT  INDICATE  THE │
│                STATION  IN  THIS NODE  WILL  BE  A RSP_SENDER.   PEND_ACT_SEC_1  IS │
│                ENTERED UPON  RECEIVING AN XID  THAT SPECIFIES THE  ADJACENT STATION │
│                WILL BE  THE CMD_SENDER;  THE STATE  IS EXITED  UPON SENDING  AN XID │
│                SPECIFYING THAT THE STATION IN THIS NODE IS TO BE THE RSP_SENDER, OR │
│                UPON   SENDING  A   CONTACTED   REQUEST   WITH  AN   ERROR   STATUS. │
│                PEND_ACT_SEND_2 IS  ENTERED UPON SENDING  AN XID THAT  SPECIFIES THE │
│                STATION IN THIS  NODE AS THE RSP_SENDER, AND IS  EXITED BY RECEIVING │
│                AN XID OR, WHEN THE XID EXCHANGE IS COMPLETE, BY SENDING A CONTACTED │
│                REQUEST.                                                  │
│                                                                          │
│                PEND_ACT_CONT IS ENTERED  WHEN THIS NODE HAS SENT  AN XID SPECIFYING │
│                ITS STATION  AS CMD_SENDER AND THEN  RECEIVES AN XID  SPECIFYING THE │
│                ADJACENT LINK STATION AS CMD_SENDER.  IT IS EXITED BY SENDING AN XID │
│                SPECIFYING THE  STATION IN  THIS NODE  AS CMD_SENDER  OR RSP_SENDER, │
│                DEPENDING ON THE OUTCOME OF THE ALGORITHM SHOWN UNDER OUTPUT CODE A2 │
│                IN THIS FSM, OR BY SENDING A CONTACTED REQUEST WITH AN ERROR STATUS. │
│                                                                          │
│                ACTIVE STATE  IS ENTERED  WHEN A  CONTACTED REQUEST  WITH STATUS  OF │
│                LOADED IS SENT;  AT THAT POINT, THE  PROTOCOL SHOWN IN THIS  FSM HAS │
│                RESULTED IN AGREEMENT  ON WHICH STATION IS TO BE  THE RSP_SENDER AND │
│                WHICH IS TO  BE THE CMD_SENDER.  IT  IS EXITED ONLY WHEN  THE FSM IS │
│                RESET.  ONCE THIS STATE HAS BEEN  ENTERED, THE ADJACENT LINK STATION │
│                MAY BE USED TO TRANSMIT DATA.                             │
│                                                                          │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                CONTACT_CONFIG                    PAGE 11-44              │
│                DACTLINK_RCV_CHECKS               PAGE 11-39              │
│                NS.ALS_RESET                      PAGE 11-95              │
│                NS.DUMP_PROC                      PAGE 11-48              │
│                NS.LOAD_PROC                      PAGE 11-46              │
│                NS.RPO_PROC                       PAGE 11-50              │
│                PU.SVC_MGR.NS.RCV                 PAGE 11-28              │
│                STATION_CONTACTED                 PAGE 11-72              │
│                SUCCESSFUL_XID_EXCHANGE           PAGE 11-72              │
│                XID_ERR_RCV                       PAGE 11-73              │
│                XID_ERR_SEND                      PAGE 11-75              │
│                XID_FORMAT_2_BUILD                PAGE 11-71              │
│                XID_FORMAT_2_RCV                  PAGE 11-67              │
│                                                                          │
│     REFERS TO THE FOLLOWING PROCEDURE(S):                                │
│                NS.INOP_PROC                      PAGE 11-90              │
│                UPM_PRI_SEC_ROLE                  PAGE 11-115             │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES------->	RESET	PEND ACT PRI SEC	PEND ACT PRI_1	PEND ACT PRI_2	PEND ACT SEC_1	PEND ACT SEC_2	PEND ACT CONT	ACTIVE
STATE NUMBERS----->	01	02	03	04	05	06	07	08
INPUT								
CONTACT	2	/	/	/	/	/	/	/
S, CMD_SENDER, ¬ERR	/	3	/	3	/	/	3	/
R, CMD_SENDER, ¬ERR	-(A1)	/	7(A2)	/	/	5	/	/
S, RSP_SENDER, ¬ERR	/	6	/	/	6	/	6	/
R, RSP_SENDER, ¬ERR	-(A1)	/	4	-	/	1(A1)	/	/
CONTACTED(LOADED_STA)	/	/	/	8	/	8	/	/
CONTACTED(NOT_LOADED)	-	/	/	1	1	1	1	/
R, ERR	-	/	1	/	/	1	/	>(A3)
'RESET'	-	1	1	1	1	1	1	1

OUTPUT CODE	FUNCTION
A1	LSCB.CONTACTED_STATUS = INCOMPATIBLE_STATIONS; LSCB.XID_SEND.ERROR_STATUS = EXCHANGED_PARMS_INCOMPAT;
A2	SELECT (LSCB.XID_RCV.STA_ROLE_SEC); . WHEN (NO) . . LSCB.XID_SEND.CONTACT_OR_LOAD_STAT = RSP_SENDER; . WHEN (YES) . . DO; . . . IF NCB.NODE_SUBAREA_ADDRESS < LSCB.XID_RCV.SA THEN . . . . LSCB.XID_SEND.CONTACT_OR_LOAD_STAT = RSP_SENDER; . . . ELSE . . . . LSCB.XID_SEND.CONTACT_OR_LOAD_STAT = CMD_SENDER; . . . CALL UPM_PRI_SEC_ROLE; /* PAGE 11-115 */ . . END; END;
A3	CALL NS.INOP_PROC(LSCB.EA); /* PAGE 11-90 */

END FSM_XID_FORMAT_2;

```
FSM_INPUT_DEFINITION:

 ABCONN                    NS_RQ_CODE=ABCONN     & RRI = RQ;
 ABCONNOUT                 NS_RQ_CODE=ABCONNOUT  & RRI = RQ;
 ACTCONNIN                 NS_RQ_CODE=ACTCONNIN  & RRI = RQ;
 ACTLINK                   NS_RQ_CODE=ACTLINK    & RRI = RQ;
 'ACTIVE'                  FSMINPUT = 'ACTIVE';
 ACTPU                     RQ_CODE=ACTPU         & RRI = RQ;
 ACTTRACE                  RQ_CODE=ACTTRACE      & RRI = RQ;
 'ALS_RESET'               FSMINPUT = 'ALS_RESET';
 'ALS_RESET_COMPLETE'      FSMINPUT = 'ALS_RESET_COMPLETE';
 CMD_SENDER                XID_2.CONTACT_OR_LOAD_STAT = CMD_SENDER;
 'CONNECT_OUT_SUCCESSFUL'  FSMINPUT = 'CONNECT_OUT_SUCCESSFUL';
 'CONNECTED'               FSMINPUT = 'CONNECTED';
 CONNOUT                   NS_RQ_CODE=CONNOUT    & RRI = RQ;
 CONTACT                   NS_RQ_CODE=CONTACT    & RRI = RQ;
 CONTACTED(LOADED)         NS_RQ_CODE=CONTACTED  & RRI = RQ  & CONTACTED_RQ.STATUS=X'01';
 CONTACTED(LOAD_REQUIRED)  NS_RQ_CODE=CONTACTED  & RRI = RQ  & CONTACTED_RQ.STATUS=X'02';
 CONTACTED(ERROR)          NS_RQ_CODE=CONTACTED  & RRI = RQ  & CONTACTED_RQ.STATUS=X'03';
 CONTACTED(NOT_LOADED)     NS_RQ_CODE=CONTACTED  & RRI = RQ  & CONTACTED_RQ.STATUS=¬X'04';
 CONTACTED(LOADED_STA)     NS_RQ_CODE=CONTACTED  & RRI = RQ  & CONTACTED_RQ.STATUS=X'04';
 DACTCONNIN                NS_RQ_CODE=DACTCONNIN & RRI = RQ;
 DACTLINK                  NS_RQ_CODE=DACTLINK   & RRI = RQ;
 DACTPU                    RQ_CODE=DACTPU        & RRI = RQ;
 DACTTRACE                 RQ_CODE=DACTTRACE     & RRI = RQ;
 DISCONTACT                NS_RQ_CODE=DISCONTACT & RRI = RQ;
 DUMPFINAL                 NS_RQ_CODE=DUMPFINAL  & RRI = RQ;
 DUMPINIT                  NS_RQ_CODE=DUMPINIT   & RRI = RQ;
 DUMPTEXT                  NS_RQ_CODE=DUMPTEXT   & RRI = RQ;
 ERR                       XID_2.ERROR_STATUS ¬= X'0';
 EXECTEST                  NS_RQ_CODE=EXECTEST   & RRI = RQ;
 IPL_REQUIRED              ACTPU_RSP.TYPE_ACTIVATION = X'3';
 IPLFINAL                  NS_RQ_CODE=IPLFINAL   & RRI = RQ;
 IPLINIT                   NS_RQ_CODE=IPLINIT    & RRI = RQ;
 IPLTEXT                   NS_RQ_CODE=IPLTEXT    & RRI = RQ;
 LINE_TRACE                ACTTRACE_RQ.TRACE_TYPE = X'01';
 'LINK_RESET'              FSMINPUT = 'LINK_RESET';
 'LINK_RESET_COMPLETE'     FSMINPUT = 'LINK_COMPLETE';
 'LINK_TEST_COMPLETED'     FSMINPUT = 'LINK_TEST_COMPLETED';
 NC_IPL_FINAL              RQ_CODE = NC_IPL_FINAL;
 NC_IPL_INIT               RQ_CODE = NC_IPL_INIT;
 NC_IPL_TEXT               RQ_CODE = NC_IPL_TEXT;
 PROCSTAT                  NS_RQ_CODE = NC_IPL_TEXT;
 R                         MUCB.DIRECTION = RECEIVE;
 RCV.TGN=0                 LSCB.XID_RCV.TGN = 0;
 RCV.TGN=SEND.TGN          LSCB.XID_RCV.TGN=LSCB.XID_SEND.TGN;
 'RESET'                   FSMINPUT = 'RESET';
 RPO                       NS_RQ_CODE=RPO        & RRI = RQ;
 +RSP(ABCONN)              NS_RQ_CODE=ABCONN     & RRI = RSP & RTI = POS;
 +RSP(ABCONNOUT)           NS_RQ_CODE=ABCONNOUT  & RRI = RSP & RTI = POS;
 -RSP(ABCONNOUT)           NS_RQ_CODE=ABCONNOUT  & RRI = RSP & RTI = NEG;
 +RSP(ACTCONNIN)           NS_RQ_CODE=ACTCONNIN  & RRI = RSP & RTI = POS;
 +RSP(ACTLINK)             NS_RQ_CODE=ACTLINK    & RRI = RSP & RTI = POS;
 -RSP(ACTLINK)             NS_RQ_CODE=ACTLINK    & RRI = RSP & RTI = NEG;
 +RSP(CONNOUT)             NS_RQ_CODE=CONNOUT    & RRI = RSP & RTI = POS;
 -RSP(CONNOUT)             NS_RQ_CODE=CONNOUT    & RRI = RSP & RTI = NEG;
 +RSP(DACTCONNIN)          NS_RQ_CODE=DACTCONNIN & RRI = RSP & RTI = POS;
 +RSP(DACTLINK)            NS_RQ_CODE=DACTLINK   & RRI = RSP & RTI = POS;
 ±RSP(DISCONTACT)          NS_RQ_CODE=DISCONTACT & RRI = RSP;
 +RSP(DUMPFINAL)           NS_RQ_CODE=DUMPFINAL  & RRI = RSP & RTI = POS;
 -RSP(DUMPFINAL)           NS_RQ_CODE=DUMPFINAL  & RRI = RSP & RTI = NEG;
 +RSP(DUMPINIT)            NS_RQ_CODE=DUMPINIT   & RRI = RSP & RTI = POS;
 -RSP(DUMPINIT)            NS_RQ_CODE=DUMPINIT   & RRI = RSP & RTI = NEG;
 ±RSP(DUMPTEXT)            NS_RQ_CODE=DUMPTEXT   & RRI = RSP;
 +RSP(IPLFINAL)            NS_RQ_CODE=IPLFINAL   & RRI = RSP & RTI = POS;
 -RSP(IPLFINAL)            NS_RQ_CODE=IPLFINAL   & RRI = RSP & RTI = NEG;
 +RSP(IPLINIT)             NS_RQ_CODE=IPLINIT    & RRI = RSP & RTI = POS;
 -RSP(IPLINIT)             NS_RQ_CODE=IPLINIT    & RRI = RSP & RTI = NEG;
 ±RSP(IPLTEXT)             NS_RQ_CODE=IPLTEXT    & RRI = RSP;
 +RSP(NC_IPL_ABORT)        RQ_CODE = NC_IPL_ABORT & RRI = RSP & RTI = POS;
 -RSP(NC_IPL_ABORT)        RQ_CODE = NC_IPL_ABORT & RRI = RSP & RTI = NEG;
 +RSP(NC_IPL_FINAL)        RQ_CODE = NC_IPL_FINAL & RRI = RSP & RTI = POS;
 -RSP(NC_IPL_FINAL)        RQ_CODE = NC_IPL_FINAL & RRI = RSP & RTI = NEG;
 +RSP(NC_IPL_INIT)         RQ_CODE = NC_IPL_INIT  & RRI = RSP & RTI = POS;
 -RSP(NC_IPL_INIT)         RQ_CODE = NC_IPL_INIT  & RRI = RSP & RTI = NEG;
 +RSP(NC_IPL_TEXT)         RQ_CODE = NC_IPL_TEXT  & RRI = RSP & RTI = POS;
 -RSP(NC_IPL_TEXT)         RQ_CODE = NC_IPL_TEXT  & RRI = RSP & RTI = NEG;
 +RSP(NS_IPL_ABORT)        NS_RQ_CODE = NS_IPL_ABORT & RRI = RSP & RTI = POS;
 -RSP(NS_IPL_ABORT)        NS_RQ_CODE = NS_IPL_ABORT & RRI = RSP & RTI = NEG;
 +RSP(NS_IPL_FINAL)        NS_RQ_CODE = NS_IPL_FINAL & RRI = RSP & RTI = POS;
 -RSP(NS_IPL_FINAL)        NS_RQ_CODE = NS_IPL_FINAL & RRI = RSP & RTI = NEG;
 +RSP(NS_IPL_INIT)         NS_RQ_CODE = NS_IPL_INIT  & RRI = RSP & RTI = POS;
 -RSP(NS_IPL_INIT)         NS_RQ_CODE = NS_IPL_INIT  & RRI = RSP & RTI = NEG;
 +RSP(NS_IPL_TEXT)         NS_RQ_CODE = NS_IPL_TEXT  & RRI = RSP & RTI = POS;
 -RSP(NS_IPL_TEXT)         NS_RQ_CODE = NS_IPL_TEXT  & RRI = RSP & RTI = NEG;
 ±RSP(RPO)                 NS_RQ_CODE=RPO        & RRI = RSP;
 RSP_SENDER                XID_2.CONTACT_OR_LOAD_STAT = RSP_SENDER;
 S                         MUCB.DIRECTION = SEND;
 SDT                       RQ_CODE = SDT & RRI = RQ & RU_CTGY = SC;
 SEND.TGN=0                LSCB.XID_SEND.TGN = 0;
 TESTMODE                  NS_RQ_CODE=TESTMODE   & RRI = RQ;
 TG_TRACE                  ACTTRACE_RQ.TRACE_TYPE = X'81';
 'XID'                     FSMINPUT = 'XID';
 'XID_COMPLETED'           FSMINPUT = 'XID_COMPLETED';

END FSM_INPUT_DEFINITION;
```

PU SERVICES MANAGER, PATH CONTROL ROUTE MANAGER

This chapter describes the path control route manager component of the PU.SVC_MGR, PU.SVC_MGR.PC_ROUTE_MGR (or PC_ROUTE_MGR for short), consisting of the receive router, explicit route (ER) manager, and virtual route (VR) manager.

Explicit and virtual routes provide logical connections on which PIUs travel within and between the subarea nodes of a network. The explicit route manager (ER_MGR) and virtual route manager (VR_MGR) activate, deactivate, and test explicit and virtual routes, respectively. The receive router (RCV) directs PIUs and signals to the proper PU.SVC_MGR.PC_ROUTE_MGR component. Figure 12-1 shows the flow of control between PU.SVC_MGR.PC_ROUTE_MGR and other SNA components. Figures 12-5 (page 12-15) and 12-12 (page 12-78) show the PIUs and signals sent to and from the PU.SVC_MGR.PC_ROUTE_MGR components.

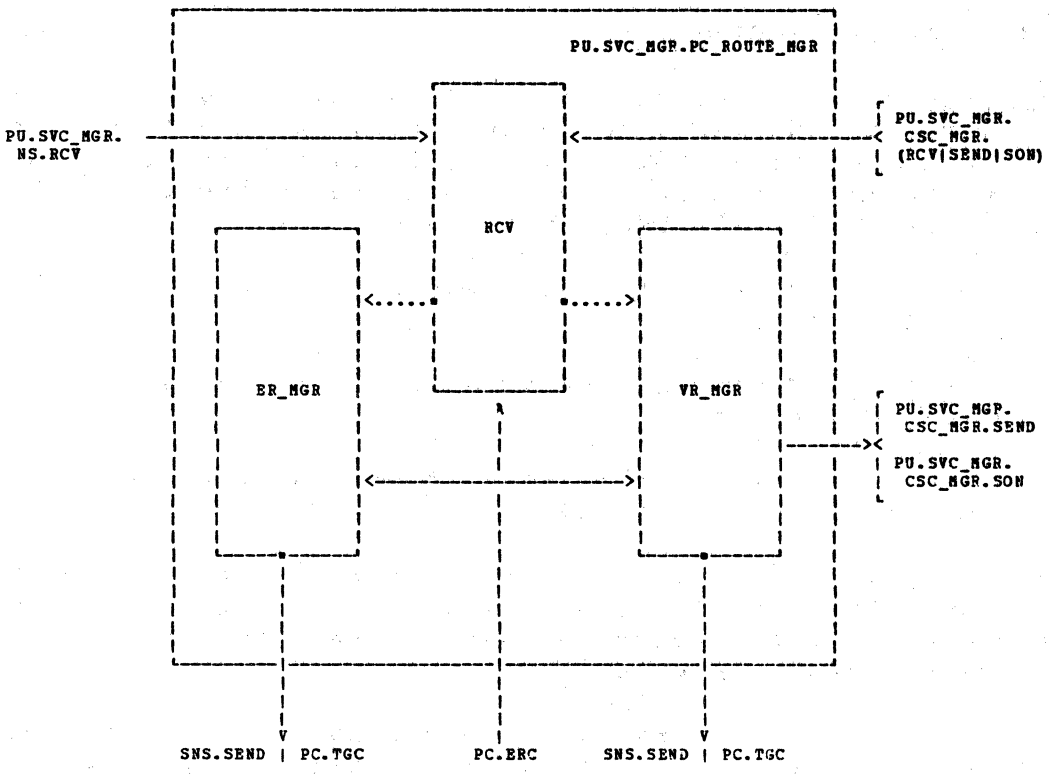


Figure 12-1. Structure of PU.SVC_MGR.PC_ROUTE_MGR

EXPLICIT ROUTES AND VIRTUAL ROUTES

Explicit routes provide pure routing capabilities from one subarea node to another (not necessarily adjacent) within the network. An explicit route (ER) is a bidirectional logical connection between subareas and can be identified by the quadruple (SA1, SA2, ERN, RERN), where:

- SA1 is the address of the subarea at one end of the explicit route.
- SA2 is the address of the subarea at the other end of the explicit route.
- ERN is the explicit route number carried in PIUs transmitted from SA1 to SA2.
- RERN is the explicit route number carried in PIUs transmitted from SA2 to SA1 (and is referred to as the reverse explicit route number). The RERN may be a value different from the ERN.

An ER includes a set of transmission groups (TGs) connecting subarea nodes. The set of transmission groups traversed by the route specified by the ERN is the same as the set for the route specified by the RERN.

A maximum of 16 explicit route numbers exists for each direction of flow between any two subarea nodes.

A virtual route (VR) logically connects the subareas in which the NAUs participating in a session reside, building global flow-control properties onto the routing capabilities provided by explicit routes. A virtual route is a bidirectional logical connection between subareas that can be identified by the quadruple (SA1, SA2, VRN, TPF), where:

- SA1 is the address of the subarea at one end of the virtual route.
- SA2 is the address of the subarea at the other end of the virtual route.
- VRN is the virtual route number carried in PIUs transmitted between SA1 and SA2.
- TPF is the transmission priority assigned to the virtual route.

PIUs are transmitted over the explicit route (or set of TGs) underlying a virtual route according to transmission priority. Up to 16 virtual route numbers and 3 transmission priorities (low, medium, and high) can be used between any two subarea nodes, yielding up to 48 virtual routes.

The path control (PC) component of SNA uses explicit routing (i.e., routing over a fixed set of TGs for a given ER) to transport PIUs through a network. Even though two subarea addresses (SA1 and SA2) and two explicit route numbers (ERN and RERN) are needed to denote an explicit route, the explicit route control (ERC) component of path control uses "source-independent" routing to determine the transmission group to an adjacent subarea node over which the PIU should be sent in order to move it along the explicit route towards its destination. This method of routing ignores both the originating subarea and RERN of the explicit route on which the PIU is flowing; only the destination subarea address (DSA) and explicit route number (ERN) are used to determine the transmission group over which the PIU should be sent. The routing table giving this mapping to transmission group uses only DSA and ERN, not the entire explicit route denotation, as keys; the table would be larger if the origin subarea address (OSA) were used as a part of the key. (The TH does not even include the RERN value.)

At any node, the routing table defines exactly one transmission group to be used when routing a PIU to a DSA on an ERN, regardless of the subarea that originated the PIU. That is, if two explicit routes having the same destination and explicit route number meet at an intermediate node along their respective routes, they will continue to use the same set of transmission groups from that intermediate node until the destination node.

For example, in Figure 12-2 a network cannot have an explicit route designated by ERN3 from node A, through nodes F and M, to destination node Z, and another explicit route also designated by ERN3 originating in node B, through nodes F and N, to destination node Z. At node F, the routing tables do not differentiate between PIUs on different explicit routes designated by the same ERN. Therefore, in node F all PIUs specifying ERN3 and destined for node Z, regardless of their originating node (A or B) are sent along the same transmission group (to either M or N).

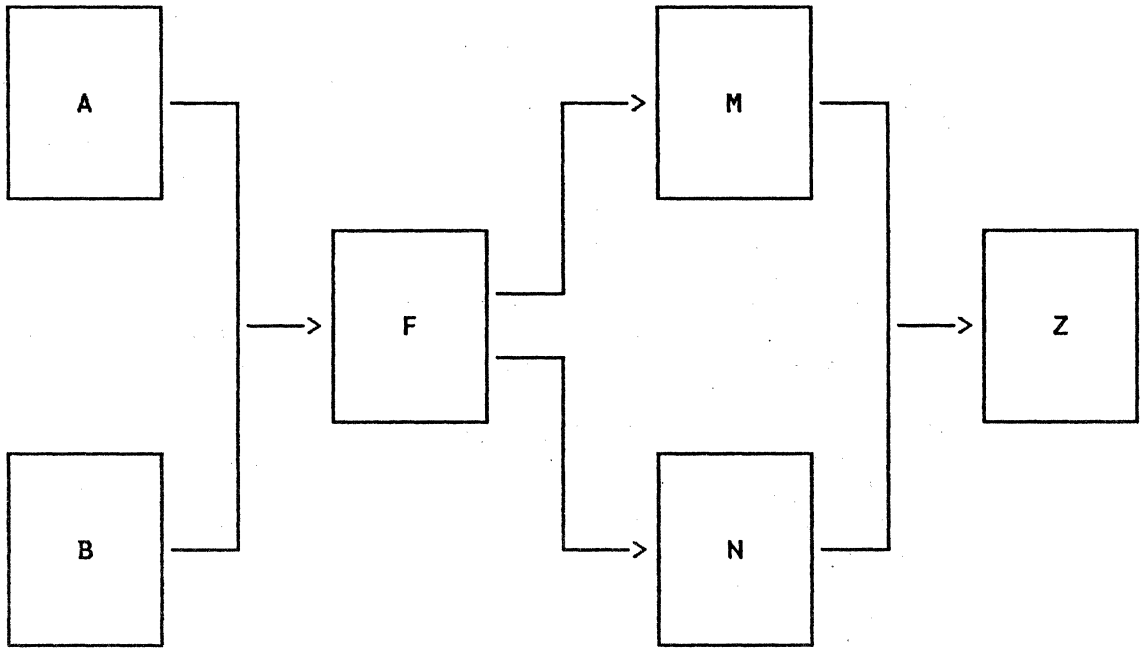


Figure 12-2. Illegal Explicit Routing Example

When a session is activated, it is assigned to a virtual route. When a half-session sends a message unit to the other half-session, path control maps the session to its assigned virtual route and then to the defined explicit route for that virtual route. Flow control (VR-level pacing), priority, and integrity (sequence numbering) facilities are associated with each virtual route.

SAMPLE OPERATION SEQUENCES

This section illustrates some ways that components of PU.SVC_MGR.PC_ROUTE_MGR interact within and across nodes. The descriptions and examples given below provide an overview of the flow of control and information, not a complete specification of the components' behavior; detailed descriptions of the functions appear in the sections pertaining to the appropriate component.

Some network control operations involving TGs, ERs, and VRs are initiated either from outer levels of SNA by a session activation request or from inner levels by a TG changing its operational status. Session activation requires the path control route manager to assign the session to an active VR, which may necessitate activating the VR and its underlying ER. A change in the state of a TG also causes the path control route manager to update the status of the explicit and virtual routes supported by that TG. All subarea nodes having ERs (and therefore VRs) that use the TG are informed of its new status: an inoperative TG forces deactivation of the supported ERs and VRs; an operative TG allows the activation of the supported ERs and VRs. The next sections detail the interactions between the VR manager, ER manager, and other components of an SNA node when processing a session activation request or a change from an operative to an inoperative TG.

Session Activation

Before NAUs can have an active session, the subarea nodes in which they reside must be connected so that message units can flow between them. This connection consists of an active virtual route, supported by an active explicit route and by a sequence of operative transmission groups. The common session control (CSC) manager (PU.SVC_MGR.CSC_MGR, Chapter 13) receives a BIND, ACTPU, ACTLU, or ACTCDRM session activation request from the (SSCP|LU).SVC_MGR. In addition to the activation request, the CSC manager receives a virtual route (VR) identifier list specifying VRs acceptable for the data traffic of this session. The CSC manager passes this VR identifier list to the VR manager and requests that the VR manager select the first VR that is active or able to be activated from the list and return its identifier. (The SSCP of the primary half-session derives this VR identifier list from the class of service (COS) name specified in the session initiation request.)

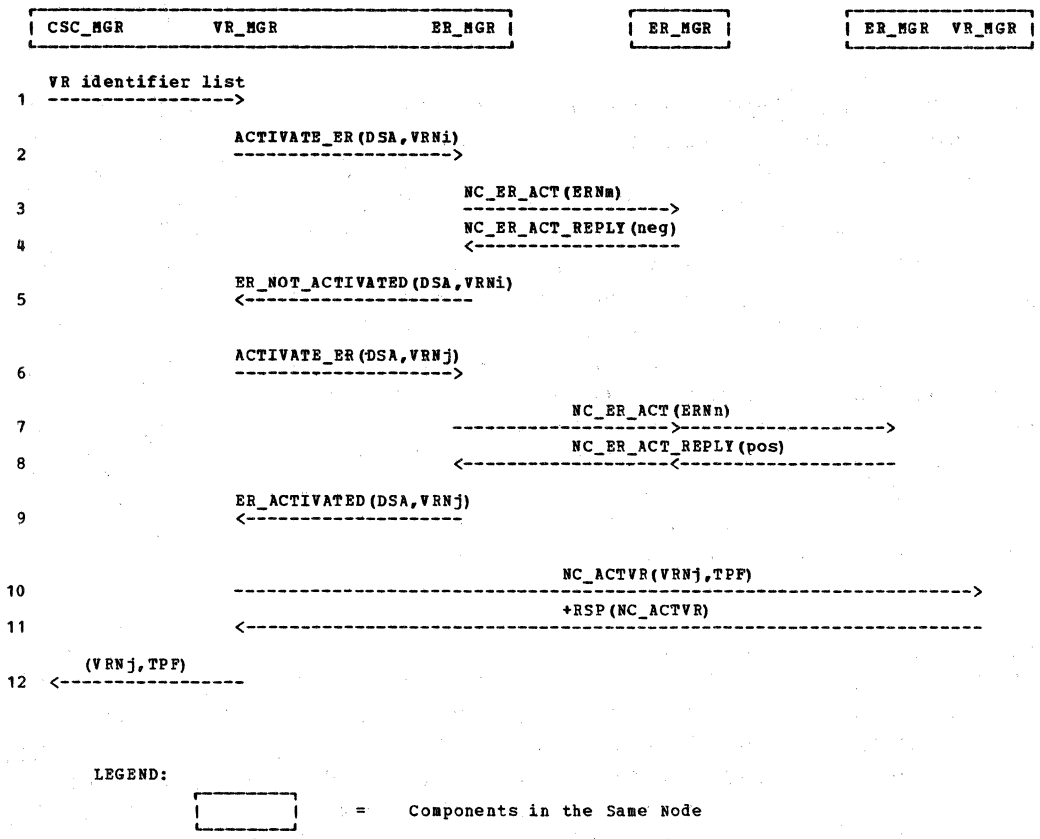
For each VR in the list, the VR manager determines if it is already active or can be made active. If the VR is already active, the VR manager identifies this VR to the CSC manager, which assigns the session to that VR. If the VR is not active, the VR manager attempts to activate the VR in the following manner. The VR manager passes an ACTIVATE_ER

signal to the ER manager (using a PARM_ACT_ER parameter list that contains the DSA and VRN). The ER manager maps the VR (specified by (DSA, VRN)) to the underlying ER (specified by (DSA, ERN)) and determines its status. If the ER is inoperative, the ER manager returns an ER_NOT_ACTIVATED signal to the VR manager; if the ER is active, the ER manager returns an ER_ACTIVATED signal to the VR manager; if the ER is operative but not active, the ER manager sends an NC_ER_ACT request into the network to activate the ER. When the NC_ER_ACT_REPLY request returns, the ER manager passes an ER_NOT_ACTIVATED or ER_ACTIVATED signal to the VR manager, depending on the status of the ER.

The VR manager receives the ER_ACTIVATED or ER_NOT_ACTIVATED signal from the ER manager. If the signal is an ER_ACTIVATED, the VR manager sends an NC_ACTVR through the network (on the ER returned by the ER manager in the ER_ACTIVATED signal) to the VR manager in the subarea containing the session partner. If a +RSP(NC_ACTVR) is returned, the VR manager returns to the CSC manager the original session activation request and an identifier of the VR to be used by the session.

If the VR manager receives the ER_NOT_ACTIVATED signal as a result of its ACTIVATE_ER to the ER manager, or if the VR cannot be activated (as indicated by a -RSP(NC_ACTVR)), the VR manager tries the next VR from the VR identifier list. If no VR from the list can be activated, the VR manager negatively responds to the CSC manager.

Several sessions may use the same VR simultaneously. When later session activations are requested, the VR manager can return a VR identifier to the CSC manager without calling the ER manager. Similarly, if an ER is already active, the ER manager can return an ER_ACTIVATED signal to the VR manager without having to send an NC_ER_ACT request into the network.



NOTE: This figure shows a sample sequence of interactions involving the path control route manager during session activation. The CSC manager sends a VR identifier list (and the session activation request) to the VR manager (1). The VR manager determines that the first VR identified by (DSA, VRN, TPF) in the VR identifier list is not already active, so the (DSA, VRN) is passed to the ER manager (2). The ER manager maps the (DSA, VRN) to a (DSA, ERN), which identifies an ER that happens not to be currently active either, so an NC_ER_ACT request is sent into the network (3). At some subarea node along the ER an error is detected, and a negative NC_ER_ACT_REPLY request is returned to the NC_ER_ACT originator (4). That ER manager recognizes that the ER cannot be activated, and passes an ER_NOT_ACTIVATED signal back to the VR manager (5). The VR manager now examines the second VR in the VR identifier list. Again the VR is not already active, so the ER manager is requested (via an ACTIVATE_ER signal) to activate the underlying ER (6). The ER manager maps the (DSA, VRN) to a (DSA, ERN), and the identified ER is also not active. Another NC_ER_ACT request is sent into the network (7), but this time it reaches its destination successfully and a positive NC_ER_ACT_REPLY is returned (8). The ER manager passes an ER_ACTIVATED signal to the VR manager indicating that the underlying ER is active (9). The VR manager sends an NC_ACTVR request (10) and receives the positive response (11) indicating that the VR is active. The VR manager then responds to the CSC manager with a VR that can be used for the session being activated (12).

Figure 12-3. PC Route Manager Activity during Session Activation

TG Inoperative

PU.SVC_MGR.PC_ROUTE_MGR activity is also required when a transmission group becomes inoperative (for whatever reason), all ERs, VRs, and sessions supported by that TG need to be deactivated. The PU.SVC_MGR.NS in the subarea nodes at each end of a TG that becomes inoperative passes a signal to the respective ER managers identifying the particular TG that has become inoperative. The ER manager sends an NC_ER_INOP request to the ER manager of all adjacent subarea nodes (except the one connected to the other end of the inoperative TG) identifying the ERs for which routing has been lost. Each of those adjacent nodes that uses one of the now inoperative ERs in turn sends an NC_ER_INOP to all of its adjacent subarea nodes. This propagation of request units stops when all nodes with explicit routes using the inoperative TG have been notified of the failure.

The originating node and each node that receives an NC_ER_INOP deactivates any ERs that use the TG that is inoperative. The list of corresponding (DSA, ERN) pairs is sent to the SSCP in an ER_INOP request. The ER manager also passes an ERINOP signal to the VR manager listing all newly inoperative ERs (using their (DSA, ERN) pairs) so that all appropriate VRs can be deactivated. The VR manager then identifies those VRs (using their (DSA, VRN) pairs) in a VR_INOP request unit to the appropriate SSCPs and in a VRINOP signal to the CSC manager.

NETWORK CONTROL RH VALUES

All network control request and response units have the following RH values:

RU Category indicator	01
Format indicator	1
Begin Chain indicator	1
End Chain indicator	1
Queued Response indicator	0
Pacing indicator	0
Begin Bracket indicator	0
End Bracket indicator	0
Change Direction indicator	0
Code Selection indicator	0
Enciphered Data indicator	0
Padded Data indicator	0

All network control request units processed by the ER manager are sent with no-response requested; all network control request units processed by the VR manager and the NC_IPL commands processed by the PU.SVC_MGR.NS (Chapter 11) are sent with definite response requested (RQD1).

NETWORK CONTROL TH VALUES

All network control requests and responses are sent using the expedited flow (the EFI bit is set to EXP in the TH).

Before a route (either explicit or virtual) can be activated, any PIUs using a previous activation of that route must be flushed from it. All route activation PIUs flow with TG Sweep set to SWEEP, so that any PIUs already on the route will be pushed ahead of the activation PIU and expelled from the route before the route is reactivated. At each node along a route, a PIU having TG Sweep set to SWEEP will force all PIUs using the same TG to be sent to the node at the other end of the TG before the PIU doing the sweeping.

The TPF setting for a route activation PIU is determined by whether it is a virtual or explicit route being activated--for a virtual route, the TPF is that of the VR being activated; for an explicit route, the TPF is L_PRTY (low). When the TG Sweep indicator in a PIU is set to SWEEP, all PIUs of equal and higher priority to be forced ahead of the sweeping PIU. Therefore, the virtual route activation request and response units will sweep both the VR being activated and all virtual routes having equal or higher priorities. For explicit route activation, virtual routes of all priorities are swept in order to support any and all virtual routes defined to use the explicit route. (See Chapter 3 for a description of how the TG Sweep indicator and TPF field are used by path control.)

NC_ER_OP and NC_ER_INOP flow with the TG Sweep set to SWEEP and TPF set to H_PRTY (high). Since both RUs have the same TPF value, they will stay in the same order relative to each other; i.e., an NC_ER_OP can never pass an NC_ER_INOP, nor vice versa. The H_PRTY TPF causes both NC_ER_OP and NC_ER_INOP to be pushed ahead of all explicit and virtual route activation RUs (requests or responses)--e.g., an NC_ER_ACT request can not move ahead of an NC_ER_INOP request. However, because of its higher TPF value, an NC_ER_INOP can overtake and pass an explicit route activation request. If the NC_ER_OP or NC_ER_INOP enters a route ahead of a route activation RU, it will stay ahead of the activation RU. If the NC_ER_OP or NC_ER_INOP passes a route activation RU, the order of RU processing at a node will be changed, but the net result will be the same.

For example, if an NC_ER_ACT is flowing on a route followed by an NC_ER_INOP and then an NC_ER_OP, the NC_ER_INOP and NC_ER_OP may pass the NC_ER_ACT, but they stay in the same order relative to each other (i.e., the NC_ER_INOP remains first). If they do not pass the NC_ER_ACT, the route may be partially activated, but the NC_ER_INOP will cause it to be deactivated--the net effect being a non-activated ER. If

both the NC_ER_INOP and NC_ER_OP pass the NC_ER_ACT, the originator of the route activation request will reject the NC_ER_ACT_REPLY, because the NC_ER_INOP received by the originator of the NC_ER_ACT will reset the control block recording the sending of the route activation being replied to.

The NC_ER_TEST and NC_ER_TEST_REPLY requests use the L_PRTY TPF so as not to delay important message units in the network.

All NC request and response units handled by the VR manager flow on the VR that the RUs are controlling and use associated VRN, ERN, and TPF values. All NS RUs handled by the ER or VR manager flow on SSCP-PU sessions. The following table specifies the type of flow, priority, and TG Sweep setting for RUs processed by this chapter.

RU	Flow		TG Sweep	TPF
NC_ACTVR	VR_MGR-->VR_MGR,	END	SWEEP	VR's
RSP(NC_ACTVR)	VR_MGR-->VR_MGR,	END	SWEEP	VR's
NC_DACTVR	VR_MGR-->VR_MGR,	END	SWEEP	VR's
RSP(NC_DACTVR)	VR_MGR-->VR_MGR,	END	SWEEP	VR's
NC_ER_TEST	ER_MGR-->ER_MGR,	SEQ	¬SWEEP	L_PRTY
NC_ER_TEST_REPLY	ER_MGR-->ER_MGR,	SEQ	¬SWEEP	L_PRTY
NC_ER_ACT	ER_MGR-->ER_MGR,	SEQ	SWEEP	L_PRTY
NC_ER_ACT_REPLY	ER_MGR-->ER_MGR,	SEQ	SWEEP	L_PRTY
NC_ER_OP	ER_MGR-->ER_MGR,	FAN	¬SWEEP	H_PRTY
NC_ER_INOP	ER_MGR-->ER_MGR,	FAN	¬SWEEP	H_PRTY
VR_INOP	VR_MGR-->SSCP,	SSCP-PU	¬SWEEP	-
ER_INOP	ER_MGR-->SSCP,	SSCP-PU	¬SWEEP	-
ROUTE_TEST	SSCP-->VR_MGR,	SSCP-PU	¬SWEEP	-
RSP(ROUTE_TEST)	VR_MGR-->SSCP,	SSCP-PU	¬SWEEP	-
ER_TESTED	ER_MGR-->SSCP,	SSCP-PU	¬SWEEP	-

LEGEND

SEQ: sequential propagation
FAN: fan-out propagation
END: route end node to route end node
SSCP-PU: SSCP-PU session
VR's: TPF value of the VR
-: TPF value of the VR supporting the SSCP-PU session
ER_MGR: ER manager
VR_MGR: VR manager

Note: Sequential and fan-out propagation flows are described in the section, "Request Flows" (page 12-22).

Figure 12-4. TH Settings for PC_ROUTE_MGR RUs

```
PU.SVC_MGR.PC_ROUTE_MGR.RCV: PROCEDURE;

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO CALL THE APPROPRIATE PROCEDURE TO PROCESS A SIGNAL, REQUEST, OR │
 │                 RESPONSE SENT TO THE PATH CONTROL ROUTE MANAGER COMPONENT OF  │
 │                 PU.SVC_MGR                                                   │
 │                                                                              │
 │      INPUT:     SIGNALS, RU'S, OR PIU'S FROM PC.ERC (CHAPTER 3), PU.SVC_MGR.NS │
 │                 (CHAPTER 11), PU.SVC_MGR.CSC_MGR (CHAPTER 13), OR THE HIGHER-LEVEL │
 │                 SCHEDULER (APPENDIX C)                                       │
 │                                                                              │
 │      OUTPUT:    SIGNALS, RU'S, OR PIU'S TO ER MANAGER OR VR MANAGER           │
 │                                                                              │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
 │                 ER_MGR                          PAGE 12-31                   │
 │                 VR_MGR                          PAGE 12-79                   │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SELECT ANYORDER;                                                              /*

                    ┌──────────────────────────────────────┐
                    │          INPUT FROM PC.ERC           │
                    └──────────────────────────────────────┘
                                                                                */
  . WHEN((INPUT(RQ) | INPUT(RSP)) & RU_CTGY = NC & RQ_CODE = (NC_ACTVR | NC_DACTVR))
  .   CALL VR_MGR;                                          /* PAGE 12-79      */
  .
  . WHEN(INPUT(RQ) & RU_CTGY = NC &
  .      RQ_CODE = (NC_ER_OP | NC_ER_INOP | NC_ER_ACT | NC_ER_ACT_REPLY |
  .      NC_ER_TEST | NC_ER_TEST_REPLY | LSA))
  .   CALL ER_MGR;                                          /* PAGE 12-31      */
  .                                                                             /*

                    ┌──────────────────────────────────────┐
                    │      INPUT FROM PU.SVC_MGR.NS        │
                    └──────────────────────────────────────┘
                                                                                */
  . WHEN(INPUT(RQ) & RU_CTGY = FMD & NSC_RQ.NS_HEADER = ROUTE_TEST_HDR)
  .   CALL VR_MGR;                                          /* PAGE 12-79      */
  .
  . WHEN(INPUT('TG_OP') | INPUT('TG_INOP_NORMAL') | INPUT('TG_INOP_ERROR'))
  .   CALL ER_MGR;                                          /* PAGE 12-31      */
  .                                                                             /*

                    ┌──────────────────────────────────────┐
                    │     INPUT FROM PU.SVC_MGR.CSC_MGR    │
                    └──────────────────────────────────────┘
                                                                                */
  . WHEN(INPUT(RQ) & RU_CTGY = SC &
  .      RQ_CODE = (ACTCDRM | ACTLU | ACTPU | BIND | DACTCDRM | DACTLU | DACTPU | UNBIND))
  .   CALL VR_MGR;                                          /* PAGE 12-79      */
  .
  .
  .                                                                             /*

                    ┌──────────────────────────────────────┐
                    │  INPUT FROM THE HIGHER-LEVEL SCHEDULER │
                    └──────────────────────────────────────┘
                                                                                */
  . WHEN(INPUT('SEND_DACTVR_F'))
  .   CALL VR_MGR;                                          /* PAGE 12-79      */
  .
  .
  .
  . OTHERWISE                                               /* INVALID INPUT   */
  .   DO;
  .   . CALL UPM_LOG('UNRECOGNIZED INPUT TO PC_ROUTE_MGR.RCV');  /* APPENDIX B   */
  .   . DISCARD MU;
  .   END;
  .
  END;

  RETURN;
END PU.SVC_MGR.PC_ROUTE_MGR.RCV;
```

EXPLICIT ROUTE MANAGER

The main responsibilities of the explicit route manager are to build, process, and propagate the requests dealing with the operational status, testing, and activation of explicit routes.

- The PU.SVC_MGR.NS signals its ER manager when a transmission group (TG) attached to the node changes its state from operative to inoperative or vice versa. The ER manager broadcasts (see the discussion of fan-out propagation in the section, "Request Flows") the change in operational status of ERs caused by the change in the TG to all affected ER managers.

- Explicit route testing procedures are initiated when an SSCP requests the virtual route manager to give the status of and to test VRs. The VR manager in turn requests the ER manager to provide the status of and perform the testing of the ERs. Results of the testing procedures are returned directly to the requesting SSCP.

- Explicit route activation is initiated to provide routing for PIUs flowing on a session. Before a session can be started, a VR must be designated to carry the message units for that session. The VR manager selects a VR to be used by the session and signals the ER manager to activate the ER that supports that VR. There is no protocol for requesting the deactivation of an ER; an ER is deactivated as the result of one of its TGs becoming inoperative.

The ER manager receives inputs (requests or signals) from PU.SVC_MGR.NS, PC.ERC, and the VR manager; it sends outputs (requests or signals) to SNS, PC.TGC, and the VR manager. Figure 12-5 shows the components that interact with the ER manager and the requests or signals that are exchanged.

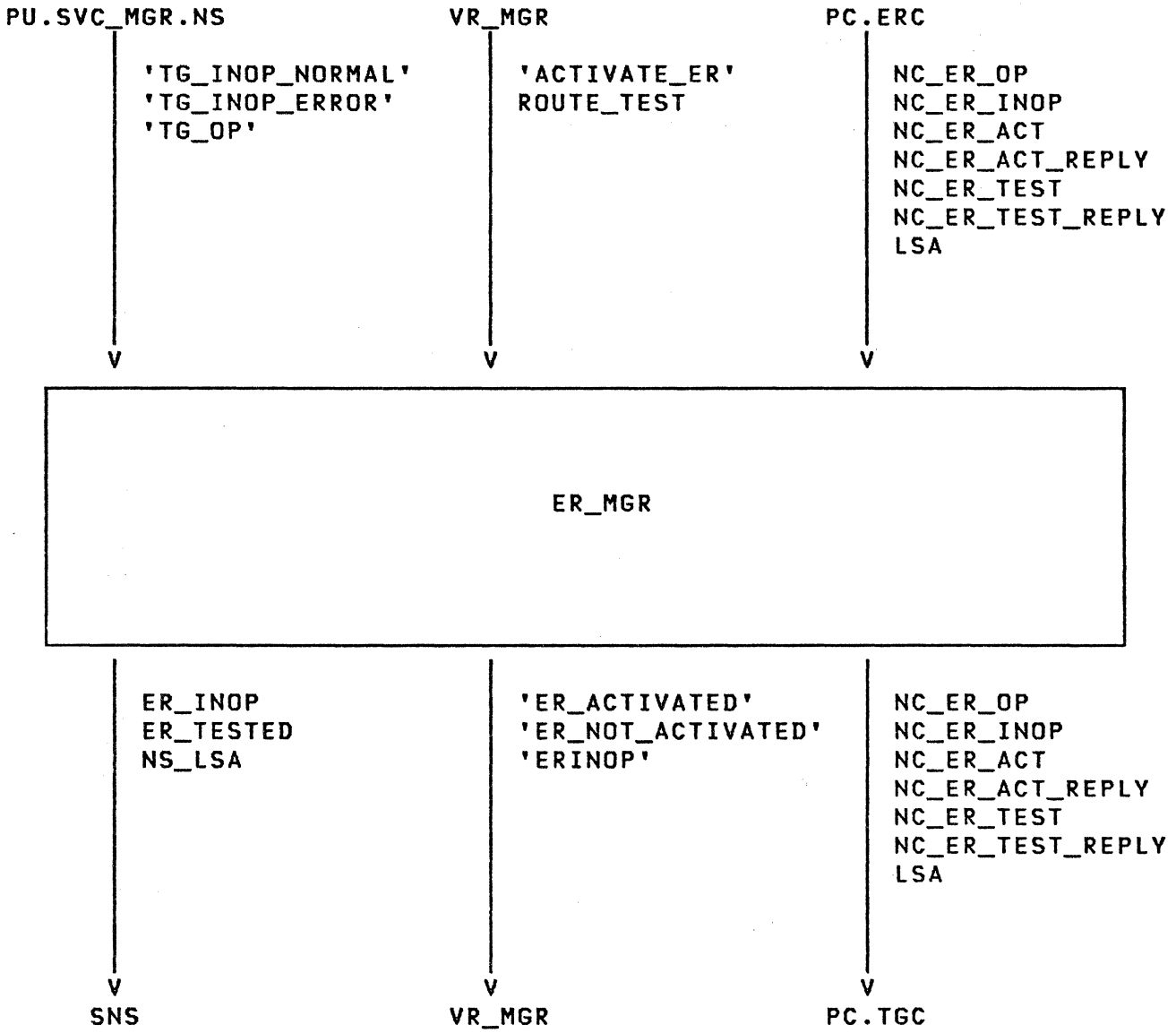


Figure 12-5. ER Manager Inputs and Outputs

The categories of requests and signals handled by the ER manager are:

- Requests relating to the operational status of ERs:
  - NC_ER_OP
  - NC_ER_INOP
  - ER_INOP
  - LSA
  - NS_LSA

  Signals relating to the operational status of TGs (and hence, ERs):
  - TG_OP
  - TG_INOP_NORMAL
  - TG_INOP_ERROR
  - ERINOP

- Requests relating to the activation of ERs:
  - NC_ER_ACT
  - NC_ER_ACT_REPLY

  Signals relating to the activation of the ERs:
  - ACTIVATE_ER
  - ER_ACTIVATED
  - ER_NOT_ACTIVATED

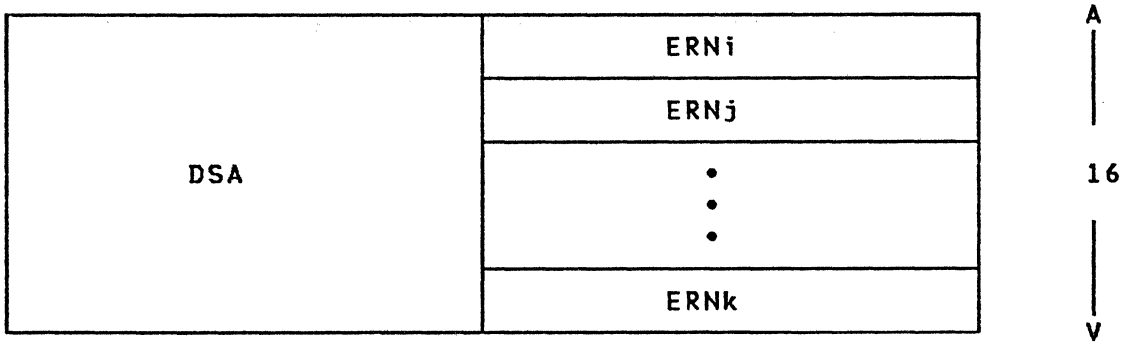
- Requests relating to the testing of ERs:
  - NC_ER_TEST
  - NC_ER_TEST_REPLY
  - ER_TESTED
  - ROUTE_TEST

- Signal relating to the definition of ERs:
  - DEFINE

DATA STRUCTURES

The major data structures used by the ER manager are the ERN_MAP_LIST, SUBAREA_ROUTING_LIST, and ERCB_LIST (see Appendix A). Their usage in relation to the ER manager functions is described below.

During session activation the VR manager may request that the ER manager activate the ER that supports the selected VR for the session. The ER manager maps the VR number (VRN) supplied by the VR manager to the ER number (ERN) defined to support it, and passes that ERN to the VR manager (so that PC.VRC (Chapter 3) can enter into the TH the proper ERN for the VR). The ERN_MAP_LIST contains the set of mappings between VRN and ERN for each destination subarea (DSA) that the node can route to. There is one entry (called an ERN_MAP) in this list for each DSA; each entry contains an array of 16 ERN values, which is indexed into by VRN value (see Figure 12-6). This list is generated during system definition; the mappings may be changed by implementation-dependent means, but only when the affected VR is neither active nor in the process of being activated.

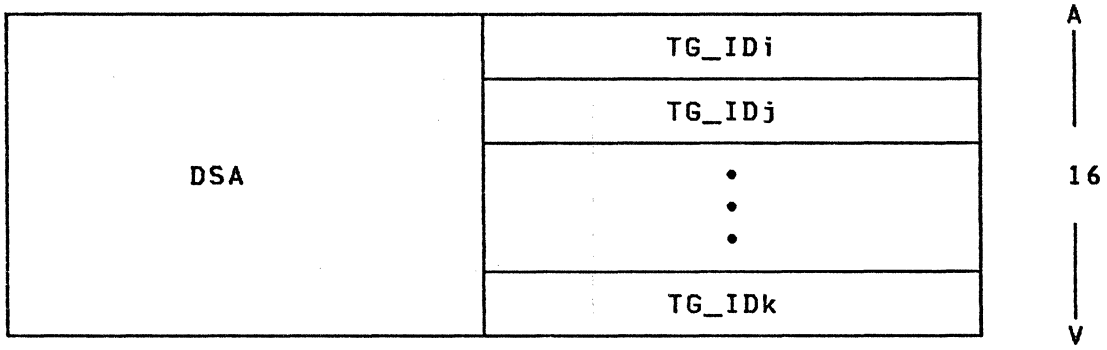


Note: ERNi is the ERN supporting VR0, ERNj is the ERN supporting VR1, and so on. The ERN values are not required to be different.

Figure 12-6. ERN_MAP_LIST Entry

When sending message units on an ER, PC.ERC (Chapter 3) uses the SUBAREA_ROUTING_LIST to map the (DSA, ERN) in the TH to a transmission group identifier (TG_ID), which specifies the TG over which the message unit should be sent. The TG_ID consists of a transmission group number (TGN) and an adjacent subarea address (ADJ_SA), which uniquely identify a particular transmission group. There is one entry (called a SUBAREA_ROUTING) in this list for every destination subarea to which this node may send PIUs. Each entry contains an array of 16 TG_IDs, indexed by ERN; each TG_ID identifies the TG over which PIUs are sent when routing via a particular (DSA, ERN). If the TG_ID is nonzero, the (DSA, ERN) is defined; if the TG_ID is zero, the (DSA, ERN) is undefined. There is no SUBAREA_ROUTING_LIST entry for the node in which the list resides, nor is there an entry for any node that is not the destination of some explicit route, even if that node is an intermediate node on some explicit route from this node.

Figure 12-7 shows the format of each SUBAREA_ROUTING_LIST entry. This list is generated during system definition, but the correspondence between (DSA, ERN) and TG_ID may be changed by implementation-dependent means or by NC_ER_OP requests (see the section, "Dynamic Routing Definition").



Notes:
1. TG_ID = (TGN, ADJ_SA)
2. TG_IDi is the TG_ID for ERN0, TG_IDj is the TG_ID for ERN1, and so on. The TG_ID values are not required to be different.

Figure 12-7. SUBAREA_ROUTING_LIST Entry

The ER manager maintains the status of and information about a specific (DSA, ERN) pair in an explicit route control block (ERCB); all ERCBs for the node are kept in the ERCB_LIST. Each ERCB contains an FSM specifying the status of the explicit route and several fields that characterize the ER when it is active (e.g., the number of transmission groups it contains). The set of states maintained in the ERCB reflects the type of information that is used by ER manager procedures, such as whether message units can be transmitted on the ER (ACTIVE state) or whether the ER is in the process of being put into the ACTIVE state (PEND_ACT).

Figure 12-8 illustrates a possible explicit route configuration where ERN0 and ERN1 are defined along the same set of transmission groups from node A to node B, and ERN2 is defined along the same set of transmission groups (but in the reverse order) from node B to node A. In node A, the ERCB for DSA B and ERN0 refers to an ER with RERN2. However, in node B, the ERCB for DSA A and ERN2 actually refers to multiple ERs because ERN2 is associated with multiple ERNs (ERN0 and ERN1) in the opposite, or reverse, direction (RERNs).

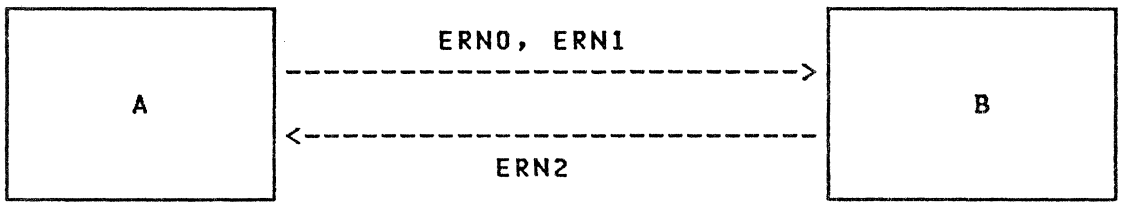


Figure 12-8. Multiple Explicit Routes Using the Same Set of TGs between Nodes

The SUBAREA_ROUTING_LIST specifies the defined mapping of (DSA, ERN) to TG_ID, but this mapping may not be given or it may be changed while the network is operating. Therefore, the ER manager maintains information about both the defined TG_ID for the (DSA, ERN) and any other TG_IDs that the ER manager is informed about via the NC_ER_OP and NC_ER_INOP requests. The information pertaining to a particular TG_ID for a (DSA, ERN) is kept in a path control block (PATHCB); all PATHCBs for a given (DSA, ERN) are contained in the PATHCB_LIST attached to the ERCB. A PATHCB is created when an NC_ER_OP is received for a (DSA, ERN) along a particular TG_ID, and is destroyed when an NC_ER_INOP is received for it. An ERCB is created when the first PATHCB for that (DSA, ERN) is created, and is destroyed when the last PATHCB for that (DSA, ERN) is destroyed. Figure 12-9 shows the format of an ERCB and its associated PATHCBs.

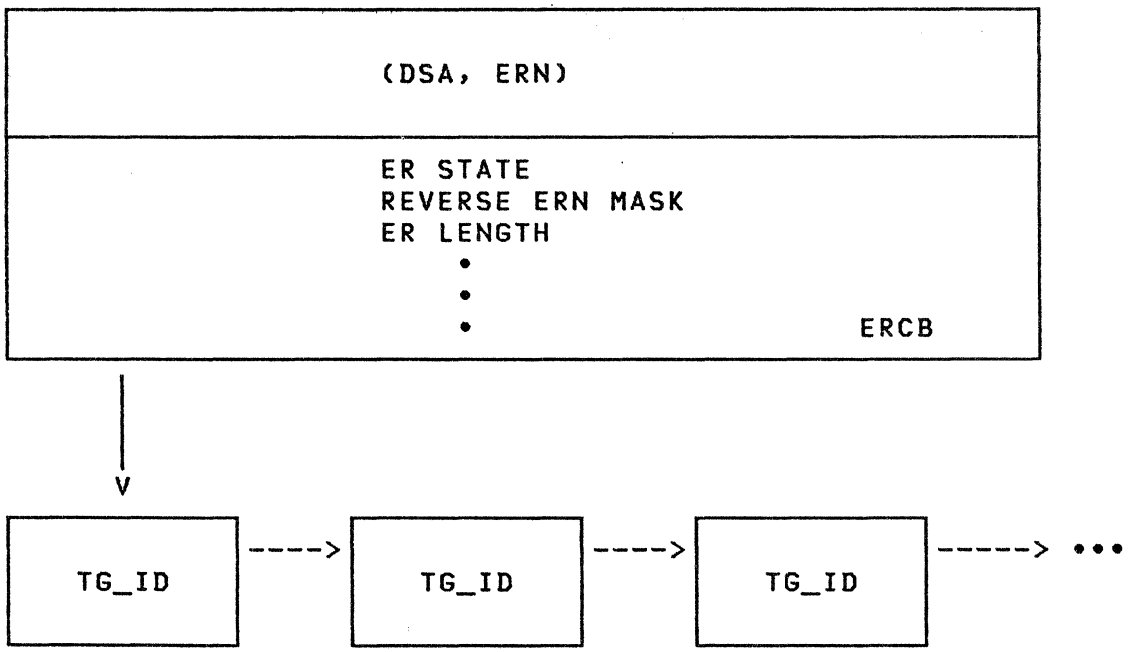


Figure 12-9. ERCB Entry and Associated PATHCB Entries

An ERCB may have multiple PATHCBs attached to it if multiple NC_ER_OP requests are received for the same (DSA, ERN), but over different TGs. For example, in Figure 12-10 the SUBAREA_ROUTING_LIST in node A could specify that PIUs destined for DSA D on ERN1 should go through node B or through node C. Regardless of what TG_ID is defined for the ER (i.e., through node B or node C), node A will receive an NC_ER_OP from node B indicating routing from B to D for ERN1, and node A will also receive an NC_ER_OP from node C indicating routing from C to D for ERN1. Node A retains information in different PATHCBs about both ways of routing PIUs on ERN1 to DSA D because the SUBAREA_ROUTING_LIST specifying the defined TG_ID for ERN1 to DSA D could be changed (one or more times) during the operation of the network.

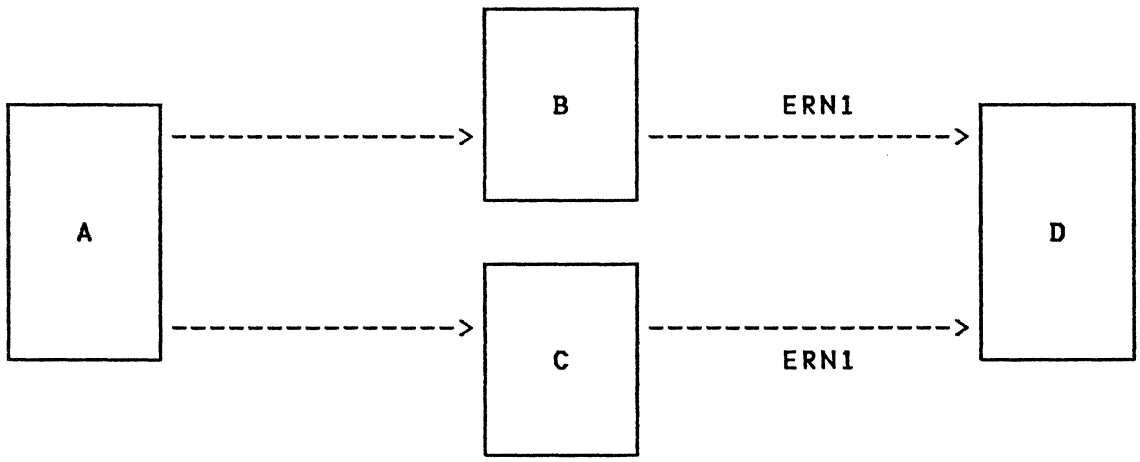


Figure 12-10. Configuration Generating Multiple PATHCBs

REQUEST FLOWS

Requests generated by the ER manager use one of two types of flows when moving through a network: sequential propagation or fan-out propagation. The requests that activate and test a single explicit route (NC_ER_ACT, NC_ER_ACT_REPLY, NC_ER_TEST, and NC_ER_TEST_REPLY) use the sequential propagation flow. These requests are processed by the ER manager in each subarea node along an explicit route; because there is no guarantee that the explicit route is properly defined and operative from one end of the explicit route to the other end, the ER manager in each subarea node along the route processes the request, including checking for various error conditions (e.g., no usable ERN in the reverse direction of the ER or the ER has a "loop" in it). The sequential propagation flow allows information about the explicit route (e.g., its length) to be gathered as the request traverses the route. The OSAF and DSAF fields in the FID4 TH for these requests do not specify the node that originated the request and the node that the request is destined for, but rather, the two adjacent subarea nodes connected by the transmission group being traversed. Accordingly, at every subarea node along the explicit route, these fields are changed to specify the adjacent subarea nodes on the TG currently being traversed. The subarea addresses of the two ends of the explicit route are contained within the request.

The requests that update the operational status of ERs (NC_ER_OP and NC_ER_INOP) use the fan-out propagation flow. These requests are sent from a subarea node to every adjacent subarea node (and over every transmission group to each adjacent subarea node) except to the node from which the request arrived. This type of flow allows all subarea nodes affected by a transmission group's change of status to be notified of the event. Other details of this propagation are discussed in "Operational Status of Explicit Routes."

Figure 12-4 (page 12-12) shows the type of request flow used by all RUs processed by the path control route manager.

PROTOCOL BOUNDARY WITH PATH CONTROL (PC)

All message units sent by the ER manager on the PU-PU flow are sent to PC.TGC. When sent to PC.TGC the message unit is completely ready to be sent out of the node, that is, all fields in the RU, RH, and TH are already filled in. TGCB_PTR is also set to indicate over which TG the message unit is to be sent. Message units entering a node destined for the ER manager are routed to PU.SVC_MGR.PC_ROUTE_MGR.RCV from PC.ERC. TGCB_PTR is set to indicate on which TG the message unit arrived.

PROTOCOL BOUNDARY WITH THE VR MANAGER

All message units for a session flow on virtual routes; the virtual route manager assigns the session to an appropriate virtual route as the session is being activated. If the desired virtual route is not already active, the VR manager signals the ER manager to activate the ER supporting that virtual route. The signal is accompanied by the virtual route number and the subarea address at the other end of the virtual route (PARTNER_SA). Using the ERN_MAP_LIST, the ER manager determines which ER supports that VR and initiates the ER activation process. (Alternative actions are described in detail in the section, "Activation of Explicit Routes.") After deciding whether or not the ER can be activated and used to support the VR, the ER manager signals the VR manager.

The ER manager exchanges information with the VR manager in two other cases: when the ER manager sends an ERINOP signal to the VR manager informing it of the change to inoperative status of an explicit route, and when the VR manager has been requested to test routes according to an ROUTE_TEST request. When testing routes, the VR manager may communicate with the ER manager in two different ways. The ER manager is called to fill in various fields in the response to ROUTE_TEST indicating the status of ERs, and also, if necessary, to send NC_ER_TEST to test an explicit route (see the section, "Explicit Route Testing").

PROTOCOL BOUNDARY WITH THE PU.SVC_MGR.NS

The PU.SVC_MGR.NS signals the ER manager of changes in the status of transmission groups. A transmission group becomes operative when a link in an inoperative transmission group is contacted using a link-level procedure, and it becomes inoperative when the last remaining link in the transmission group fails or is discontacted. The PU.SVC_MGR.NS sets TGCB_PTR to the address of the TGCB for the affected transmission group and sends the ER manager either a TG_OP, TG_INOP_NORMAL, or TG_INOP_ERROR signal. The ER manager then builds and sends the appropriate NC_ER_OP or NC_ER_INOP requests.

OPERATIONAL STATUS OF EXPLICIT ROUTES

An explicit route between two subareas is operative when all the transmission groups along the ER between the two subareas are operative. The PU.SVC_MGR.NS in each of the subarea nodes at the two ends of the transmission group becoming operative signals the ER manager as described in the preceding section. The ER manager builds an NC_ER_OP request containing a specification of the TG that became operative and a list of ERNs identifying the ERs that are currently operative. The two ER managers send the NC_ER_OP

requests to each other on the now operative transmission group. Each ER manager receiving an NC_ER_OP, including the two at the ends of the operative TG and all subsequent receivers (via fan-out propagation), updates the list of operative ERs in the request. When the NC_ER_OP is sent from an ER manager, the list of operative ERs includes only those ERs it uses that have become operative as a result of the transmission group becoming operative. The resulting NC_ER_OP is sent to adjacent subarea nodes via the fan-out propagation flow; the NC_ER_OP is not propagated if there are no entries left in its list of operative ERs.

An inoperative condition of a transmission group is handled in a similar way. Upon receiving a signal from the PU.SVC_MGR.NS indicating that a transmission group has become inoperative, the ER manager builds an NC_ER_INOP listing all the ERs that have become inoperative. The nodes at the ends of the inoperative TG and every node that receives the NC_ER_INOP use the fan-out propagation flow to communicate the new status of the transmission group to all subarea nodes that are affected. Each ER manager receiving an NC_ER_INOP, updates the list of inoperative ERs in the request to contain only those ERs it uses that have become inoperative as a result of the transmission group becoming inoperative. The NC_ER_INOP is not propagated if there are no entries left in its list of inoperative ERs.

Each subarea node that sends an NC_ER_INOP also builds and sends an ER_INOP request and an ERINOP signal. The ER_INOP is sent for each CP-PU session in which SDT has flowed to provide information on inoperative ERs. The ERINOP signal and a list of ERs is sent to the VR manager to cause it to initiate session outage notification and reset any VRs affected by the now inoperative ERs.

Subarea nodes that do not support ER-VR protocols are not sent the NC_ER_OP or NC_ER_INOP requests. During fan-out propagation, an NC_ER_OP is not sent to such a node. An NC_ER_INOP, however, is translated into an LSA and sent to that node. Such a node not supporting ER-VR protocols sends an LSA to its adjacent subarea node when it recognizes a link outage, so nodes supporting ER-VR protocols convert received LSA requests into NC_ER_INOP requests.

ACTIVATION OF EXPLICIT ROUTES

Explicit routes are activated either when the VR manager requests it as part of the VR activation process (described in this section) or when multiple NC_ER_OP requests are received for the same as yet undefined and dynamically definable (DSA, ERN) pairs (see the section, "Dynamic Routing Definition"). Unlike virtual routes, explicit routes are not deactivated when no sessions are using the route; ERs are deactivated only when a TG that is a part of

that ER becomes inoperative (see the section, "Operational Status of Explicit Routes"). To activate an explicit route, a subarea node sends an NC_ER_ACT to the other end of the explicit route and receive a positive NC_ER_ACT_REPLY in return. After receiving a positive NC_ER_ACT_REPLY, only the path control route manager component that sent the NC_ER_ACT can send an NC_ACTVR on the ER--the path control route manager component at the other end of the ER cannot send an NC_ACTVR until it activates the ER from its end by sending an NC_ER_ACT and receiving a positive NC_ER_ACT_REPLY.

One cause of explicit route activation is session activation. The CSC manager requests from the VR manager an active virtual route to be used by the session. As part of the VR activation process, the VR manager passes an ACTIVATE_ER signal to the ER manager; this signal specifies which virtual route number and DSA is being requested by the VR manager. Using the ERN_MAP_LIST, the ER manager determines which ERN supports that VRN and attempts to activate the ER if it is not already active. ER manager action is determined by the state of FSM_ERN (anchored in the ERCB), as follows.

- If the state is inoperative (RESET), the VR manager is signaled via an ER_NOT_ACTIVATED that the ER cannot be activated.

- If the state is operative (OP), an NC_ER_ACT is sent on the explicit route to activate it. No immediate response is sent to the VR manager; when the NC_ER_ACT_REPLY returns, the ER manager signals the VR manager, indicating whether or not the ER can be used to support the requested VR.

- If the state is pending activation (PEND_ACT), an NC_ER_ACT has already been sent to activate the route. When the NC_ER_ACT_REPLY returns, the ER manager signals the VR manager, indicating whether (ER_ACTIVATED) or not (ER_NOT_ACTIVATED) the ER can be used to support the requested VR.

- If the state is active (ACTIVE), the VR manager is signaled via an ER_ACTIVATED that the ER is active and can be used to support the VR.

- If the state is pending ER definition resolution (CONTEND) (see the section, "Dynamic Routing Definition"), multiple NC_ER_ACT requests have already been sent to determine which TG_ID should be used when routing message units on the ER. When the status of the ER is determined by examining the returned NC_ER_ACT_REPLY requests, the ER manager signals the VR manager, indicating whether or not the ER can be used to support the requested VR.

When sending an NC_ER_ACT to activate an ER, the ER manager uses the SUBAREA_ROUTING_LIST to determine over which TG to send the request. (See the section, "Dynamic Routing Definition," for the procedure when the (DSA, ERN) is not defined.) The request follows the sequential propagation flow along the explicit routes determined by the (DSA, ERN) (see Figure 12-8) The set of possible reverse ERNs is established as the request traverses the ERN. If at any node on the explicit route, there is no valid reverse ERN defined, an NC_ER_ACT_REPLY indicating unsuccessful activation is returned to the NC_ER_ACT originator. The ER manager at each subarea node receiving the NC_ER_ACT, increments the explicit route length (counted in terms of the number of transmission groups already traversed by the NC_ER_ACT), and compares it against the maximum specified in the MAX_ER_LENGTH field in the NC_ER_ACT. An NC_ER_ACT_REPLY indicating unsuccessful activation is sent if the ER length is exceeded. A negative NC_ER_ACT_REPLY is also returned if any subarea node along the explicit route does not have a definition (i.e., TG_ID) for the (DSA, ERN) being activated or if the transmission group specified in the definition is not currently operative.

If the NC_ER_ACT reaches its destination subarea node, the ER manager at that node builds an NC_ER_ACT_REPLY and sends it back to the node that originated the NC_ER_ACT. Thus, the originator of NC_ER_ACT receives an NC_ER_ACT_REPLY whether or not the activation is successful. (The only exception is when a TG on the ER has failed after the NC_ER_ACT has passed the TG, but before the NC_ER_ACT_REPLY has returned.) In this case, the NC_ER_ACT originator is informed of the TG failure via an NC_ER_INOP, which implies that the ER cannot be activated.) Upon receiving an NC_ER_ACT_REPLY, the ER manager sends to the VR manager an ER_ACTIVATED signal, indicating the ER can support a VR, or an ER_NOT_ACTIVATED signal, indicating that it cannot.

DYNAMIC ROUTING DEFINITION

A VR is specified in the VR identifier list derived from a COS name if its underlying ER provides the characteristics or properties required by the class of service. Whether the ER satisfies the class of service is determined by the set of TGs that are traversed by the ER. Generally, the mapping of an ER to TG in the SUBAREA_ROUTING_LIST is designed to provide certain characteristics, but if all possible TGs will provide those characteristics, then that ER to TG mapping need not be predefined. In this case, the ER manager can choose to route traffic for a (DSA, ERN) on the first TG that becomes operative, rather than waiting for a particular, predefined TG.

The ERN_SYSDEF bit, associated with each (DSA, ERN) in the SUBAREA_ROUTING_LIST, indicates whether the ER manager of a subarea node may or may not require that a (DSA, ERN) be predefined (i.e., mapped to a predetermined TG_ID). A subarea node may elect not to require (DSA, ERN) definition, but, rather, to choose a TG_ID for the (DSA, ERN) according to NC_ER_OP requests received from adjacent subarea nodes. Upon receipt of the first NC_ER_OP for a (DSA, ERN), the TG over which it arrived (as specified by the TGCB_PTR set by PC.TGC) becomes the temporarily defined value for the (DSA, ERN) (the TG_ID of the TG is placed in the SUBAREA_ROUTING_LIST for the (DSA, ERN)).

If a second NC_ER_OP arrives for the same (DSA, ERN), but on a different TG, the TG_ID field in the SUBAREA_ROUTING_LIST for that (DSA, ERN) is set to 0. To decide which TG_ID should be put into the TG_ID field, an NC_ER_ACT for the (DSA, ERN) is sent over the two transmission groups that received NC_ER_OP. (The DYNAMIC_ER_DEFN bit is set in each NC_ER_ACT to indicate that the requests are being sent to resolve the dynamic definition ambiguity.) The first positive NC_ER_ACT_REPLY that returns activates the route and determines which TG will be used when routing message units on the (DSA, ERN); the TG_ID for that TG is entered into the SUBAREA_ROUTING_LIST for the (DSA, ERN). If further NC_ER_OP requests are received for the same (DSA, ERN), but over different TGs, no further NC_ER_ACT requests are sent.

Figure 12-11 illustrates a case where the ER managers at the two ends of an ER may not appear to be synchronized. Node A allows ERN0 to be dynamically defined; both nodes B and C have ERN0 defined to node D, so node A could use ERN0 through node B or through node C. ERN1 is defined from node D to node A through node B and ERN2 is defined from node D to node A through node C. If node A receives multiple NC_ER_OP requests indicating different TGs can be used to route to node D over ERN0, node A will send multiple NC_ER_ACT requests to node D attempting to resolve this ambiguity.

The receiver of the multiple NC_ER_ACT requests (node D) could accept each one and not recognize they were related because a receiving subarea node processes the requests in terms of the sending node's reverse ERNs, not the ERN being activated by the sender. Therefore, the receiver returns a positive NC_ER_ACT_REPLY and enters the ACT_RCV state for each of its ERNs, which are the originator's RERNs. The ACT_RCV state indicates that an NC_ACTVR can be received, but cannot be sent.

The NC_ER_ACT_REPLY receiver--the NC_ER_ACT sender--accepts at most one of the NC_ER_ACT_REPLYs and enters the ACTIVE state for that (DSA, ERN). The node rejects all other NC_ER_ACT_REPLYs without telling the partner node--the partner may be in the ACT_RCV state for ERNs that will never be part of an active ER. This apparent state mismatch presents no difficulties, however, because the partner in ACT_RCV state will never receive an NC_ACTVR, and the ACT_RCV state does not allow the node to activate its own VR and therefore get into conflict with the state of the ER at the other end.

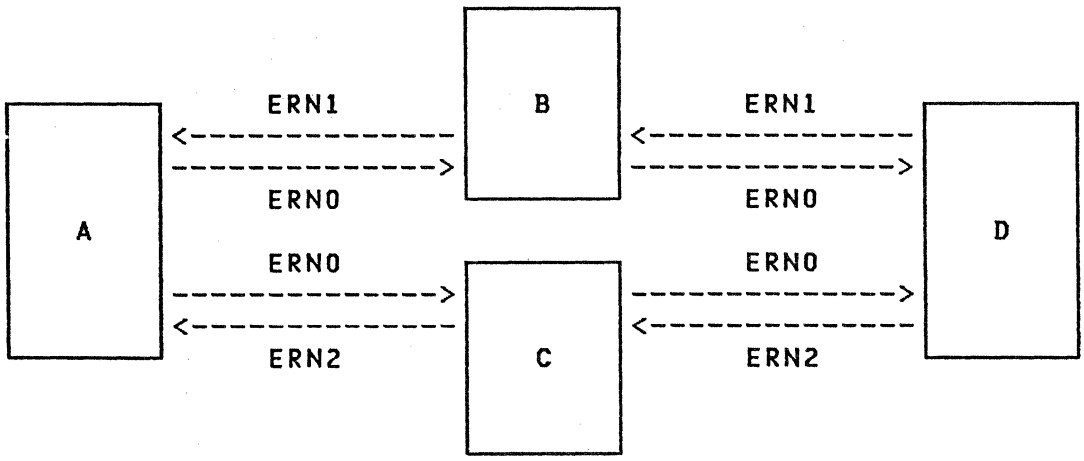


Figure 12-11. Dynamic Route Definition Example

The above dynamic route definition discussion considers the case where only one end of an ER allows dynamic route definition. When both ends allow it, both nodes might be in the process of resolving TG_ID ambiguity at the same time. If both nodes are sending multiple NC_ER_ACT requests and activating the ER based on the order that the NC_ER_ACT_REPLY requests return, a conflict may arise between the TG_IDs chosen for the ER by the two partner nodes. In this case, when two nodes are simultaneously resolving the TG_ID conflict, the subarea node having the larger subarea address is considered the "winner." The winning node rejects all NC_ER_ACT requests being used by the partner node to resolve its TG_ID conflict. The "losing" node accepts the first NC_ER_ACT request it receives from the winning node, and returns a positive NC_ER_ACT_REPLY. (This node may also accept later NC_ER_ACT requests being used to resolve the other node's TG_ID conflict.) The losing node defines its (DSA, ERN) that was being resolved to use the TG_ID over which the ER partner's first NC_ER_ACT arrived, and then sends another NC_ER_ACT request on the now defined TG_ID. This NC_ER_ACT, however, does not indicate (via the DYNAMIC_ER_DEFN bit in the NC_ER_ACT request) that it is being used to resolve a TG_ID conflict.

The TG_ID for an undefined and not yet active (DSA, ERN) can be determined by means other than sending out multiple NC_ER_ACT requests. The simplest case is that the VR manager requests an active ER after one NC_ER_OP has been received for a (DSA, ERN), but before a second has arrived--the TG over which the one NC_ER_OP arrived is used for routing the NC_ER_ACT. The TG can also be determined based on actions by the subarea node at the other end of the ER. If that subarea node sends an NC_ER_ACT, the TG over which it arrived becomes the one to be used for the (DSA, ERN). Once a (DSA, ERN) becomes active, whether it was dynamically or statically defined is irrelevant except that if the (DSA, ERN) ever becomes reset as the result of receiving an NC_ER_INOP, the whole process of determining which TG to use for the (DSA, ERN) starts over.

TESTING OF EXPLICIT ROUTES

Testing of explicit routes is initiated by the ROUTE_TEST request, which is sent from an SSCP to the VR manager in a PU. It specifies another subarea in the network and a list of VRNs or ERNs. The VR manager component handles testing of the VRs (see the "Virtual Route Manager" section of this chapter); the ER manager tests the ERs according to the following rules. If the list contains ERNs, all TGs of all explicit routes using those ERNs for PIUs flowing from the subarea of the PU receiving the ROUTE_TEST request to the subarea specified in the request are tested. If the list contains VRNs, only the defined TG of the explicit route number underlying those virtual route numbers is tested.

Explicit route testing involves two different activities. The first is always performed and entails reporting the states of the ERs, as known in the node receiving the ROUTE_TEST request. These states are reported to the SSCP by means of the ROUTE_TEST response. The second activity is performed depending on a field in the ROUTE_TEST; the second activity determines the condition of the ERs by sending NC_ER_TEST and reporting the result to appropriate CPs using ER_TESTED requests.

One set of procedures in the ER manager determines the status of an ER and puts that value directly into the response to the ROUTE_TEST; another set of procedures provide the protocols for actually testing the ER.

The NC_ER_TEST request is subjected to the same processing as an NC_ER_ACT, however, its operation differs from the NC_ER_ACT in two aspects. First, NC_ER_TEST simply tests the ER, not changing the state of the ER. Second, NC_ER_TEST may be sent even if the explicit route is known to be inoperative, in which case the test fails and the NC_ER_TEST_REPLY identifies the reason and location of the test failure.

If an ER is to be tested, the ER manager sends an NC_ER_TEST along that ER using the sequential propagation flow. At each node along the explicit route, the same checks as for an NC_ER_ACT are performed (existence of a reverse ERN, comparison of the ER length, operative TG, support for ER-VR protocols, and definition of the (DSA, ERN)). An NC_ER_TEST_REPLY is generated at the node where the NC_ER_TEST fails, or at the destination node if the test is successful. Upon receiving the NC_ER_TEST_REPLY, the ER manager generates an ER_TESTED request and sends it to the SSCP that originated the ROUTE_TEST request. If the test fails, the ER manager detecting the failure sends an ER_TESTED for each SSCP-PU session in which SDT has flowed.

```
ER_MGR: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO CALL THE APPROPRIATE ER MANAGER  PROCEDURE TO PROCESS A SIGNAL OR │
 │                 A REQUEST                                            │
 │                                                                      │
 │      INPUT:     SIGNAL      OR      REQUEST      (POSSIBLY      ROUTED      THROUGH │
 │                 PU.SVC_MGR.PC_ROUTE_MGR.RCV) FROM PC.ERC  (CHAPTER 3), PU.SVC_MGR.NS │
 │                 (CHAPTER 11), VR_MGR (CHAPTER 12)                    │
 │                                                                      │
 │      OUTPUT:    SIGNAL OR REQUEST TO APPROPRIATE PROCEDURE           │
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 PU.SVC_MGR.PC_ROUTE_MGR.RCV        PAGE 12-13        │
 │                                                                      │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
 │                 ACT_SEND                           PAGE 12-55        │
 │                 ACT_TEST_RCV                       PAGE 12-60        │
 │                 ACT_TEST_REPLY_RCV                 PAGE 12-64        │
 │                 INOP_RCV                           PAGE 12-44        │
 │                 INOP_SEND                          PAGE 12-42        │
 │                 LSA_RCV                            PAGE 12-45        │
 │                 OP_RCV                             PAGE 12-40        │
 │                 OP_SEND                            PAGE 12-39        │
 │                 TEST_SEND                          PAGE 12-56        │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  SELECT ANYORDER;                                                      /*

                  ┌──────────────────────────────────────┐
                  │          INPUT SIGNAL FROM VR_MGR    │
                  └──────────────────────────────────────┘

                                                                        */
  . WHEN(INPUT('ACTIVATE_ER'))
  .   CALL ACT_SEND;                                  /* PAGE 12-55    */
                                                                        /*

                  ┌──────────────────────────────────────┐
                  │      INPUT SIGNALS FROM PU.SVC_MGR.NS.               │
                  └──────────────────────────────────────┘

                                                                        */
  . WHEN(INPUT('TG_OP'))
  .   CALL OP_SEND;                                   /* PAGE 12-39    */
  .
  . WHEN(INPUT('TG_INOP_NORMAL'))
  .   CALL INOP_SEND;                                 /* PAGE 12-42    */
  .
  . WHEN(INPUT('TG_INOP_ERROR'))
  .   CALL INOP_SEND;                                 /* PAGE 12-42    */
                                                                        /*

                  ┌──────────────────────────────────────┐
                  │          INPUT RU FROM VR_MGR        │
                  └──────────────────────────────────────┘

                                                                        */
  . WHEN(NSC_RQ.NS_HEADER = ROUTE_TEST_HDR)
  .   CALL TEST_SEND;                                 /* PAGE 12-56    */
                                                                        /*

                  ┌──────────────────────────────────────┐
                  │          INPUT RU'S FROM PC.ERC      │
                  └──────────────────────────────────────┘

                                                                        */
  . WHEN(RU_CTGY = NC)
  .   SELECT ANYORDER(RQ_CODE);
  .   . WHEN(NC_ER_OP)
  .   .   CALL OP_RCV;                                /* PAGE 12-40    */
  .   . WHEN(NC_ER_INOP)
  .   .   CALL INOP_RCV;                              /* PAGE 12-44    */
  .   . WHEN(LSA)
  .   .   CALL LSA_RCV;                               /* PAGE 12-45    */
  .   . WHEN(NC_ER_ACT)
  .   .   CALL ACT_TEST_RCV;                          /* PAGE 12-60    */
  .   . WHEN(NC_ER_ACT_REPLY)
  .   .   CALL ACT_TEST_REPLY_RCV;                    /* PAGE 12-64    */
  .   . WHEN(NC_ER_TEST)
  .   .   CALL ACT_TEST_RCV;                          /* PAGE 12-60    */
  .   . WHEN(NC_ER_TEST_REPLY)
  .   .   CALL ACT_TEST_REPLY_RCV;                    /* PAGE 12-64    */
  .   END;
  .
  .
  . OTHERWISE
  .   DISCARD MU;
  END;

  RETURN;
END ER_MGR;
```

EXPLICIT ROUTING DEFINITION

An explicit route is defined when it is given a mapping to a particular TG_ID, i.e., when an entry exists in the SUBAREA_ROUTING_LIST for the (DSA, ERN). Generally, this mapping for every subarea node in a network is generated during system definition; those subarea nodes having explicit route redefinition capability allow this mapping to be changed while the network is in operation.

A UPM generates the signal and associated information detailing the changes to be made to the explicit routing tables. The ER manager performs the requested changes only if the explicit route is not currently active or in the process of being activated. The routing definition may require an NC_ER_ACT to be sent into the network. A received NC_ER_ACT may have been rejected previously because there were no ERNs defined from the node to the NC_ER_ACT originator along the required sequence of transmission groups. If its ER definition provides such an ERN, the node sends an NC_ER_ACT to the node that sent the previously rejected NC_ER_ACT.

```
DEFINE_ER_TO_TG: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO ALTER THE (DSA, ERN) TO TG_ID MAPPING SPECIFIED IN THE    │
  │                 SUBAREA_ROUTING_LIST                                         │
  │                                                                              │
  │      INPUT:     FROM UPM, PARM_DEFINE_ER ADDRESSED BY PARM_PTR               │
  │                                                                              │
  │      OUTPUT:    UPDATED SUBAREA_ROUTING, AND POSSIBLY AN NC_ER_ACT ON THE (DSA, ERN)
  │                 (DONE IN FSM_PATH--PAGE 12-75)                               │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 CREATE_SUBAREA_ROUTING          PAGE 12-67                   │
  │                 FSM_PATH                        PAGE 12-75                   │
  │                 UPM_ALLOW_ER_DEFINITION         PAGE 12-33                   │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  ENTITY(PARM_DEFINE_ER),
        2 DEST_SA BIT(32),              /* SUBAREA AT OTHER END OF THE ER       */
        2 ER_NUM BIT(8),                /* EXPLICIT ROUTE NUMBER BEING DEFINED  */
        2 TG_ID,
          3 TGN BIT(8),                 /* TG USED TO NEXT SUBAREA ALONG ER     */
          3 ADJ_SA BIT(32);             /* NEXT SUBAREA ALONG ER                */

  PARM_DEFINE_ER_PTR = PARM_PTR;

                                                                                /*
                    ┌──────────────────────────────────────────┐
                    │   LOCATE ANY CONTROL BLOCKS NEEDED TO CHANGE
                    │              ROUTING DEFINITIONS.        │
                    └──────────────────────────────────────────┘
                                                                                */
  FIND SUBAREA_ROUTING IN SUBAREA_ROUTING_LIST
       WHERE(SUBAREA_ROUTING.DEST_SA = PARM_DEFINE_ER.DEST_SA);
  IF SUBAREA_ROUTING_PTR = NULL THEN
    DO;
    . CALL CREATE_SUBAREA_ROUTING(PARM_DEFINE_ER.DEST_SA);      /* PAGE 12-67   */
    . SUBAREA_ROUTING.ER_SYSDEF(PARM_DEFINE_ER.ER_NUM) = STATIC_DEFINITION;
    END;


  FIND ERCB IN ERCB_LIST
       WHERE(ERCB.PARTNER_SA = PARM_DEFINE_ER.DEST_SA & ERCB.ER_NUM = PARM_DEFINE_ER.ER_NUM);

  IF ERCB_PTR = NULL THEN
    SUBAREA_ROUTING.TG_ID(PARM_DEFINE_ER.ER_NUM) = PARM_DEFINE_ER.TG_ID;
  ELSE
    IF UPM_ALLOW_ER_DEFINITION = YES THEN                       /* PAGE 12-33   */
      DO;
      . SUBAREA_ROUTING.TG_ID(PARM_DEFINE_ER.ER_NUM) = PARM_DEFINE_ER.TG_ID;
      .
      . FIND PATHCB IN PATHCB_LIST WHERE(PATHCB.TG_ID = PARM_DEFINE_ER.TG_ID);
      .
      . IF PATHCB_PTR ¬= NULL THEN
      .   DO;
      .   . FIND TGCB IN TGCB_LIST WHERE(TGCB.TG_ID = PARM_DEFINE_ER.TG_ID);
      .   . CALL FSM_PATH('DEFINE');                            /* PAGE 12-75   */
      .   END;
      END;

  RETURN;
END DEFINE_ER_TO_TG;



UPM_ALLOW_ER_DEFINITION: PROCEDURE RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO DETERMINE WHETHER THE ER DEFINITION REQUEST IS VALID ACCORDING TO
  │                 THE STATE OF THE ER BEING REDEFINED                          │
  │                                                                              │
  │      INPUT:     ERCB_PTR                                                     │
  │                                                                              │
  │      OUTPUT:    YES IS RETURNED IF ER DEFINITION IS ALLOWED; NO IS RETURNED IF ER
  │                 DEFINITION IS NOT ALLOWED.  CHANGED PATHCB_PTR.              │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 DEFINE_ER_TO_TG                 PAGE 12-33                   │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  RETURN(YES);

END UPM_ALLOW_ER_DEFINITION;
```

OPERATIONAL STATUS OF EXPLICIT ROUTES

The operational status of explicit routes is communicated between subarea nodes using NC_ER_OP and NC_ER_INOP. Both requests use the fan-out propagation flow (implemented in procedure FANOUT_PROP) to disseminate the new routing status to all affected subarea nodes. When an ER becomes inoperative, appropriate messages are sent to the VR manager (ERINOP signal) and the affected SSCPs and PUCP (ER_INOP request).

A subarea node that does not support ER-VR protocols is never sent an NC_ER_OP; any NC_ER_INOP requests meant for such a node are converted to LSA requests before being sent.

EXPLICIT ROUTE OPERATIVE (NC_ER_OP)
EXPLICIT ROUTE INOPERATIVE (NC_ER_INOP)

Flow: ER manager to ER manager (Expedited), with TG Sweep = ¬SWEEP, at high transmission priority

Principal FSMs: FSM_PATH (Page 12-75)
FSM_ERN (Page 12-73)

The NC_ER_OP is generated when a link of an inoperative transmission group becomes operative. The ER managers in the subarea nodes on each side of the transmission group generate and exchange NC_ER_OP requests. Each NC_ER_OP contains a specification of the explicit routing knowledge in the originating ER manager--the set of (DSA, ERN) pairs that can be used for routing PIUs (i.e., have nonzero TG_ID values in the SUBAREA_ROUTING_LIST) and that are known to be operational as a result of previous NC_ER_OP flows. The set of (DSA, ERN) pairs for each DSA is represented in the NC_ER_OP as a single DSA value (SA) and a 16-bit mask (MASK) indicating which ERNs are operative. The NC_ER_OP also includes the transmission group number of the operative transmission group and the subarea addresses at its two ends.

Any subarea node receiving an NC_ER_OP modifies its own routing tables and the routing information in the request. A path control block (PATHCB) with an FSM in the operative state is attached to each affected ERCB, signifying that routing for that (DSA, ERN) is available using the TG over which the NC_ER_OP arrived. The NC_ER_OP contains only routing information that may be used by receiving nodes when they attempt to activate an explicit route. An NC_ER_OP may specify routing for a particular (DSA, ERN), but using a different TG than is defined in the SUBAREA_ROUTING_LIST in the receiving node. In such a case, where the routing information from the NC_ER_OP does not match the node's definitions, the specification in the NC_ER_OP is erased so that nodes receiving the propagated NC_ER_OP will not try to route traffic for that (DSA, ERN) through this node. After processing all (DSA, ERN) pairs in the NC_ER_OP, the updated NC_ER_OP is propagated on each transmission group to each adjacent subarea node, except the node from which it arrived. The request is not propagated if there are no remaining entries in the list of operative explicit routes.

The NC_ER_INOP is initiated when the last remaining link of the transmission group has failed or is discontacted via a link-level procedure. It is originated by the ER managers in the nodes on each side of the transmission group, and sent on each operative transmission group to each adjacent subarea node. Each originating node builds an NC_ER_INOP that contains a specification of the explicit routing that

is no longer possible--the set of (DSA, ERN) pairs that have become inoperative as a result of the transmission group becoming inoperative. As in NC_ER_OP, the set of (DSA, ERN) pairs for each DSA is represented in NC_ER_INOP as a single DSA value (SA) and a 16-bit mask (MASK) indicating which ERNs are inoperative. The NC_ER_INOP also contains the transmission group number and two subarea addresses that designate the inoperative transmission group.

As discussed for NC_ER_OP, a node's routing tables may not match the routing information specified in the NC_ER_INOP. Each receiving subarea node deletes those entries from the (DSA, ERN) specification in the NC_ER_INOP for which the TG over which it arrived does not correspond to the TG_ID in the routing tables (SUBAREA_ROUTING_LIST). After processing all (DSA, ERN) pairs in the NC_ER_INOP, the updated NC_ER_INOP is transmitted using fan-out propagation, unless there are no remaining entries in the list of inoperative explicit routes.

EXPLICIT ROUTE INOPERATIVE (ER_INOP)

Flow: ER manager to CP (Normal)

Principal FSMs: None

ER_INOP is generated by the ER manager when it receives an NC_ER_INOP, and optionally is sent for each CP-PU session (in which SDT has flowed) to notify each CP that certain ERs have become inoperative. The ER_INOP includes the list of (DSA, ERN) pairs (corresponding to inoperative explicit routes) that were included in the NC_ER_INOP.

NETWORK SERVICES LOST SUBAREA (NS_LSA)

Flow: ER manager to SSCP (Normal)

Principal FSMs: None

Upon receiving an NC_ER_INOP, the ER manager generates and sends an NS_LSA instead of an ER_INOP for each SSCP-PU session in which SDT has flowed and the SSCP does not support ER-VR protocols. The NS_LSA includes the list of destination subarea addresses included in the NC_ER_INOP.

LOST SUBAREA (LSA)

Flow: ER manager to PU (Normal)

Principal FSMs: None

When LSA is received from a node that does not support ER-VR protocols, the ER manager converts it to an NC_ER_INOP and processes it accordingly. If the node to which an NC_ER_INOP is to be sent does not support ER-VR protocols, the ER manager transforms the NC_ER_INOP into an LSA. The LSA includes the list of destination subarea addresses included in the NC_ER_INOP, but no ERN values.

This page intentionally left blank

```
OP_SEND: PROCEDURE;

                                                                                /*
┌──────────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO CREATE AND SEND NC_ER_OP UPON RECEIPT OF A TG_OP SIGNAL FROM
│                 PU.SVC_MGR.NS (CHAPTER 11).  THE NC_ER_OP IS SENT TO THE OTHER
│                 SUBAREA NODE ATTACHED TO THE TRANSMISSION GROUP THAT JUST BECAME
│                 OPERATIVE.  NC_ER_OP SPECIFIES THE SET OF EXPLICIT ROUTES FROM THIS
│                 NODE THAT COULD BE USED FOR ROUTING PIU'S TO SOME DESTINATION BEFORE
│                 THE TG BECAME OPERATIVE.  TO BE INCLUDED IN THIS LIST, A (DSA, ERN)
│                 MUST BE OPERATIVE AND BE DEFINED (I.E., HAVE AN ENTRY IN THE
│                 SUBAREA_ROUTING_LIST IDENTIFYING WHICH TG_ID TO USE WHEN ROUTING
│                 PIU'S).
│
│      INPUT:     TG_OP SIGNAL AND TGCB_PTR (IDENTIFYING THE TGCB FOR THE NEWLY
│                 OPERATIVE TG)
│
│      OUTPUT:    NC_ER_OP TO PC.TGC (CHAPTER 3)
│
│      NOTES:     1. IF THE OTHER SUBAREA NODE DOES NOT SUPPORT ER-VR PROTOCOLS, DO
│                    NOT SEND AN NC_ER_OP TO IT.
│
│                 2. NC_ER_OP MAY PASS OTHER NETWORK CONTROL RU'S (E.G., NC_ER_ACT)
│                    THAT FLOW WITH TPF=L_PRTY.
│
│                 3. THE FIRST ER_FIELD ENTRY SPECIFIES ROUTING CAPABILITY TO THE
│                    CURRENT NODE (I.E., THE NODE GENERATING THE NC_ER_OP).
│
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                 ER_MGR                              PAGE 12-31
│
│      REFERS TO THE FOLLOWING PROCEDURE(S):
│                 BUILD_NC_TH_RH                      PAGE 12-123
│                 FSM_ERN                             PAGE 12-73
└──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
DCL ER_NUM BIT(4);                      /* USED TO INDEX FOR ER NUMBERS        */


IF TGCB.ER_VR_SUPP = NO THEN                                    /* APPENDIX A  */
  RETURN;                                                       /* NOTE 1      */

CREATE MU;
CALL BUILD_NC_TH_RH(MU_PTR);                                    /* PAGE 12-123 */

TG_SWEEP = ¬SWEEP;
ERN = RESERVED_ZERO;
IERN = RESERVED_ZERO;
VRN = RESERVED_ZERO;
TPF = H_PRTY;                                                   /* NOTE 2      */
DSAF = SUBAREA_ROUTING.DEST_SA;                                 /* APPENDIX A  */

RQ_CODE = NC_ER_OP;
NC_ER_OP_RQ.FORMAT = FORMAT1;
NC_ER_OP_RQ.ORIGINATING_SA = NCB.NODE_SUBAREA_ADDRESS;          /* APPENDIX A  */
NC_ER_OP_RQ.TG_ADJ_SA = TGCB.ADJ_SA;                            /* APPENDIX A  */
NC_ER_OP_RQ.TG_NUM = TGCB.TGN;                                  /* APPENDIX A  */

NC_ER_OP_RQ.SA(1) = NCB.NODE_SUBAREA_ADDRESS;                   /* NOTE 3, APPENDIX A */
NC_ER_OP_RQ.MASK(1) = ALL_ON;
NC_ER_OP_RQ.CNT_ER_FIELD = 2;

                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │          CREATE ENTRIES FOR ER_FIELD     │
                  └──────────────────────────────────────────┘
                                                                                */
SCAN SUBAREA_ROUTING_LIST PTR(SUBAREA_ROUTING_PTR);
. NC_ER_OP_RQ.SA(NC_ER_OP_RQ.CNT_ER_FIELD) = SUBAREA_ROUTING.DEST_SA;
. NC_ER_OP_RQ.MASK(NC_ER_OP_RQ.CNT_ER_FIELD) = ALL_OFF;
.
. DO ER_NUM = 0 TO NCB.MAX_ER_NUM;                              /* APPENDIX A  */
. . FIND ERCB IN ERCB_LIST
. .      WHERE(ERCB.PARTNER_SA = SUBAREA_ROUTING.DEST_SA & ERCB.ER_NUM = ER_NUM);
. . IF ERCB_PTR ¬= NULL THEN
. .   FIND PATHCB IN PATHCB_LIST WHERE(PATHCB.TG_ID = SUBAREA_ROUTING.TG_ID(ER_NUM));
. . ELSE
. .   PATHCB_PTR = NULL;
. . IF PATHCB_PTR ¬= NULL |
. .      (ERCB_PTR ¬= NULL & FSM_ERN = CONTEND) THEN            /* PAGE 12-73  */
. .   NC_ER_OP_RQ.MASK(NC_ER_OP_RQ.CNT_ER_FIELD,ER_NUM:ER_NUM) = ON;
. END;
.
. IF NC_ER_OP_RQ.MASK(NC_ER_OP_RQ.CNT_ER_FIELD) ¬= ALL_OFF THEN
.   NC_ER_OP_RQ.CNT_ER_FIELD = NC_ER_OP_RQ.CNT_ER_FIELD + 1;
SCANEND;

NC_ER_OP_RQ.CNT_ER_FIELD = NC_ER_OP_RQ.CNT_ER_FIELD - 1;
DCF = RH_LENGTH + 15 + (6 * NC_ER_OP_RQ.CNT_ER_FIELD);          /* 15 IS THE LENGTH */
                                        /* OF THE FIXED PART OF THE RU AND     */
                                        /* 6 IS THE LENGTH OF THE ARRAY ELEMENTS */
SEND MU TO PC.TGC.LIST_BY_PRTY;                                 /* CHAPTER 3   */

RETURN;
END OP_SEND;
```

```
OP_RCV: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO UPDATE ROUTING TABLES BASED ON A RECEIVED NC_ER_OP THAT │
│                SPECIFIES OPERATIVE (DSA, ERN) ROUTES. FOR EACH (DSA,     │
│                ERN) THAT HAS BECOME OPERATIVE, A PATHCB IS BUILT AND     │
│                INITIALIZED. IF NO PATHCB EXISTS FOR THE (DSA, ERN),      │
│                THIS PROCEDURE ALSO BUILDS AND INITIALIZES AN ERCB. THE   │
│                NC_ER_OP IS PROPAGATED TO OTHER SUBAREA NODES IF THE RU   │
│                SPECIFIES ROUTING INFORMATION STILL TO BE DISTRIBUTED     │
│                THROUGH THE NETWORK.                                      │
│                                                                          │
│     INPUT:     NC_ER_OP AND TGCB_PTR (INDICATING THE TG OVER WHICH THE   │
│                REQUEST WAS RECEIVED)                                     │
│                                                                          │
│     OUTPUT:    CREATED AND INITIALIZED PATHCB'S (AND MAYBE ERCB'S) FOR   │
│                EACH ERN TO A DSA THAT IS NOW OPERATIVE. IF APPROPRIATE,  │
│                COPIES OF THE NC_ER_OP ARE PROPAGATED TO ADJACENT SUBAREA │
│                NODES (BY FANOUT_PROP). THE RECEIVED NC_ER_OP IS          │
│                DISCARDED.                                                │
│                                                                          │
│     NOTES:     1. ANY ER_FIELD WHOSE MASK INDICATES THAT NO ERN'S ARE     │
│                   OPERATIVE IS ELIMINATED FROM THE NC_ER_OP REQUEST.     │
│                   THIS ARRAY POSITION MARKER POINTS TO THE NEXT          │
│                   LOCATION THAT A VALID ER_FIELD (ONE THAT INDICATES     │
│                   AT LEAST ONE OPERATIVE ERN) IS MOVED TO.               │
│                                                                          │
│                2. A SUBAREA NODE MAY RECEIVE AN NC_ER_OP SPECIFYING      │
│                   ROUTING FOR A (DSA, ERN) WHERE THE DSA IS THE SAME     │
│                   AS THE LOCAL SUBAREA ADDRESS. THIS SITUATION ARISES    │
│                   IF EITHER ANOTHER NODE IN THE NETWORK HAS THE SAME     │
│                   SUBAREA ADDRESS (A SYSTEM DEFINITION ERROR) AND CAN    │
│                   NOW BE ROUTED TO, OR THE LOCAL SUBAREA CAN BE ROUTED   │
│                   TO FROM BOTH SUBAREAS AT THE ENDS OF THE TG THAT       │
│                   BECAME OPERATIVE. IN BOTH CASES THE ROUTING            │
│                   INFORMATION IN THE NC_ER_OP FOR THIS DSA IS IGNORED.   │
│                                                                          │
│                3. IF NO SUBAREA_ROUTING_LIST ENTRY EXISTS FOR A DSA      │
│                   SPECIFIED IN THE NC_ER_OP AND THIS NODE ALLOWS         │
│                   DYNAMIC DEFINITION OF ROUTING, GENERATE A              │
│                   SUBAREA_ROUTING_LIST ENTRY FOR THAT DSA AND DEFINE     │
│                   THE ERN'S (AS SPECIFIED IN THE MASK FIELD) TO USE      │
│                   THE TG OVER WHICH THE NC_ER_OP ARRIVED.                │
│                                                                          │
│                4. CREATE A PATHCB IN WHICH TO ANCHOR THE FSM_PATH.       │
│                                                                          │
│                5. THE PROPAGATED NC_ER_OP RETAINS ROUTING INFORMATION    │
│                   FOR A (DSA, ERN) ONLY IF THE TG ON WHICH THE           │
│                   NC_ER_OP ARRIVED MATCHES THE DEFINITION AS GIVEN IN    │
│                   THE SUBAREA_ROUTING_LIST, OR IF THAT (DSA, ERN) IS     │
│                   NOT DEFINED IN THE SUBAREA_ROUTING_LIST AND THIS IS    │
│                   THE FIRST NC_ER_OP RECEIVED FOR THE (DSA, ERN).        │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                ER_MGR                              PAGE 12-31            │
│                                                                          │
│     REFERS TO THE FOLLOWING PROCEDURE(S):                                │
│                CREATE_SUBAREA_ROUTING              PAGE 12-67            │
│                FANOUT_PROP                         PAGE 12-46            │
│                FSM_ERN                             PAGE 12-73            │
│                FSM_PATH                            PAGE 12-75            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

```
DCL DSA_CNT BIT(8);                     /* USED TO INDEX ER_FIELD                    */
DCL ER_NUM BIT(4);                      /* INDEX ERNS TO EACH DEST SA                */
DCL MOVE_TO BIT(8);                                          /* NOTE 1               */

MOVE_TO = 1;
DO DSA_CNT = 1 TO NC_ER_OP_RQ.CNT_ER_FIELD;
. IF NCB.NODE_SUBAREA_ADDRESS ¬= NC_ER_OP_RQ.SA(DSA_CNT) THEN      /* APPENDIX A      */
.   DO;                                                          /* NOTE 2          */
.   . FIND SUBAREA_ROUTING IN SUBAREA_ROUTING_LIST
.   .      WHERE(SUBAREA_ROUTING.DEST_SA = NC_ER_OP_RQ.SA(DSA_CNT));
.   .
.   . IF SUBAREA_ROUTING_PTR = NULL & NC_ER_OP_RQ.MASK(DSA_CNT) ¬= ALL_OFF THEN
.   .   IF NCB.ERN_DEFINITION_CAPABILITY = ¬STATIC_ONLY THEN  /* APPENDIX A          */
.   .     DO;                                                  /* NOTE 3          */
.   .     . CALL CREATE_SUBAREA_ROUTING(NC_ER_OP_RQ.SA(DSA_CNT));  /* PAGE 12-67  */
.   .     . DO ER_NUM = 0 TO NCB.MAX_ER_NUM;                   /* APPENDIX A      */
.   .     . . IF NC_ER_OP_RQ.MASK(DSA_CNT,ER_NUM:ER_NUM) = ON THEN
.   .     . .   SUBAREA_ROUTING.TG_ID(ER_NUM) = TGCB.TG_ID;
.   .     . END;
.   .     END;
.   .   ELSE
.   .     NC_ER_OP_RQ.MASK(DSA_CNT) = ALL_OFF;
.   .
.   .
.   . NC_ER_OP_RQ.SA(MOVE_TO) = NC_ER_OP_RQ.SA(DSA_CNT);
.   . NC_ER_OP_RQ.MASK(MOVE_TO) = NC_ER_OP_RQ.MASK(DSA_CNT);
.   .
.   . DO ER_NUM = 0 TO NCB.MAX_ER_NUM;                         /* APPENDIX A      */
.   . . IF NC_ER_OP_RQ.MASK(MOVE_TO,ER_NUM:ER_NUM) = ON THEN
.   . .   DO;
.   . .   . FIND ERCB IN ERCB_LIST
.   . .   .      WHERE(ERCB.PARTNER_SA = NC_ER_OP_RQ.SA(DSA_CNT) &
.   . .   .      ERCB.ER_NUM = ER_NUM);
.   . .   . IF ERCB_PTR = NULL THEN
.   . .   .   DO;                                             /* BUILD NEW ERCB  */
.   . .   .   . CREATE ERCB;
.   . .   .   . INSERT ERCB IN ERCB_LIST;
.   . .   .   . ERCB.PARTNER_SA = NC_ER_OP_RQ.SA(DSA_CNT);
.   . .   .   . ERCB.ER_NUM = ER_NUM;
.   . .   .   . NEWLIST ERCB.PATHCB_LIST ENTRY_NAME(PATHCB);
.   . .   .   . PATHCB_PTR = NULL;
.   . .   .   END;
.   . .   . FIND PATHCB IN ERCB.PATHCB_LIST WHERE(PATHCB.TG_ID = TGCB.TG_ID);
.   . .   .
.   . .   . IF PATHCB_PTR = NULL THEN                         /* NOTE 4          */
.   . .   .   DO;
.   . .   .   . CREATE PATHCB;                                 /* APPENDIX A      */
.   . .   .   . PATHCB.TG_ID = TGCB.TG_ID;
.   . .   .   . INSERT PATHCB IN ERCB.PATHCB_LIST;
.   . .   .   END;
.   . .   .
.   . .   . CALL FSM_PATH;                                     /* PAGE 12-75      */
.   . .   . CALL FSM_ERN;                                      /* PAGE 12-73      */
.   . .   . IF SUBAREA_ROUTING.TG_ID(ERCB.ER_NUM) ¬= PATHCB.TG_ID THEN
.   . .   .   NC_ER_OP_RQ.MASK(MOVE_TO,ER_NUM:ER_NUM) = OFF;  /* NOTE 5           */
.   . .   END;
.   . .
.   . END;
                                                                                    /*
                  ┌──────────────────────────────────────────┐
                  | IF NO MASK BITS IN THE REQUEST ARE TURNED ON,  |
                  | THEN NO ROUTING INFORMATION IS BEING  |
                  | PROPAGATED FOR THIS DSA, AND THEREFORE, THIS  |
                  | ER_FIELD IS ELIMINATED FROM THE RU. BY NOT  |
                  | INCREMENTING MOVE_TO, THE NEXT ER_FIELD WILL  |
                  | BE PLACED ON TOP OF THIS ER_FIELD.  |
                  └──────────────────────────────────────────┘
                                                                                    */
.   . IF NC_ER_OP_RQ.MASK(MOVE_TO) ¬= ALL_OFF THEN
.   .   MOVE_TO = MOVE_TO + 1;
.   .
.   END;
END;

NC_ER_OP_RQ.CNT_ER_FIELD = MOVE_TO - 1;        /* NUMBER OF NON-EMPTY ER_OP_FIELD'S */
                                                                                    /*
                  ┌──────────────────────────────────────────┐
                  | DETERMINE IF THE NC_ER_OP IS TO BE PROPAGATED  |
                  └──────────────────────────────────────────┘
                                                                                    */
IF NCB.INTERMEDIATE_FUNCTION = YES & NC_ER_OP_RQ.CNT_ER_FIELD ¬= 0 THEN  /* APPENDIX A */
  CALL FANOUT_PROP;                                          /* PAGE 12-46        */

DISCARD MU;

RETURN;
END OP_RCV;
```

```
INOP_SEND: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO CREATE AND SEND NC_ER_INOP UPON RECEIPT OF A TG_INOP_NORMAL OR
 │                 TG_INOP_ERROR SIGNAL FROM PU.SVC_MGR.NS (CHAPTER 11). THE
 │                 NC_ER_INOP IS SENT TO ALL ADJACENT SUBAREA NODES, INDICATING WHICH
 │                 EXPLICIT ROUTES ARE NO LONGER OPERATIVE.
 │
 │      INPUT:     EITHER A TG_INOP_NORMAL OR TG_INOP_ERROR SIGNAL AND TGCB_PTR
 │                 (INDICATING THE TGCB FOR THE NOW INOPERATIVE TG)
 │
 │      OUTPUT:    NC_ER_INOP REQUEST TO PC (CHAPTER 3) (BY FANOUT_PROP), ER_INOP
 │                 REQUEST TO PC (CHAPTER 3) (BY NS_ER_INOP_SEND), ERINOP SIGNAL TO
 │                 VR_MGR (BY VRMGR_INOP_SEND)
 │
 │      NOTE:      NC_ER_INOP MAY MOVE AHEAD OF OTHER NC RUS (E.G., NC_ER_ACT) THAT
 │                 FLOW WITH TPF=L_PRTY.
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 ER_MGR                              PAGE 12-31
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 ARE_ANY_PATHS_PENDING               PAGE 12-72
 │                 BUILD_NC_TH_RH                      PAGE 12-123
 │                 FANOUT_PROP                         PAGE 12-46
 │                 FSM_ERN                             PAGE 12-73
 │                 FSM_PATH                            PAGE 12-75
 │                 NS_ER_INOP_SEND                     PAGE 12-47
 │                 VRMGR_INOP_SEND                     PAGE 12-48
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
 DCL ER_NUM BIT(4);                                     /* TO INDEX ERN'S  */

                                                                        /*
                 ┌──────────────────────────────────────┐
                 │       CREATE AND INITIALIZE NC_ER_INOP TO       │
                 │          DISTRIBUTE INOPERATIVE STATUS          │
                 └──────────────────────────────────────┘
                                                                        */
 CREATE MU;
 CALL BUILD_NC_TH_RH(MU_PTR);                           /* PAGE 12-123     */

 TG_SWEEP = ¬SWEEP;
 ERN = RESERVED_ZERO;
 IERN = RESERVED_ZERO;
 VRN = RESERVED_ZERO;
 TPF = H_PRTY;                                          /* NOTE            */

 RQ_CODE = NC_ER_INOP;
 NC_ER_INOP_RQ.FORMAT = FORMAT1;

 SELECT ANYORDER;
 . WHEN(INPUT('TG_INOP_ERROR'))
 .    NC_ER_INOP_RQ.REASON_CODE = X'01';
 . WHEN(INPUT('TG_INOP_NORMAL'))
 .    NC_ER_INOP_RQ.REASON_CODE = X'02';
 END;

 NC_ER_INOP_RQ.ORIGINATING_SA = NCB.NODE_SUBAREA_ADDRESS;  /* APPENDIX A      */
 NC_ER_INOP_RQ.TG_ADJ_SA = TGCB.ADJ_SA;
 NC_ER_INOP_RQ.TG_NUM = TGCB.TGN;
 NC_ER_INOP_RQ.CNT_ER_FIELD = 1;
```

```
                                                                                /*
                    ┌──────────────────────────────────────┐
                    │  CREATE ENTRIES FOR ER_FIELD         │
                    └──────────────────────────────────────┘
                                                                                */
  SCAN SUBAREA_ROUTING_LIST PTR(SUBAREA_ROUTING_PTR);
  . NC_ER_INOP_RQ.SA(NC_ER_INOP_RQ.CNT_ER_FIELD) = SUBAREA_ROUTING.DEST_SA;
  .
  . DO ER_NUM = 0 TO NCB.MAX_ER_NUM;                          /* APPENDIX A      */
  . . FIND ERCB IN ERCB_LIST
  . .      WHERE(ERCB.PARTNER_SA = SUBAREA_ROUTING.DEST_SA & ERCB.ER_NUM = ER_NUM);
  . . IF ERCB_PTR ¬= NULL THEN
  . .   DO;
  . .   . FIND PATHCB IN PATHCB_LIST WHERE(PATHCB.TG_ID = TGCB.TG_ID);
  . .   . IF PATHCB_PTR ¬= NULL THEN
  . .   .   DO;
  . .   .   . IF (SUBAREA_ROUTING.TG_ID(ERCB.ER_NUM) = TGCB.TG_ID) |
  . .   .   .      (FSM_ERN = CONTEND &  ¬ARE_ANY_PATHS_PENDING) THEN /* PAGE 12-73, 12-72  */
  . .   .   .   NC_ER_INOP_RQ.MASK(NC_ER_INOP_RQ.CNT_ER_FIELD,ER_NUM:ER_NUM) = ON;
  . .   .   . CALL FSM_ERN;                                   /* PAGE 12-73      */
  . .   .   . CALL FSM_PATH;                                  /* PAGE 12-75      */
  . .   .   END;
  . .   END;
  . END;
  .
  . IF NC_ER_INOP_RQ.MASK(NC_ER_INOP_RQ.CNT_ER_FIELD) ¬= ALL_OFF THEN
  .   NC_ER_INOP_RQ.CNT_ER_FIELD = NC_ER_INOP_RQ.CNT_ER_FIELD + 1;
  SCANEND;

  NC_ER_INOP_RQ.CNT_ER_FIELD = NC_ER_INOP_RQ.CNT_ER_FIELD - 1;
  DCF = RH_LENGTH + 15 + (6 * NC_ER_INOP_RQ.CNT_ER_FIELD);      /* 15 IS THE LENGTH  */
                                          /* OF THE FIXED PART OF THE RU AND       */
                                          /* 6 IS THE LENGTH OF THE ARRAY ELEMENTS */


  IF NC_ER_INOP_RQ.CNT_ER_FIELD ¬= 0 THEN
    DO;
    . IF NCB.INTERMEDIATE_FUNCTION = YES THEN
    .   CALL FANOUT_PROP;                                     /* PAGE 12-46      */
    . CALL NS_ER_INOP_SEND;                                   /* PAGE 12-47      */
    . CALL VRMGR_INOP_SEND;                                   /* PAGE 12-48      */
    END;

  DISCARD MU;

  RETURN;
END INOP_SEND;
```

```
INOP_RCV: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO UPDATE ROUTING TABLES BASED ON A RECEIVED NC_ER_INOP THAT │
│                SPECIFIES (DSA, ERN) PAIRS THAT ARE NO LONGER OPERATIVE. FOR EACH │
│                (DSA, ERN) THAT HAS BECOME INOPERATIVE, THE PATHCB IS DESTROYED. IF │
│                NO OTHER PATHCB EXISTS FOR THE (DSA, ERN), THE ENTIRE ERCB IS │
│                DESTROYED. THE NC_ER_INOP IS PROPAGATED TO OTHER SUBAREA NODES IF │
│                IT SPECIFIES ROUTING INFORMATION STILL TO BE DISTRIBUTED THROUGH THE │
│                NETWORK.                                                  │
│                                                                          │
│     INPUT:     NC_ER_INOP AND TGCB_PTR (INDICATING OVER WHICH TG THE REQUEST WAS │
│                RECEIVED)                                                 │
│                                                                          │
│     OUTPUT:    NC_ER_INOP REQUEST TO PC (CHAPTER 3) (BY FANOUT_PROP), ER_INOP │
│                REQUEST TO PC (CHAPTER 3) (BY NS_ER_INOP_SEND), ERINOP SIGNAL TO │
│                VR_MGR (BY VRMGR_INOP_SEND); RECEIVED NC_ER_INOP REQUEST IS │
│                DISCARDED.                                                │
│                                                                          │
│     NOTE:      EVERY ER_FIELD WHOSE MASK DOES NOT SPECIFY ANY ERN IS ELIMINATED │
│                FROM THE NC_ER_INOP REQUEST. THIS ARRAY POSITION MARKER POINTS TO │
│                THE NEXT LOCATION THAT A VALID ER_FIELD (ONE THAT SPECIFIES AT LEAST │
│                ONE ERN) IS TO BE MOVED TO.                               │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                ER_MGR                          PAGE 12-31                │
│                LSA_RCV                         PAGE 12-45                │
│                                                                          │
│     REFERS TO THE FOLLOWING PROCEDURE(S):                                │
│                ARE_ANY_PATHS_PENDING           PAGE 12-72                │
│                FANOUT_PROP                     PAGE 12-46                │
│                FSM_ERN                         PAGE 12-73                │
│                FSM_PATH                        PAGE 12-75                │
│                NS_ER_INOP_SEND                 PAGE 12-47                │
│                VRMGR_INOP_SEND                 PAGE 12-48                │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

```
  DCL ER_NUM BIT(4);                    /* USED TO INDEX IN ARRAYS OF ER'S.          */
  DCL DSA_CNT BIT(8);                   /* INDEX INTO ER_FIELD                       */
  DCL MOVE_TO BIT(8);                                   /* NOTE                      */

                                                                                    /*
                  ┌──────────────────────────────────────────┐
                  │ FOR EACH ENTRY IN ER_FIELD, CALL FSM TO FREE │
                  │ THE STORAGE FOR PATHCB (AND POSSIBLY ERCB).  │
                  └──────────────────────────────────────────┘
                                                                                    */
  MOVE_TO = 1;                          /* LOCATION TO PUT NEXT VALID ER_FIELD       */

  DO DSA_CNT = 1 TO NC_ER_INOP_RQ.CNT_ER_FIELD;
  . NC_ER_INOP_RQ.ER_FIELD(MOVE_TO) = NC_ER_INOP_RQ.ER_FIELD(DSA_CNT);
  .
  . DO ER_NUM = 0 TO NCB.MAX_ER_NUM;                    /* APPENDIX A                */
  . . IF NC_ER_INOP_RQ.MASK(MOVE_TO,ER_NUM:ER_NUM) = ON THEN
  . .   DO;
  . .   . FIND SUBAREA_ROUTING IN SUBAREA_ROUTING_LIST
  . .   .      WHERE(SUBAREA_ROUTING.DEST_SA = NC_ER_INOP_RQ.SA(MOVE_TO));
  . .   .
  . .   . /* IF SUBAREA_ROUTING NOT FOUND, ERCB WILL NOT BE FOUND EITHER           */
  . .   . FIND ERCB IN ERCB_LIST
  . .   .      WHERE(ERCB.PARTNER_SA = NC_ER_INOP_RQ.SA(MOVE_TO) &
  . .   .      ERCB.ER_NUM = ER_NUM);
  . .   . IF ERCB_PTR ¬= NULL THEN
  . .   .   DO;
  . .   .   . FIND PATHCB IN PATHCB_LIST WHERE(PATHCB.TG_ID = TGCB.TG_ID);
  . .   .   . IF PATHCB_PTR ¬= NULL THEN
  . .   .   .   DO;
  . .   .   .   . IF (SUBAREA_ROUTING.TG_ID(ERCB.ER_NUM) ¬= TGCB.TG_ID) &
  . .   .   .   .      (FSM_ERN ¬= CONTEND  |  ARE_ANY_PATHS_PENDING) THEN
  . .   .   .   .                           /* PAGE 12-73, 12-72  */
  . .   .   .   .   NC_ER_INOP_RQ.MASK(MOVE_TO,ER_NUM:ER_NUM) = OFF;
  . .   .   .   . CALL FSM_ERN;                         /* PAGE 12-73                */
  . .   .   .   . CALL FSM_PATH;                        /* PAGE 12-75                */
  . .   .   .   END;
  . .   .   END;
  . .   .
  . .   END;
  . .
  . END;
  END;

  NC_ER_INOP_RQ.CNT_ER_FIELD = MOVE_TO - 1;

                                                                                    /*
                  ┌──────────────────────────────────────────┐
                  │ NC_ER_INOP IS PROPAGATED IF THERE IS AT LEAST │
                  │ ONE MASK ENTRY THAT CONTAINS USEFUL          │
                  │ INFORMATION.                                 │
                  └──────────────────────────────────────────┘
                                                                                    */
  IF NC_ER_INOP_RQ.CNT_ER_FIELD ¬= 0 THEN
    DO;
    . IF NCB.INTERMEDIATE_FUNCTION = YES THEN           /* APPENDIX A                */
    .   CALL FANOUT_PROP;                               /* PAGE 12-46                */
    . CALL NS_ER_INOP_SEND;                             /* PAGE 12-47                */
    . CALL VRMGR_INOP_SEND;                             /* PAGE 12-48                */
    END;

  DISCARD MU;

  RETURN;
END INOP_RCV;



LSA_RCV: PROCEDURE;

                                                                                    /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO RECEIVE AN LSA FROM A SUBAREA NODE THAT DOES NOT SUPPORT ER-VR │
  │                 PROTOCOLS.  THE LSA IS CONVERTED TO AN NC_ER_INOP, WHICH IS │
  │                 PROCESSED LIKE AN NC_ER_INOP RECEIVED FROM A NODE THAT DOES SUPPORT │
  │                 ER-VR PROTOCOLS.                                         │
  │                                                                          │
  │      INPUT:     LSA AND TGCB_PTR (INDICATING OVER WHICH TG THE REQUEST WAS RECEIVED) │
  │                                                                          │
  │      OUTPUT:    NC_ER_INOP AND TGCB_PTR                                  │
  │                                                                          │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
  │                 ER_MGR                          PAGE 12-31               │
  │                                                                          │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
  │                 INOP_RCV                        PAGE 12-44               │
  │                 UPM_CHANGE_LSA_TO_INOP          PAGE 12-46               │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                                    */
  CALL UPM_CHANGE_LSA_TO_INOP;                          /* PAGE 12-46                */

  CALL INOP_RCV;                                        /* PAGE 12-44                */

  RETURN;
END LSA_RCV;
```

```
                                                                          /*
 ┌──────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO CONVERT AN LSA  RECEIVED FROM A NODE THAT DOES  NOT SUPPORT ER-VR │
 │                 PROTOCOLS TO AN NC_ER_INOP.  THE  NC_ER_OP.MASK FIELDS INDICATE THAT │
 │                 ONLY ERN 0 IS OPERATIVE TO THE DSA'S SPECIFIED IN THE LSA.  │
 │                                                                          │
 │      INPUT:     LSA                                                      │
 │                                                                          │
 │      OUTPUT:    NC_ER_INOP (ADDRESSED BY MU_PTR); THE RECEIVED LSA IS DISCARDED. │
 │                                                                          │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
 │                 LSA_RCV                          PAGE 12-45              │
 └──────────────────────────────────────────────────────────────────────────┘
                                                                          */

  RETURN;
END UPM_CHANGE_LSA_TO_INOP;



FANOUT_PROP: PROCEDURE;

                                                                          /*
 ┌──────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO PROPAGATE A REQUEST OVER ALL TRANSMISSION GROUPS TO EACH ADJACENT │
 │                 SUBAREA  NODE EXCEPT  TO THE  SUBAREA NODE  REFERENCED BY  TGCB_PTR. │
 │                 NC_ER_OP IS NOT SENT  TO AN ADJACENT SUBAREA NODE IF  THAT NODE DOES │
 │                 NOT SUPPORT ER-VR PROTOCOLS.  NC_ER_INOP IS  CONVERTED TO AN LSA AND │
 │                 SENT TO AN ADJACENT SUBAREA NODE IF THAT NODE DOES NOT SUPPORT ER-VR │
 │                 PROTOCOLS.                                               │
 │                                                                          │
 │      INPUT:     EITHER NC_ER_OP OR  NC_ER_INOP AND TGCB_PTR (INDICATING  THE TG THAT │
 │                 CHANGED STATUS OR THE TG OVER WHICH THE REQUEST WAS RECEIVED) │
 │                                                                          │
 │      OUTPUT:    NC_ER_INOP OR  LSA, OR  NC_ER_OP TO  MULTIPLE PC.TGC'S  (CHAPTER 3). │
 │                 THE RECEIVED REQUEST IS NOT DISCARDED.                   │
 │                                                                          │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
 │                 INOP_RCV                         PAGE 12-44              │
 │                 INOP_SEND                        PAGE 12-42              │
 │                 OP_RCV                           PAGE 12-40              │
 │                                                                          │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
 │                 UPM_CREATE_LSA_FROM_INOP         PAGE 12-47              │
 └──────────────────────────────────────────────────────────────────────────┘
                                                                          */
  DCL COPY_MU_PTR PTR;                    /* NEW COPY OF REQUEST FOR EACH TG  */
  DCL SAVED_TGCB_PTR PTR;                                 /* SAVE TGCB_PTR  */

  SAVED_TGCB_PTR = TGCB_PTR;

                                                                          /*
                 ┌──────────────────────────────────────────┐
                 │ PROPAGATE NC_ER_OP OR NC_ER_INOP  ON EVERY TG │
                 │ FROM THIS SUBAREA NODE  EXCEPT TO THE SUBAREA │
                 │ NODE REFERENCED BY TGCB_PTR.  IF THE ADJACENT │
                 │ SUBAREA   NODE   DOES   NOT   SUPPORT   ER-VR │
                 │ PROTOCOLS, CONVERT AN NC_ER_INOP TO AN LSA.   │
                 └──────────────────────────────────────────┘
                                                                          */
  SCAN TGCP_LIST PTR(TGCB_PTR);
  . IF TGCB.ADJ_SA ¬= SAVED_TGCB_PTR->TGCB.ADJ_SA THEN
  .   DO;
  .   . IF TGCB.ER_VR_SUPP = NO THEN
  .   .   DO;                             /* ER-VR PROTOCOLS NOT SUPPORTED  */
  .   .   . IF RQ_CODE = NC_ER_INOP THEN
  .   .   .   DO;                         /* CONVERT NC_ER_INOP TO LSA       */
  .   .   .   . COPY_MU_PTR = UPM_CREATE_LSA_FROM_INOP;        /* PAGE 12-47  */
  .   .   .   . SEND COPY_MU_PTR->MU TO PC.TGC.LIST_BY_PRTY;   /* CHAPTER 3   */
  .   .   .   END;
  .   .   END;                            /* SEND NOTHING FOR NC_ER_OP       */
  .   . ELSE
  .   .   DO;                             /* ER-VR PROTOCOLS ARE SUPPORTED   */
  .   .   . CREATE COPY_MU_PTR->MU;
  .   .   . COPY_MU_PTR->MU = MU;                          /* COPY ALL FIELDS OF MU */
  .   .   . COPY_MU_PTR->OSAF = NCB.NODE_SUBAREA_ADDRESS;  /* APPENDIX A      */
  .   .   . COPY_MU_PTR->DSAF = TGCB.ADJ_SA;
  .   .   . COPY_MU_PTR->MUCB.DIRECTION = SEND;
  .   .   . SEND COPY_MU_PTR->MU TO PC.TGC.LIST_BY_PRTY;   /* CHAPTER 3       */
  .   .   END;
  .   END;
  SCANEND;

  TGCB_PTR = SAVED_TGCB_PTR;

  RETURN;
END FANOUT_PROP;
```

```
UPM_CREATE_LSA_FROM_INOP: PROCEDURE RETURNS(PTR);

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO CREATE AN LSA FROM AN NC_ER_INOP                          │
  │                                                                              │
  │      INPUT:     NC_ER_INOP                                                   │
  │                                                                              │
  │      OUTPUT:    LSA ADDRESSED BY RETURNED POINTER, NC_ER_INOP ADDRESSED BY MU_PTR
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 FANOUT_PROP                        PAGE 12-46                │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL LSA_MU_PTR PTR;

  RETURN(LSA_MU_PTR);
END UPM_CREATE_LSA_FROM_INOP;



NS_ER_INOP_SEND: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  OPTIONALLY TO BUILD AND SEND ER_INOP FOR EACH CP-PU SESSION IN WHICH
  │                 SDT HAS FLOWED                                               │
  │                                                                              │
  │      INPUT:     NC_ER_INOP                                                   │
  │                                                                              │
  │      OUTPUT:    ER_INOP OR NS_LSA TO SNS; SCB_PTR, CPCB_PTR, AND CP_INDIRECT_PTR ARE
  │                 CHANGED.                                                     │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 INOP_RCV                           PAGE 12-44                │
  │                 INOP_SEND                          PAGE 12-42                │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 BUILD_NS_RQN_RH                    PAGE 12-124               │
  │                 UPM_CREATE_NS_LSA_FROM_INOP        PAGE 12-48                │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL ER_INOP_MU_PTR PTR;                                   /* PTR TO ER_INOP RU  */
  DCL NS_LSA_MU_PTR PTR;                                    /* PTR TO NS_LSA RU   */

  SCAN NRCB.CP_INDIRECT_LIST PTR(CP_INDIRECT_PTR);
  . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;                    /* APPENDIX A         */
  . IF FSM_CP_SESS_SDT = ACTIVE THEN                        /* CHAPTER 11         */
  .   DO;
  .   . SCB_PTR = CPCB.CP_SCB_ID;                           /* APPENDIX A         */
  .   .
  .   . IF CPCB.NS_LSA_RQD = NO THEN                       /* APPENDIX A         */
  .   .   DO;                              /* ER-VR PROTOCOLS NOT SUPPORTED       */
  .   .   . CREATE ER_INOP_MU_PTR->MU;
  .   .   . ER_INOP_MU_PTR->MU = MU;
  .   .   . ER_INOP_MU_PTR->ER_INOP_RQ = NC_ER_INOP_RQ,BY NAME;
  .   .   . ER_INOP_MU_PTR->ER_INOP_RQ.NS_HEADER = ER_INOP_HDR;
  .   .   . CALL BUILD_NS_RQN_RH(ER_INOP_MU_PTR);          /* PAGE 12-124        */
  .   .   . SEND ER_INOP_MU_PTR->MU TO SNS.SEND;           /* OPTIONAL, CHAPTER 6*/
  .   .   END;
  .   . ELSE                               /* ER-VR PROTOCOLS NOT SUPPORTED       */
  .   .   DO;
  .   .   . NS_LSA_MU_PTR = UPM_CREATE_NS_LSA_FROM_INOP;   /* PAGE 12-48         */
  .   .   . SEND NS_LSA_MU_PTR->MU TO SNS.SEND;            /* OPTIONAL, CHAPTER 6*/
  .   .   END;
  .   END;
  SCANEND;

  RETURN;
END NS_ER_INOP_SEND;
```

```
UPM_CREATE_NS_LSA_FROM_INOP: PROCEDURE RETURNS(PTR);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO GENERATE AN NS_LSA FROM AN NC_ER_INOP                  │
   │                                                                          │
   │      INPUT:     NC_ER_INOP                                               │
   │                                                                          │
   │      OUTPUT:    NS_LSA ADDRESSED  BY RETURNED POINTER,  NC_ER_OP STILL  ADDRESSED BY
   │                 MU_PTR                                                   │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 NS_ER_INOP_SEND                  PAGE 12-47              │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL NS_LSA_MU_PTR PTR;

  RETURN(NS_LSA_MU_PTR);
END UPM_CREATE_NS_LSA_FROM_INOP;



VRMGR_INOP_SEND: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  UPON RECEIPT OF  AN NC_ER_INOP, TO SEND  AN ERINOP SIGNAL TO  THE VR
   │                 MANAGER,  SIGNIFYING THAT  CERTAIN ER'S  ARE  NOW INOPERATIVE.   THE
   │                 PARM_ER_INOP STRUCTURE, SENT ALONG WITH  THE SIGNAL, IS GENERATED BY
   │                 COPYING FIELDS FROM NC_ER_INOP.                          │
   │                                                                          │
   │      INPUT:     NC_ER_INOP                                               │
   │                                                                          │
   │      OUTPUT:    ERINOP SIGNAL AND PARM_ER_INOP STRUCTURE TO VR_MGR        │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 INOP_RCV                         PAGE 12-44              │
   │                 INOP_SEND                        PAGE 12-42              │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  CREATE PARM_ER_INOP;                                    /* PAGE 12-127      */
  PARM_ER_INOP = NC_ER_INOP_RQ, BY NAME;                  /* PAGE 12-127      */

  SEND 'ERINOP' TO VR_MGR USING(PARM_PTR = PARM_ER_INOP_PTR);    /* PAGE 12-79, 12-127  */

  RETURN;
END VRMGR_INOP_SEND;
```

EXPLICIT ROUTE ACTIVATION AND TESTING

The explicit route activation and testing protocols use the sequential propagation flow (see the section, "Request Flows"). These protocols test the correctness of the routing tables in the nodes along an explicit route and therefore must flow from one ER manager to the next along the explicit route.

EXPLICIT ROUTE ACTIVATE (NC_ER_ACT)
EXPLICIT ROUTE ACTIVATE REPLY (NC_ER_ACT_REPLY)

Flow: ER manager to ER manager (Expedited), with TG Sweep = SWEEP, at low transmission priority

Principal FSMs: FSM_PATH (Page 12-75)
FSM_ERN (Page 12-73)

NC_ER_ACT is sent by the ER manager in a subarea node in order to activate an explicit route. NC_ER_ACT uses the sequential propagation flow to move from the originating node to the first adjacent node along the explicit route, and from there to each successive adjacent node along the explicit route, until it arrives at the destination subarea node.

Each ER manager receiving an NC_ER_ACT checks various conditions to determine if the requested ER can be activated. The SUBAREA_ROUTING_LIST is examined to determine if there exists a reverse ERN back to the origin subarea and if the explicit route number is defined from the current subarea to the ultimate destination of the request. The transmission group used to send the request along the explicit route must be operative. The number (ER_LENGTH) of transmission groups traversed by the request is incremented by 1 and compared to the maximum value (MAX_ER_LENGTH) specified in the request. Unless one of these checks fails, the NC_ER_ACT continues along the ERN towards the destination subarea where it is converted to an NC_ER_ACT_REPLY and returned to the originating subarea node. If the NC_ER_ACT fails, the ER manager detecting the failure generates the NC_ER_ACT_REPLY and sends it back. NC_ER_ACT_REPLY follows the sequential propagation flow on the explicit route, in a direction opposite to that of the corresponding NC_ER_ACT.

If the activation is successful, the node generating the NC_ER_ACT_REPLY puts the FSM for the (DSA, ERN) used by the NC_ER_ACT_REPLY into a state indicating that a virtual route can be activated on the explicit route. Only the node that sent the NC_ER_ACT, not the one that received it, can initiate activation of a virtual route using the explicit route.

The NC_ER_ACT originating ER manager, upon receiving the NC_ER_ACT_REPLY, sets the state of the FSM for the NC_ER_ACT that caused the NC_ER_ACT_REPLY to be generated. If the activation is unsuccessful, the FSM for the ER enters the operative state and, if necessary, the failure is reported to the VR manager. If the activation is successful, information such as the number of transmission groups contained in the ER and the set of reverse explicit route numbers is transferred to the ERCB. (In the case where

multiple NC_ER_ACT requests were sent to resolve ambiguity caused by the dynamic route definition capability, the first successful NC_ER_ACT_REPLY is used to define which sequence of TGs should become the defined route (as specified in the SUBAREA_ROUTING_LIST) and to fill in the appropriate information in the ERCB.)

The NC_ER_ACT and NC_ER_ACT_REPLY requests carry an identifier (ACT_SEQ_ID) that is unique for each activation attempt. This identifier, which is saved in the PATHCB when an NC_ER_ACT is sent, allows the ER manager to distinguish between multiple attempts to activate the same ER. This situation may occur, for example, when an NC_ER_ACT is sent to activate an ER, but a TG along that ER becomes inoperative and then operative. These changes invalidate the outstanding NC_ER_ACT; after the ER again becomes operative, another NC_ER_ACT can be sent. If the first NC_ER_ACT returns, it is ignored because the route properties it carries (e.g., reverse ERNs) might be incorrect; the ACT_SEQ_ID enables the ER manager to differentiate the activation requests.

If a VR manager request had initiated the ER activation process or the VR manager had requested an ER that was in the process of being activated (i.e., to resolve a routing definition ambiguity), then the result of the activation process is signaled to the VR manager.

ROUTE TEST (ROUTE_TEST)

Flow: SSCP to PU (Normal)

Principal FSMs: FSM_PATH (Page 12-75)
FSM_ERN (Page 12-73)

The ER manager checks FSM_ERN and FSM_PATH for the states of explicit routes specified in the ROUTE_MASK field of ROUTE_TEST, and reports these states (e.g., active, operative and defined, pending activation) in the response to ROUTE_TEST. The ER manager sends NC_ER_TEST requests along the explicit route if the TEST_CODE field of the ROUTE_TEST so specifies.

EXPLICIT ROUTE TEST (NC_ER_TEST)
EXPLICIT ROUTE TEST REPLY (NC_ER_TEST_REPLY)

Flow: ER manager to ER manager (Expedited), with TG Sweep = ¬SWEEP, at low transmission priority

Principal FSMs: FSM_PATH (Page 12-75)
FSM_ERN (Page 12-73)

NC_ER_TEST is sent by a subarea node that requires testing of an explicit route to a specified destination subarea. The test is initiated upon receiving an ROUTE_TEST from the VR manager. Like NC_ER_ACT, NC_ER_TEST flows using sequential propagation.

Each ER manager receiving an NC_ER_TEST along the explicit route makes the same set of checks as it does for an NC_ER_ACT. The SUBAREA_ROUTING_LIST is examined to determine if there exists a reverse ERN back to the origin subarea and if the explicit route number is defined from the current subarea to the ultimate destination of the request. The transmission group used to send the request along the explicit route must be operative. The number (ER_LENGTH) of transmission groups traversed by the request is incremented by 1 and compared to the maximum value (MAX_ER_LENGTH) specified in the request. Unless one of these checks fails, the NC_ER_TEST continues along the ER towards the destination subarea, where it is converted to an NC_ER_TEST_REPLY and returned to the originating subarea node. If the NC_ER_TEST fails, the ER manager detecting the failure generates the NC_ER_TEST_REPLY and sends it back. NC_ER_TEST_REPLY follows the sequential propagation flow on the explicit route, in a direction opposite to that of the corresponding NC_ER_TEST until it reaches the node that originated the NC_ER_TEST. If a failure occurs, an ER_TESTED is generated and sent for each SSCP-PU session in which SDT has flowed.

Upon receiving an NC_ER_TEST_REPLY the ER manager converts it to an ER_TESTED, which is sent to the SSCP that initiated the testing process.

EXPLICIT ROUTE TESTED (ER_TESTED)

Flow: ER manager to SSCP (Normal)

Principal FSMs: None

An ER_TESTED is sent by a subarea node to one or more SSCPs to provide the the status of an ER as follows:

- when an NC_ER_TEST fails the ER manager detecting the failure sends an ER_TESTED for each SSCP-PU session in which SDT has flowed.
- when an NC_ER_TEST_REPLY reaches its destination (i.e., the originator of the NC_ER_TEST), the ER manager at that node sends an ER_TESTED to the SSCP that originated the ROUTE_TEST.

```
ACT_SEND: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │     FUNCTION:  TO ACTIVATE THE EXPLICIT ROUTE SUPPORTING THE VIRTUAL ROUTE │
 │                SPECIFIED BY THE VR MANAGER.  IF THE ER IS ALREADY ACTIVE OR IS NOT │
 │                ABLE TO BE ACTIVATED, SIGNAL ACCORDINGLY (ER_ACTIVATED OR │
 │                ER_NOT_ACTIVATED RESPECTIVELY) TO THE VR MANAGER; OTHERWISE, TRY TO │
 │                ACTIVATE THE ER BY SENDING AN NC_ER_ACT TO THE OTHER END OF THE ER │
 │                USING SEQUENTIAL PROPAGATION.                         │
 │                                                                      │
 │     INPUT:     PARM_ACT_ER ADDRESSED BY PARM_PTR                     │
 │                                                                      │
 │     OUTPUT:    NC_ER_ACT TO PC.TGC OR SIGNAL (ER_ACTIVATED OR ER_NOT_ACTIVATED) TO │
 │                VR_MGR                                                │
 │                                                                      │
 │     NOTE:      PARM_PTR ADDRESSES THE PARM_ACT_ER ENTITY WHEN THIS PROCEDURE │
 │                STARTS, AND THAT SAME ENTITY IS RETURNED TO VR_MGR WHEN THE │
 │                PROCEDURE STOPS.                                      │
 │                                                                      │
 │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                        │
 │                ER_MGR                          PAGE 12-31            │
 │                                                                      │
 │     REFERS TO THE FOLLOWING PROCEDURE(S):                            │
 │                FSM_ERN                         PAGE 12-73            │
 │                VRN_TO_ERN_MAP                  PAGE 12-124           │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL ER_NUM BIT(4);
  DCL VR_NUM BIT(4);

  PARM_ACT_ER_PTR = PARM_PTR;                                  /* PAGE 12-126   */
  VR_NUM = INDEX(PARM_ACT_ER.VRN_MASK,ON);                     /* PAGE 12-126   */

  IF VRN_TO_ERN_MAP(PARM_ACT_ER.PARTNER_SA,VR_NUM,ER_NUM) = ¬EXIST THEN /* PAGE 12-124 */
                                  /* NO VR TO ER MAPPING IS DEFINED    */
    SEND 'ER_NOT_ACTIVATED' TO VR_MGR USING(PARM_PTR = PARM_ACT_ER_PTR);  /* NOTE */

  ELSE
    DO;
    . FIND SUBAREA_ROUTING IN SUBAREA_ROUTING_LIST
    .      WHERE(SUBAREA_ROUTING.DEST_SA = PARM_ACT_ER.PARTNER_SA);  /* PAGE 12-126 */
    . FIND ERCB IN ERCB_LIST
    .      WHERE(ERCB.PARTNER_SA = PARM_ACT_ER.PARTNER_SA &      /* PAGE 12-126   */
    .      ERCB.ER_NUM = ER_NUM);
    . IF SUBAREA_ROUTING_PTR = NULL | ERCB_PTR = NULL THEN
    .   SEND 'ER_NOT_ACTIVATED' TO VR_MGR USING(PARM_PTR = PARM_ACT_ER_PTR); /* PAGE 12-79 */
    . ELSE
    .   DO;
    .   . FIND PATHCB IN PATHCB_LIST WHERE(PATHCB.TG_ID = SUBAREA_ROUTING.TG_ID(ER_NUM));
    .   . CALL FSM_ERN('ACTIVATE_ER');                          /* PAGE 12-73    */
    .   END;
    END;

  RETURN;
END ACT_SEND;
```

```
TEST_SEND: PROCEDURE;

                                                                                /*
┌──────────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO  SEND  AN  NC_ER_TEST  FOR  EACH  (DSA,  ERN)  SPECIFIED  IN  THE  │
│                 ROUTE_TEST  RECEIVED  FROM  THE  VR  MANAGER.  IF  THE  ROUTE_TEST  │
│                 SPECIFIES TESTING OF VR'S, THE DEFINED ER SUPPORTING THAT VR WILL BE  │
│                 TESTED  (DEPENDING ON  THE  TEST_CODE  SETTING); IF  THE  ROUTE_TEST  │
│                 SPECIFIES TESTING  OF ERS, BOTH  THE DEFINED  TG AND ANY  OTHER TG'S  │
│                 THAT HAVE BECOME  KNOWN BECAUSE OF NC_ER_OP'S WILL  BE TESTED (AGAIN  │
│                 DEPENDING ON THE TEST_CODE SETTING).                          │
│                                                                               │
│      INPUT:     ROUTE_TEST                                                    │
│                                                                               │
│      OUTPUT:    NC_ER_TEST REQUESTS TO PC.TGC                                 │
│                                                                               │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                │
│                 ER_MGR                          PAGE 12-31                    │
│                 ROUTE_TEST_RCV                  PAGE 12-113                   │
│                                                                               │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                                    │
│                 ACT_TEST_SEND                   PAGE 12-59                    │
│                 BUILD_NC_ER_ACT_OR_TEST         PAGE 12-68                    │
│                 UPM_TEST_CODE_FORCES_SEND       PAGE 12-57                    │
│                 VRN_TO_ERN_MAP                  PAGE 12-124                   │
└──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
DCL ER_NUM BIT(4);                                          /* ERN TO TEST     */
DCL VR_NUM BIT(4);                      /* VRN BEING TESTED   */
DCL MASK_INDEX BIT(4);                  /* INDEXES THROUGH BITS IN ROUTE_MASK  */
DCL RC BIT(1);                          /* UNUSED RETURN CODE WHEN MAPPING VRN TO ERN */

FIND SUBAREA_ROUTING IN SUBAREA_ROUTING_LIST
     WHERE(SUBAREA_ROUTING.DEST_SA = ROUTE_TEST_RQ.DESTINATION_SA);

DO MASK_INDEX = 0 TO NCB.MAX_ER_NUM;                        /* APPENDIX A      */
. IF ROUTE_TEST_RQ.ROUTE_MASK(MASK_INDEX:MASK_INDEX) = ON THEN
.   SELECT ANYORDER(ROUTE_TEST_RQ.TEST_TYPE);
                                                                                /*
                ┌──────────────────────────────────────────┐
                │  WHEN TESTING VR'S, SEND  NC_ER_TEST ALONG THE  │
                │  DEFINED  TG'S  FOR THE  ER'S  SUPPORTING  THE  │
                │  VR'S.                                   │
                └──────────────────────────────────────────┘
                                                                                */
.   . WHEN(TEST_VRS)
.   .   DO;                             /* CONVERT VRN TO ERN TO BE TESTED     */
.   .   . IF VRN_TO_ERN_MAP(ROUTE_TEST_RQ.DESTINATION_SA,VR_NUM,ER_NUM) = EXIST THEN
.   .   .                                                   /* PAGE 12-124     */
.   .   .   DO;
.   .   .   . FIND ERCB IN ERCB_LIST
.   .   .   .       WHERE(ERCB.PARTNER_SA = ROUTE_TEST_RQ.DESTINATION_SA &
.   .   .   .       ERCB.ER_NUM = ER_NUM);
.   .   .   .
.   .   .   . IF ERCB_PTR = NULL | UPM_TEST_CODE_FORCES_SEND = YES THEN     /* PAGE 12-57  */
.   .   .   .   DO;
.   .   .   .   . FIND TGCB IN TGCB_LIST WHERE(TGCB.TG_ID = SUBAREA_ROUTING.TG_ID(ER_NUM));
.   .   .   .   . MU_PTR = BUILD_NC_ER_ACT_OR_TEST('TEST');    /* PAGE 12-68      */
.   .   .   .   . CALL ACT_TEST_SEND;                          /* PAGE 12-59      */
.   .   .   .   END;
.   .   .   END;
.   .   END;
.   .
                                                                                /*
                ┌──────────────────────────────────────────┐
                │  WHEN TESTING  ER'S, SEND NC_ER_TEST  OVER ALL  │
                │  TG'S  THAT  ARE  RELATED  TO  THE  ER'S  (AS  │
                │  RECORDED IN THE PATHCB_LIST).           │
                └──────────────────────────────────────────┘
                                                                                */
.   . WHEN(TEST_ERS)
.   .   DO;
.   .   . ER_NUM = MASK_INDEX;
.   .   . FIND ERCB IN ERCB_LIST
.   .   .       WHERE(ERCB.PARTNER_SA = ROUTE_TEST_RQ.DESTINATION_SA &
.   .   .       ERCB.ER_NUM = ER_NUM);
.   .   .
.   .   . IF ERCB_PTR ¬= NULL THEN
.   .   .   SCAN PATHCB_LIST PTR(PATHCB_PTR);
.   .   .   . IF UPM_TEST_CODE_FORCES_SEND = YES THEN          /* PAGE 12-57      */
.   .   .   .   DO;
.   .   .   .   . FIND TGCB IN TGCB_LIST WHERE(TGCB.TG_ID = PATHCB.TG_ID);
.   .   .   .   . MU_PTR = BUILD_NC_ER_ACT_OR_TEST('TEST');    /* PAGE 12-68      */
.   .   .   .   . CALL ACT_TEST_SEND;                          /* PAGE 12-59      */
.   .   .   .   END;
.   .   .   SCANEND;
.   .   END;
```

```
                                                                              /*
                    ┌──────────────────────────────────────────┐
                    │ WHEN TESTING THE DEFINED TG FOR ER'S, SEND │
                    │ NC_ER_TEST ONLY OVER THE TG DEFINED FOR THE │
                    │ ER'S (AS RECORDED IN THE │
                    │ SUBAREA_ROUTING_LIST). │
                    └──────────────────────────────────────────┘
                                                                              */
   . . WHEN(TEST_DEFINED_ERS)
   . .   DO;
   . .   . ER_NUM = MASK_INDEX;
   . .   . FIND ERCB IN ERCB_LIST
   . .   .      WHERE(ERCB.PARTNER_SA = ROUTE_TEST_RQ.DESTINATION_SA &
   . .   .      ERCB.ER_NUM = ER_NUM);
   . .   .
   . .   . IF ERCB_PTR = NULL | UPM_TEST_CODE_FORCES_SEND = YES THEN    /* PAGE 12-57      */
   . .   .   DO;
   . .   .   . FIND TGCB IN TGCB_LIST WHERE(TGCB.TG_ID = SUBAREA_ROUTING.TG_ID(ER_NUM));
   . .   .   . MU_PTR = BUILD_NC_ER_ACT_OR_TEST('TEST');          /* PAGE 12-68      */
   . .   .   . CALL ACT_TEST_SEND;                                /* PAGE 12-59      */
   . .   .   END;
   . .   END;
   . .
   . END;
  END;

  RETURN;
END TEST_SEND;



UPM_TEST_CODE_FORCES_SEND: PROCEDURE RETURNS(BIT(1));
                                                                              /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO DETERMINE WHETHER AN NC_ER_TEST SHOULD BE SENT INTO THE NETWORK │
  │                 TO EXPLICITLY TEST AN ER. THIS DECISION DEPENDS ON THE STATUS OF │
  │                 THE (DSA, ERN) AND THE SETTING OF ROUTE_TEST_RQ.TEST_CODE │
  │                                                                          │
  │      INPUT:     ERCB_PTR AND ROUTE_TEST                                  │
  │                                                                          │
  │      OUTPUT:    BOOLEAN VALUE INDICATING WHETHER THE NC_ER_TEST SHOULD BE SENT (YES) │
  │                 OR NOT (NO)                                              │
  │                                                                          │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
  │                 TEST_SEND                          PAGE 12-56            │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                              */

  RETURN(YES);
END UPM_TEST_CODE_FORCES_SEND;
```

This page intentionally left blank

```
ACT_TEST_SEND: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO  SEND AN  NC_ER_TEST OR  NC_ER_ACT ON  A PARTICULAR  TG.  IF  THE │
   │                 REQUIRED  TG IS  NOT DEFINED  (I.E., TGCB_PTR  IS NULL),  THE TG  IS │
   │                 INOPERATIVE (AN EMPTY ASSOC_LSCB_LIST), OR THE ADJACENT SUBAREA NODE │
   │                 DOES  NOT  SUPPORT  ER-VR PROTOCOLS,  THEN  AN  NC_ER_TEST_REPLY  OR │
   │                 NC_ER_ACT_REPLY REQUEST IS  RETURNED TO THE ORIGINATOR  OF THE INPUT │
   │                 REQUEST.  IF THE INPUT REQUEST IS AN NC_ER_TEST AND ONE OF THE ABOVE │
   │                 FAILURES IS DETECTED, AN ER_TESTED IS  SENT FOR EACH SSCP-PU SESSION │
   │                 IN WHICH SDT HAS FLOWED.                             │
   │                                                                      │
   │      INPUT:     NC_ER_TEST OR NC_ER_ACT REQUEST AND  TGCB_PTR (INDICATING OVER WHICH │
   │                 TG THE REQUEST SHOULD BE SENT)                       │
   │                                                                      │
   │      OUTPUT:    DEPENDING ON WHETHER A FAILURE IS DETECTED, EITHER   │
   │                 1.  NC_ER_TEST OR NC_ER_ACT TO PC.TGC (CHAPTER 3) OR │
   │                 2.  NC_ER_TEST_REPLY OR  NC_ER_ACT_REPLY TO PC.TGC (CHAPTER  3), AND │
   │                     ER_TESTED TO SNS (CHAPTER 6)                     │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 ACT_TEST_RCV                        PAGE 12-60       │
   │                 TEST_SEND                           PAGE 12-56       │
   │                                                                      │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
   │                 ACT_TEST_REPLY_SEND                 PAGE 12-62       │
   │                 BUILD_NC_ER_ACT_OR_TEST_REPLY       PAGE 12-70       │
   │                 TESTED_TO_ALL_SSCPS                 PAGE 12-63       │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  SELECT INORDER;
  . WHEN(TGCB_PTR = NULL)
  .   DO;                                                    /* TG NOT DEFINED          */
  .   . CALL BUILD_NC_ER_ACT_OR_TEST_REPLY(ER_NOT_DEFINED);  /* PAGE 12-70              */
  .   . IF RQ = NC_ER_TEST_REPLY THEN
  .   .   CALL TESTED_TO_ALL_SSCPS;                          /* PAGE 12-63              */
  .   . CALL ACT_TEST_REPLY_SEND;                            /* PAGE 12-62              */
  .   END;
  .
  . WHEN(EMPTY(TGCB.ASSOC_LSCB_LIST) = YES)
  .   DO;                                                    /* TG NOT ACTIVE           */
  .   . CALL BUILD_NC_ER_ACT_OR_TEST_REPLY(TG_INOPERATIVE);  /* PAGE 12-70              */
  .   . IF RQ = NC_ER_TEST_REPLY THEN
  .   .   CALL TESTED_TO_ALL_SSCPS;                          /* PAGE 12-63              */
  .   . CALL ACT_TEST_REPLY_SEND;                            /* PAGE 12-62              */
  .   END;
  .
  . WHEN(TGCB.ER_VR_SUPP = PRE_ER_VR)
  .   DO;                          /* ADJACENT NODE DOES NOT SUPPORT ER-VR PROTOCOLS */
  .   . CALL BUILD_NC_ER_ACT_OR_TEST_REPLY(PRE_ER_VR_SUPPORT);  /* PAGE 12-70          */
  .   . IF RQ = NC_ER_TEST_REPLY THEN
  .   .   CALL TESTED_TO_ALL_SSCPS;                          /* PAGE 12-63              */
  .   . CALL ACT_TEST_REPLY_SEND;                            /* PAGE 12-62              */
  .   END;
  .
  . OTHERWISE
  .   DO;                                                    /* PROPAGATE THE RU        */
  .   . OSAF = NCB.NODE_SUBAREA_ADDRESS;                     /* APPENDIX A              */
  .   . DSAF = SUBAREA_ROUTING.ADJ_SA(TGCB.ADJ_SA);
  .   . SEND RU TO PC.TGC.LIST_BY_PRTY;                      /* CHAPTER 3               */
  .   END;
  END;

  RETURN;
END ACT_TEST_SEND;
```

```
ACT_TEST_RCV: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO RECEIVE AND PROCESS NC_ER_ACT AND NC_ER_TEST BY       │
│                 A) IF NOT DESTINED FOR THIS NODE, FORWARDING THE REQUEST  │
│                    TOWARDS ITS DESTINATION                               │
│                 B) IF DESTINED FOR THIS NODE OR AN ERROR IS DETECTED      │
│                    (E.G., NO REVERSE ERN AVAILABLE, MAXIMUM ER LENGTH     │
│                    EXCEEDED), BUILDING APPROPRIATE NC_ER_ACT_REPLY OR     │
│                    NC_ER_TEST_REPLY REQUEST AND SENDING IT BACK TO THE    │
│                    ORIGINATING NODE                                       │
│                                                                           │
│      INPUT:     NC_ER_ACT OR NC_ER_TEST                                   │
│                                                                           │
│      OUTPUT:    FOR (A): NC_ER_ACT OR NC_ER_TEST                          │
│                 FOR (B):                                                  │
│                       • NC_ER_ACT_REPLY OR NC_ER_TEST_REPLY               │
│                       • ER_TESTED TO SSCP'S (IF INPUT REQUEST IS          │
│                         NC_ER_TEST)                                       │
│                       • DISCARDED NC_ER_ACT OR NC_ER_TEST                 │
│                                                                           │
│      NOTES:     1. OPTIONALLY, RATHER THAN EXAMINE ALL POSSIBLE REVERSE   │
│                    ERN'S, THIS LOOP MAY STOP PROCESSING ERN'S AFTER THE   │
│                    FSM FOR ONE OF THE ERN'S ENTERS A STATE THAT ALLOWS    │
│                    TRAFFIC ON THE ER. IF THIS OPTION IS FOLLOWED, THE     │
│                    BITS IN THE REV_ERN_MASK OF THE NC_ER_ACT              │
│                    REPRESENTING ALL OTHER RERN'S ARE SET OFF. THE SET OF  │
│                    RERN'S RETURNED IN THE NC_ER_ACT_REPLY IS ECHOED IN    │
│                    THE NC_ACTVR REQUEST THAT ACTIVATES A VR. IF ALL       │
│                    RERN'S ARE EXAMINED, WHEN THE NODE'S VR MANAGER        │
│                    RECEIVES AN NC_ACTVR, IT CAN VERIFY THAT THE NODE'S    │
│                    VR TO ER MAPPING IS SATISFIED BY THE ER OVER WHICH     │
│                    THE NC_ACTVR IS RECEIVED. IF ALL RERN'S ARE NOT        │
│                    EXAMINED, THE NC_ACTVR RECEIVER MUST ACCEPT THE ER     │
│                    OVER WHICH THE VR IS BEING ACTIVATED, REGARDLESS OF    │
│                    ITS VR TO ER MAPPING.                                  │
│                                                                           │
│                 2. THE ER_RACE TYPE ARISES WHEN A NODE ALLOWS DYNAMIC     │
│                    ROUTE DEFINITION. FOR THIS TYPE OF FAILURE, THE        │
│                    REVERSIBILITY CHECKS ARE FIRST PASSED SUCCESSFULLY.    │
│                    HOWEVER, ALL THE REV_ERN_MASK BITS IN THE NC_ER_ACT    │
│                    ARE SET OFF BECAUSE THIS NODE IS INVOLVED IN A         │
│                    DYNAMIC ROUTE DEFINITION RACE WITH ANOTHER NODE. WHEN  │
│                    TWO NODES ARE BOTH IN THE PROCESS OF DYNAMICALLY       │
│                    DEFINING EXPLICIT ROUTES, THE NODE WITH THE SMALLER    │
│                    SUBAREA ADDRESS LOSES THE CONTENTION AND ACCEPTS THE   │
│                    NC_ER_ACT REQUESTS FROM THE NODE WITH THE LARGER       │
│                    SUBAREA ADDRESS. THE WINNING NODE REJECTS NC_ER_ACT    │
│                    REQUESTS WITH AN ER_RACE CODE.                         │
│                                                                           │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                 ER_MGR                              PAGE 12-31            │
│                                                                           │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                                │
│                 ACT_TEST_REPLY_SEND                 PAGE 12-62            │
│                 ACT_TEST_SEND                       PAGE 12-59            │
│                 BUILD_NC_ER_ACT_OR_TEST_REPLY       PAGE 12-70            │
│                 FSM_ERN                             PAGE 12-73            │
│                 REDUCE_REVERSE_ERN                  PAGE 12-62            │
│                 TESTED_TO_ALL_SSCPS                 PAGE 12-63            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL SAVE_RERN BIT(16);                  /* USED TO SAVE REV_ERN_MASK          */
  DCL RERN_NUM BIT(4);
  DCL 1 NC_ER_ACT_TEST_RQ  LIKE NC_ER_ACT_RQ  BASED(ADDR(RU));

  NC_ER_ACT_TEST_RQ.ER_LENGTH = NC_ER_ACT_TEST_RQ.ER_LENGTH + 1; /* INCREMENT ER LENGTH  */
  SAVE_RERN = NC_ER_ACT_TEST_RQ.REV_ERN_MASK; /* SAVE REVERSE ERN MASK BECAUSE      */
                                          /* IT IS DESTROYED BY REDUCE_REVERSE_ERN  */

  IF NC_ER_ACT_TEST_RQ.ER_LENGTH > NC_ER_ACT_TEST_RQ.MAX_ER_LENGTH THEN /* ER LENGTH ERROR */
    DO;
    . CALL BUILD_NC_ER_ACT_OR_TEST_REPLY(ER_LENGTH_ERROR);       /* PAGE 12-70      */
    . IF RQ = NC_ER_TEST_REPLY THEN
    .   CALL TESTED_TO_ALL_SSCPS;                                /* PAGE 12-63      */
    . CALL ACT_TEST_REPLY_SEND;                                  /* PAGE 12-62      */
    . RETURN;
    END;

  IF REDUCE_REVERSE_ERN =  ¬EXIST THEN                           /* PAGE 12-62      */
    DO;                                 /* REVERSE ERN DOES NOT EXIST FOR THIS ER  */
    . NC_ER_ACT_TEST_RQ.REV_ERN_MASK = SAVE_RERN;                /* RESTORE REVERSE ERN MASK */
    . CALL BUILD_NC_ER_ACT_OR_TEST_REPLY(NO_REVERSE_ERN_DEFINED);    /* PAGE 12-70  */
    . IF RQ = NC_ER_TEST_REPLY THEN
    .   CALL TESTED_TO_ALL_SSCPS;                                /* PAGE 12-63      */
    . CALL ACT_TEST_REPLY_SEND;                                  /* PAGE 12-62      */
    . RETURN;
    END;
```

```
IF NC_ER_ACT_TEST_RQ.DESTINATION_SA = NCB.NODE_SUBAREA_ADDRESS THEN    /* APPENDIX A       */
  DO;                                    /* NC_ER_TEST DESTINED FOR THIS SUBAREA          */
  . IF RQ_CODE = NC_ER_ACT THEN
  .   DO;                                                          /* NC_ER_ACT RECEIVED  */
  .   . DO RER_NUM = 0 TO NCB.MAX_ER_NUM;                         /* NOTE 1, APPENDIX A  */
  .   . . IF NC_ER_ACT_RQ.REV_ERN_MASK(RER_NUM:RER_NUM) = ON THEN
  .   . .   DO;
  .   . .   . FIND ERCB IN ERCB_LIST
  .   . .   .      WHERE(ERCB.PARTNER_SA = NC_ER_ACT_RQ.ORIGINATING_SA &
  .   . .   .      ERCB.ER_NUM = RER_NUM);
  .   . .   . FIND PATHCB IN PATHCB_LIST WHERE(PATHCB.TG_ID = TGCB.TG_ID);
  .   . .   . CALL FSM_ERN;                                       /* PAGE 12-73          */
  .   . .   END;
  .   . END;
  .   . IF NC_ER_ACT_RQ.REV_ERN_MASK = ALL_OFF THEN              /* APPENDIX A          */
  .   .   CALL BUILD_NC_ER_ACT_OR_TEST_REPLY(ER_RACE);           /* NOTE 2, PAGE 12-70  */
  .   . ELSE
  .   .   CALL BUILD_NC_ER_ACT_OR_TEST_REPLY(POSITIVE_REPLY);   /* PAGE 12-70          */
  .   END;
  . ELSE                                                         /* NC_ER_TEST RECEIVED */
  .   CALL BUILD_NC_ER_ACT_OR_TEST_REPLY(POSITIVE_REPLY);       /* PAGE 12-62          */
  . CALL ACT_TEST_REPLY_SEND;                                    /* PAGE 12-62          */
  . RETURN;
  END;

                                                                                          /*
                  ┌──────────────────────────────────────────┐
                  │ TRANSMIT THE REQUEST ON THE NEXT TRANSMISSION │
                  │ GROUP OF THE ER IF POSSIBLE.             │
                  └──────────────────────────────────────────┘
                                                                                          */

  FIND SUBAREA_ROUTING IN SUBAREA_ROUTING_LIST
       WHERE(SUBAREA_ROUTING.DEST_SA = NC_ER_ACT_TEST_RQ.DESTINATION_SA);
  IF SUBAREA_ROUTING_PTR ¬= NULL THEN
    FIND TGCB IN TGCB_LIST WHERE(TGCB.TG_ID = SUBAREA_ROUTING.TG_ID(ERN));
  ELSE
    TGCB_PTR = NULL;                                             /* ER IS NOT DEFINED   */
  CALL ACT_TEST_SEND;                                            /* PAGE 12-59          */

  RETURN;
END ACT_TEST_RCV;
```

```
REDUCE_REVERSE_ERN: PROCEDURE RETURNS(BIT(1));

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO UPDATE THE  REV_ERN_MASK IN AN NC_ER_ACT  OR NC_ER_TEST ACCORDING │
 │                 TO THE ROUTING TABLES AND CHECK TO  SEE IF THERE EXISTS AT LEAST ONE │
 │                 REVERSE  ERN FOR  THE  REQUEST TO  USE WHEN  TRAVELING  BACK TO  ITS │
 │                 ORIGINATION NODE                                             │
 │                                                                              │
 │      INPUT:     NC_ER_ACT OR NC_ER_TEST                                      │
 │                                                                              │
 │      OUTPUT:    UPDATED  REV_ERN_MASK IN  THE NC_ER_ACT  OR  NC_ER_TEST REQUEST  AND │
 │                 RETURNED VALUE  SET TO EXIST OR  ¬EXIST DEPENDING ON WHETHER  OR NOT │
 │                 THE ER IS REVERSIBLE AT THIS NODE                            │
 │                                                                              │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
 │                 ACT_TEST_RCV                        PAGE 12-60               │
 │                                                                              │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
 │                 FSM_ERN                             PAGE 12-73               │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL ER_NUM BIT(4);
  DCL 1 NC_ER_ACT_TEST_RQ  LIKE NC_ER_ACT_RQ BASED(ADDR(RU));

  FIND SUBAREA_ROUTING IN SUBAREA_ROUTING_LIST
       WHERE(SUBAREA_ROUTING.DEST_SA = NC_ER_ACT_TEST_RQ.ORIGINATING_SA);

  IF SUBAREA_ROUTING_PTR = NULL THEN
    NC_ER_ACT_TEST_RQ.REV_ERN_MASK = ALL_OFF;

  ELSE
    DO ER_NUM = 0 TO NCB.MAX_ER_NUM;                          /* APPENDIX A    */
    . IF NC_ER_ACT_TEST_RQ.REV_ERN_MASK(ER_NUM:ER_NUM) = ON THEN
    .   DO;
    .   . FIND ERCB IN ERCB_LIST
    .   .      WHERE(ERCB.PARTNER_SA = NC_ER_ACT_TEST_RQ.ORIGINATING_SA &
    .   .      ERCB.ER_NUM = ER_NUM);
    .   . IF ERCB_PTR = NULL THEN          /* CANNOT OCCUR -- NO OP PREVIOUSLY FOR THIS ER  */
    .   .   NC_ER_ACT_TEST_RQ.REV_ERN_MASK(ER_NUM:ER_NUM) = OFF;
    .   . ELSE
    .   .   DO;
    .   .   . FIND PATHCB IN PATHCB_LIST WHERE(PATHCB.TG_ID = TGCB.TG_ID);
    .   .   . IF (PATHCB_PTR = NULL) |     /* CANNOT OCCUR -- NO OP PREVIOUSLY ON THIS TG_ID */
    .   .   .      (SUBAREA_ROUTING.TG_ID(ER_NUM) ¬= TGCB.TG_ID &
    .   .   .      FSM_ERN ¬= CONTEND) THEN                    /* PAGE 12-73    */
    .   .   .   NC_ER_ACT_TEST_RQ.REV_ERN_MASK(ER_NUM:ER_NUM) = OFF;
    .   .   END;
    .   END;
    END;

  IF NC_ER_ACT_TEST_RQ.REV_ERN_MASK = ALL_OFF THEN
    RETURN(¬EXIST);                    /* NO REVERSE ERN'S DEFINED              */
  ELSE
    RETURN( EXIST);                    /* AT LEAST ONE REVERSE ERN DEFINED      */

END REDUCE_REVERSE_ERN;



ACT_TEST_REPLY_SEND: PROCEDURE;

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO  FINISH  BUILDING   AND  THEN  TO  SEND   AN  NC_ER_ACT_REPLY  OR │
 │                 NC_ER_TEST_REPLY.  IF THE  ORIGINAL REQUEST IS AN  NC_ER_TEST AND IT │
 │                 IS  NOT   ABLE  TO  BE  SENT   OUT  OF  THE  ORIGINATING   NODE,  AN │
 │                 NC_ER_TEST_REPLY WILL  HAVE BEEN GENERATED  AND IS SENT  DIRECTLY TO │
 │                 ER_MGR  (CHAPTER 12),  AGAIN IN THE ORIGINATING NODE,  RATHER THAN TO │
 │                 PC.TGC  (CHAPTER 3).                                         │
 │                                                                              │
 │      INPUT:     PARTIALLY CONSTRUCTED NC_ER_ACT_REPLY OR NC_ER_TEST_REPLY    │
 │                                                                              │
 │      OUTPUT:    NC_ER_ACT_REPLY OR NC_ER_TEST_REPLY TO PC.TGC  (CHAPTER 3) OR ER_MGR │
 │                 (CHAPTER 12)                                                 │
 │                                                                              │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
 │                 ACT_TEST_RCV                        PAGE 12-60               │
 │                 ACT_TEST_SEND                       PAGE 12-59               │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DSAF = TGCB.ADJ_SA;
  IF DSAF = NCB.NODE_SUBAREA_ADDRESS THEN                     /* APPENDIX A    */
    SEND MU TO PC.TGC.LIST_BY_PRTY;                           /* CHAPTER 3     */
  ELSE
    SEND MU TO ER_MGR;                                        /* PAGE 12-31    */

  RETURN;
END ACT_TEST_REPLY_SEND;
```

```
TESTED_TO_ALL_SSCPS: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO SEND ER_TESTED FOR EACH SSCP-PU SESSION IN WHICH SDT HAS FLOWED │
  │                                                                              │
  │      INPUT:     PARTIALLY CONSTRUCTED NC_ER_TEST_REPLY                       │
  │                                                                              │
  │      OUTPUT:    ER_TESTED TO SSCP'S, AND CHANGED SCB_PTR                     │
  │                                                                              │
  │      NOTE:      DO NOT SEND ER_TESTED TO THE PUCP IN THE NODE DETECTING THE  │
  │                 NC_ER_TEST FAILURE.                                          │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 ACT_TEST_RCV                    PAGE 12-60                   │
  │                 ACT_TEST_SEND                   PAGE 12-59                   │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 TESTED_SEND                     PAGE 12-63                   │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  NRCB_PTR = LOCATE_NODE_RESOURCE(NCB.PU_EA);                    /* APPENDIX B    */
  SCAN NRCB.CP_INDIRECT_LIST PTR(CP_INDIRECT_PTR);               /* APPENDIX A    */
  . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;                         /* APPENDIX A    */
  . SCB_PTR = CPCB.CP_SCB_ID;                                    /* APPENDIX A    */
  . IF FSM_CP_SESS_SDT = ACTIVE &                                /* CHAPTER 11    */
  .      SCB.PARTNER_SA ¬= NCB.NODE_SUBAREA_ADDRESS THEN         /* NOTE, APPENDIX A */
  .   CALL TESTED_SEND;                                         /* PAGE 12-63    */
  SCANEND;

  RETURN;
END TESTED_TO_ALL_SSCPS;



TESTED_SEND: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO CREATE AND SEND AN ER_TESTED REQUEST TO THE SSCP REFERENCED BY │
  │                 THE SCB_PTR                                                  │
  │                                                                              │
  │      INPUT:     PARTIALLY CONSTRUCTED NC_ER_TEST_REPLY AND SCB_PTR           │
  │                                                                              │
  │      OUTPUT:    ER_TESTED TO SSCP AND ORIGINAL INPUT REQUEST RETURNED TO CALLING │
  │                 PROCEDURE                                                    │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 ACT_TEST_REPLY_RCV              PAGE 12-64                   │
  │                 TESTED_TO_ALL_SSCPS             PAGE 12-63                   │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 BUILD_NS_RQN_RH                 PAGE 12-124                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL ER_TESTED_MU_PTR PTR;              /* SAVE PTR TO NC_ER_TEST_REPLY       */

  IF SCB_PTR = NULL THEN
    RETURN;

  CREATE ER_TESTED_MU_PTR->MU;
  ER_TESTED_MU_PTR->MU = MU;
  ER_TESTED_MU_PTR->ER_TESTED_RQ = NC_ER_TEST_REPLY_RQ,BY NAME;
  ER_TESTED_MU_PTR->ER_TESTED_RQ.NS_HEADER = ER_TESTED_HDR;
  CALL BUILD_NS_RQN_RH(ER_TESTED_MU_PTR);                        /* PAGE 12-124   */

                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │  THE FOLLOWING STATEMENTS SET UP THE OPTIONAL  │
                  │  FORMAT 2 VERSION OF ER_TESTED.          │
                  └──────────────────────────────────────────┘
                                                                                */

  ER_TESTED_MU_PTR->ER_TESTED_RQ.FORMAT = FORMAT2;
  ER_TESTED_MU_PTR->ER_TESTED_RQ.ORIGINATING_ADJ_SA = TGCB.ADJ_SA;
  ER_TESTED_MU_PTR->ER_TESTED_RQ.ORIGINATING_TGN = TGCB.TGN;


  SEND ER_TESTED_MU_PTR->MU TO SNS.SEND;                         /* CHAPTER 6     */

  RETURN;
END TESTED_SEND;
```

```
ACT_TEST_REPLY_RCV: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO RECEIVE AN  NC_ER_ACT_REPLY OR NC_ER_TEST_REPLY.  IF  THE REQUEST │
 │                 IS NOT DESTINED  FOR THIS NODE, THE SUBAREA_ROUTING_LIST  IS USED TO │
 │                 SEND THE  REQUEST ALONG  THE ER TO  ITS DESTINATION;  OTHERWISE, THE │
 │                 REQUEST IS PROCESSED  AS FOLLOWS.  FOR AN  NC_ER_ACT_REPLY, THE ERCB │
 │                 IS UPDATED AND THE VR MANAGER IS SIGNALED.  FOR AN NC_ER_TEST_REPLY, │
 │                 ER_TESTED IS SENT TO THE SSCP THAT INITIATED THE TEST PROCEDURE.     │
 │                                                                      │
 │      INPUT:     NC_ER_ACT_REPLY OR NC_ER_TEST_REPLY                  │
 │                                                                      │
 │      OUTPUT:    IF THE  REQUEST IS  NOT DESTINED  FOR THIS  SUBAREA, IT  IS SENT  TO │
 │                 PC.TGC  (CHAPTER 3)  TO BE  ROUTED TO  THE  NEXT NODE  ALONG THE  ER │
 │                 TOWARDS  ITS  DESTINATION.  IF  THE  REQUEST  IS DESTINED  FOR  THIS │
 │                 SUBAREA, EITHER ER_TESTED IS SENT TO  AN SSCP OR AN (ER_ACTIVATED OR │
 │                 ER_NOT_ACTIVATED) SIGNAL IS  SENT TO THE VR MANAGER;  THE REQUEST IS │
 │                 DISCARDED.                                           │
 │                                                                      │
 │      NOTES:     1. IF  AN  NC_ER_INOP  FOR  THE  (DSA, ERN)  IS  RECEIVED  AFTER  AN │
 │                    NC_ER_ACT IS  SENT TO ACTIVATE  AN ER,  THE PATHCB AND  MAYBE THE │
 │                    ERCB WILL HAVE BEEN DESTROYED.                    │
 │                                                                      │
 │                 2. FSM_PATH SIGNALS THE VR MANAGER.                  │
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 ER_MGR                          PAGE 12-31           │
 │                                                                      │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
 │                 FSM_ERN                         PAGE 12-73           │
 │                 FSM_PATH                        PAGE 12-75           │
 │                 TESTED_SEND                     PAGE 12-63           │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF NC_ER_ACT_REPLY_RQ.ORIGINATING_SA ¬= NCB.NODE_SUBAREA_ADDRESS THEN /* APPENDIX A      */
    DO;
    . FIND SUBAREA_ROUTING IN SUBAREA_ROUTING_LIST
    .      WHERE(SUBAREA_ROUTING.DEST_SA = NC_ER_ACT_REPLY_RQ.ORIGINATING_SA);
    . IF SUBAREA_ROUTING_PTR ¬= NULL THEN
    .   DO;
    .   . OSAF = NCB.NODE_SUBAREA_ADDRESS;                    /* APPENDIX A          */
    .   . DSAF = SUBAREA_ROUTING.ADJ_SA(NC_ER_ACT_REPLY_RQ.ER_NUM);
    .   . SEND MU TO PC.TGC.LIST_BY_PRTY;                     /* CHAPTER 3           */
    .   END;
    END;

  ELSE                                                        /* FOUND DESTINATION NODE */
    DO;
    . IF RQ_CODE = NC_ER_TEST_REPLY THEN
    .   DO;                                                   /* NC_ER_TEST_REPLY RU */
    .   . SCB_PTR = FIND_SCB_FOR_CP_PU_SESS(NC_ER_TEST_REPLY_RQ.ORIGINATING_SSCP);
                                                              /* APPENDIX B          */
    .   . CALL TESTED_SEND;                                   /* PAGE 12-63          */
    .   END;
    . ELSE
    .   DO;                                                   /* NC_ER_ACT_REPLY RU  */
    .   . FIND ERCB IN ERCB_LIST
    .   .      WHERE(ERCB.PARTNER_SA = NC_ER_ACT_REPLY_RQ.DESTINATION_SA &
    .   .      ERCB.ER_NUM = NC_ER_ACT_REPLY_RQ.ER_NUM);
    .   . IF ERCB_PTR ¬= NULL THEN
    .   .   DO;                                               /* NOTE 1              */
    .   .   . FIND PATHCB IN PATHCB_LIST WHERE(PATHCB.TG_ID = TGCB.TG_ID);
    .   .   . IF PATHCB_PTR ¬= NULL &                         /* NOTE 1              */
    .   .   .      PATHCB.ACT_SEQ_ID = NC_ER_ACT_REPLY_RQ.ACT_SEQ_ID THEN
    .   .   .   DO;
    .   .   .   . CALL FSM_ERN;                               /* PAGE 12-73          */
    .   .   .   . CALL FSM_PATH;                              /* PAGE 12-75, NOTE 2  */
    .   .   .   END;
    .   .   END;
    .   END;
    . DISCARD MU;
    END;

   RETURN;
END ACT_TEST_REPLY_RCV;
```

```
SET_ER: PROCEDURE(DEST_SA,ER_NUM);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO FILL  IN RSP(ROUTE_TEST) WITH  THE STATUS OF  ALL TG'S USED  BY A │
│                 GIVEN (DSA, ERN)  AND THE STATUS OF  THE VR'S THAT ARE  SUPPORTED BY │
│                 THE (DSA, ERN).  WHEN ROUTE_TEST INDICATES AN ER TEST, THE STATUS OF │
│                 ALL TG'S  IS GIVEN.   FOR THE  DEFINED TG,  ONE ROUTE_DATA  FIELD IS │
│                 GIVEN FOR EACH  VR THAT IS SUPPORTED  BY THE ER (THE  ER INFORMATION │
│                 FIELDS  OF  THE ROUTE_DATA  WILL  BE  IDENTICAL  FOR EACH  OF  THESE │
│                 ENTRIES).  FOR THE UNDEFINED TG_ID'S,  THE VR INFORMATION FIELDS ARE │
│                 RESERVED.                                                │
│                                                                          │
│      INPUT:     RSP(ROUTE_TEST),  DEST_SA (THE  DESTINATION SUBAREA  ADDRESS OF  THE │
│                 EXPLICIT  ROUTE),  AND ER_NUM  (THE  EXPLICIT  ROUTE NUMBER  OF  THE │
│                 EXPLICIT ROUTE)                                          │
│                                                                          │
│      OUTPUT:    INITIALIZED RSP(ROUTE_TEST) ROUTE_DATA FIELDS            │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 ROUTE_TEST_RCV                    PAGE 12-113            │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 ERN_TO_VRN_MAP                    PAGE 12-125            │
│                 FIND_VR_STATUS                    PAGE 12-115            │
│                 UPM_SET_ER_STATUS                 PAGE 12-66             │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL DEST_SA BIT(32);
  DCL ER_NUM BIT(4);
  DCL VR_NUM BIT(4);
  DCL VRN_MASK BIT(16);
  DCL STATUS BIT(8);

  FIND SUBAREA_ROUTING IN SUBAREA_ROUTING_LIST WHERE(SUBAREA_ROUTING.DEST_SA = DEST_SA);
  FIND ERCB IN ERCB_LIST WHERE(ERCB.PARTNER_SA = DEST_SA & ERCB.ER_NUM = ER_NUM);

  IF SUBAREA_ROUTING_PTR = NULL | ERCB_PTR = NULL THEN
    DO;
    . ROUTE_TEST_RSP.CNT_ROUTE_DATA = ROUTE_TEST_RSP.CNT_ROUTE_DATA + 1;
    . CALL UPM_SET_ER_STATUS(STATUS);                          /* PAGE 12-66          */
    . ROUTE_TEST_RSP.ER_STATUS(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = STATUS;
    . IF SUBAREA_ROUTING_PTR = NULL THEN
    .   ROUTE_TEST_RSP.ORIGINATING_ADJ_SA(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = ZERO;
    . ELSE
    .   ROUTE_TEST_RSP.ORIGINATING_ADJ_SA(ROUTE_TEST_RSP.CNT_ROUTE_DATA) =
    .       SUBAREA_ROUTING.ADJ_SA(ER_NUM);
    . ROUTE_TEST_RSP.VR_ID(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = ZERO;
    . ROUTE_TEST_RSP.VR_STATUS(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = ZERO;
    END;


  ELSE
    SCAN PATHCB_LIST PTR(PATHCB_PTR);
    . CALL UPM_SET_ER_STATUS(STATUS);                          /* PAGE 12-66          */
    . ROUTE_TEST_RSP.ER_NUM(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = ER_NUM;
    .
    . IF PATHCB.TG_ID = SUBAREA_ROUTING.TG_ID(ER_NUM) THEN
    .   DO;
    .   . ROUTE_TEST_RSP.ER_STATUS(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = STATUS;
    .   . ROUTE_TEST_RSP.ORIGINATING_ADJ_SA(ROUTE_TEST_RSP.CNT_ROUTE_DATA) =
    .   .       PATHCB.ADJ_SA;
    .   . ROUTE_TEST_RSP.VR_ID(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = ZERO;
    .   . ROUTE_TEST_RSP.VR_STATUS(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = ZERO;
    .   . ROUTE_TEST_RSP.CNT_ROUTE_DATA = ROUTE_TEST_RSP.CNT_ROUTE_DATA + 1;
    .   END;
    . ELSE                                 /* INSERT ENTRIES FOR SUPPORTED VR'S   */
    .   DO;
    .   . IF ERN_TO_VRN_MAP(DEST_SA,VRN_MASK,ER_NUM) = EXIST THEN /* PAGE 12-125    */
    .   .   DO VR_NUM = 0 TO NCB.MAX_VR_NUM;                   /* APPENDIX A          */
    .   .   . IF VRN_MASK(VR_NUM:VR_NUM) = ON THEN
    .   .   .   CALL FIND_VR_STATUS(DEST_SA,VR_NUM,STATUS);    /* PAGE 12-115         */
    .   .   END;
    .   END;
    SCANEND;

  RETURN;
END SET_ER;
```

```
UPM_SET_ER_STATUS: PROCEDURE(STATUS);

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO DETERMINE THE STATUS OF A PARTICULAR TG FOR A (DSA, ERN)  │
  │                                                                              │
  │      INPUT:     ERCB_PTR,  PATHCB_PTR,  AND  SUBAREA_ROUTING_PTR;  STATUS  IS  NOT
  │                 INITIALIZED                                                  │
  │                                                                              │
  │      OUTPUT:    STATUS IS THE  STATE OF THE (DSA, ERN) ALONG A  PARTICULAR TG, CODED
  │                 AS DESCRIBED IN RSP(ROUTE_TEST)                              │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 FIND_ER_STATUS                  PAGE 12-66                   │
  │                 SET_ER                          PAGE 12-65                   │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL STATUS BIT(8);

  RETURN;
END UPM_SET_ER_STATUS;



FIND_ER_STATUS: PROCEDURE(DEST_SA,VR_NUM,ER_NUM,STATUS,ADJ_SA);

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO GATHER INFORMATION ABOUT THE ER THAT SUPPORTS A VR        │
  │                                                                              │
  │      INPUT:     VR_NUM IS  THE VIRTUAL ROUTE NUMBER  AND DEST_SA IS  THE DESTINATION
  │                 SUBAREA FOR A VIRTUAL ROUTE                                  │
  │                                                                              │
  │      OUTPUT:    ER_NUM IS  THE ERN THAT  SUPPORTS THE  VIRTUAL ROUTE; STATUS  IS THE
  │                 STATE OF THE  EXPLICIT ROUTE CODED AS  DESCRIBED IN RSP(ROUTE_TEST);
  │                 ADJ_SA IS THE ADJACENT SUBAREA OF THE EXPLICIT ROUTE         │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 SET_VR                          PAGE 12-114                  │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 UPM_SET_ER_STATUS               PAGE 12-66                   │
  │                 VRN_TO_ERN_MAP                  PAGE 12-124                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL DEST_SA BIT(32);
  DCL VR_NUM BIT(4);
  DCL ER_NUM BIT(4);
  DCL STATUS BIT(8);
  DCL ADJ_SA BIT(32);

  IF VRN_TO_ERN_MAP(DEST_SA,VR_NUM,ER_NUM) =  ¬EXIST THEN        /* PAGE 12-124          */
    DO;
    . STATUS = X'00';                                           /* VR IS NOT DEFINED    */
    . ADJ_SA = ZERO;
    END;
  ELSE
    DO;
    . FIND ERCB IN ERCB_LIST WHERE(ERCB.PARTNER_SA = DEST_SA & ERCB.ER_NUM = ER_NUM);
    . FIND SUBAREA_ROUTING IN SUBAREA_ROUTING_LIST WHERE(SUBAREA_ROUTING.DEST_SA = DEST_SA);
    . FIND PATHCB IN PATHCB_LIST WHERE(PATHCB.TG_ID = SUBAREA_ROUTING.TG_ID(ER_NUM));
    . CALL UPM_SET_ER_STATUS(STATUS);                           /* PAGE 12-66           */
    . ADJ_SA = SUBAREA_ROUTING.ADJ_SA(ER_NUM);
    END;

  RETURN;
END FIND_ER_STATUS;
```

```
CREATE_SUBAREA_ROUTING: PROCEDURE(DEST_SA);

                                                                              /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │     FUNCTION:  TO CREATE  A NEW  SUBAREA ROUTING  ENTRY THAT  DEFINES THE  EXPLICIT │
  │                ROUTE TO  A PARTICULAR  SUBAREA.  THIS  CHANGE IN  SYSTEM DEFINITION │
  │                TABLES    IS   INITIATED    EITHER   BY    AN    NC_ER_OP   OR    AN │
  │                IMPLEMENTATION-DEPENDENT MEANS.                           │
  │                                                                          │
  │     INPUT:     DEST_SA IS THE  ADDRESS OF THE SUBAREA TO  WHICH ROUTING INFORMATION │
  │                IS BEING SPECIFIED.                                       │
  │                                                                          │
  │     OUTPUT:    CREATED   AND   INITIALIZED  SUBAREA_ROUTING   INSERTED   INTO   THE │
  │                SUBAREA_ROUTING_LIST                                      │
  │                                                                          │
  │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
  │                DEFINE_ER_TO_TG                  PAGE 12-33               │
  │                OP_RCV                           PAGE 12-40               │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                              */

  DCL DEST_SA BIT(32);

  CREATE SUBAREA_ROUTING;
  INSERT SUBAREA_ROUTING IN SUBAREA_ROUTING_LIST;
  SUBAREA_ROUTING.DEST_SA = DEST_SA;
  IF NCB.ERN_DEFINITION_CAPABILITY = STATIC_ONLY THEN          /* APPENDIX A   */
    SUBAREA_ROUTING.ER_SYSDEF = STATIC_DEFINITION;
  ELSE
    SUBAREA_ROUTING.ER_SYSDEF = DYNAMIC_DEFINITION;
  SUBAREA_ROUTING.TG_ID = ZERO;

  RETURN;
END CREATE_SUBAREA_ROUTING;
```

```
BUILD_NC_ER_ACT_OR_TEST: PROCEDURE(TYPE) RETURNS(PTR);

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO BUILD AN NC_ER_ACT OR NC_ER_TEST TO ACTIVATE OR TEST A (DSA, ERN)
 │
 │      INPUT:     ERCB_PTR AND PATHCB_PTR, AND PARAMETER TYPE INDICATING WHETHER
 │                 NC_ER_ACT (TYPE='ACT') OR NC_ER_TEST (TYPE='TEST') SHOULD BE BUILT.
 │                 IF TYPE IS 'TEST', ROUTE_TEST IS THE CURRENT MESSAGE UNIT.
 │
 │      OUTPUT:    NC_ER_ACT OR NC_ER_TEST ADDRESSED BY RETURNED POINTER VALUE
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 FSM_PATH                          PAGE 12-75
 │                 TEST_SEND                         PAGE 12-56
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 BUILD_NC_TH_RH                    PAGE 12-123
 │                 UPM_ACT_SEQ_ID                    PAGE 12-69
 │                 UPM_MAX_ER_LENGTH                 PAGE 12-69
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL TYPE CHAR(4);                                          /* 'ACT' OR 'TEST'  */
  DCL ACT_TEST_MU_PTR PTR;

  CREATE ACT_TEST_MU_PTR->MU;
  CALL BUILD_NC_TH_RH(ACT_TEST_MU_PTR);                      /* PAGE 12-123      */

  ACT_TEST_MU_PTR->ERN = ERCB.ER_NUM;
  ACT_TEST_MU_PTR->IERN = RESERVED_ZERO;
  ACT_TEST_MU_PTR->VRN = RESERVED_ZERO;
  ACT_TEST_MU_PTR->TPF = L_PRTY;

  ACT_TEST_MU_PTR->NC_ER_ACT_RQ.FORMAT = FORMAT1;
  ACT_TEST_MU_PTR->NC_ER_ACT_RQ.ER_LENGTH = ZERO;
  CALL UPM_MAX_ER_LENGTH;                                    /* PAGE 12-69       */
  ACT_TEST_MU_PTR->NC_ER_ACT_RQ.ER_NUM = ERCB.ER_NUM;
  ACT_TEST_MU_PTR->NC_ER_ACT_RQ.ORIGINATING_SA = NCB.NODE_SUBAREA_ADDRESS;
  ACT_TEST_MU_PTR->NC_ER_ACT_RQ.REV_ERN_MASK = ALL_ONES;     /* APPENDIX A       */
  ACT_TEST_MU_PTR->NC_ER_ACT_RQ.MAX_PIU_SIZE = ZERO;

  IF TYPE = 'ACT' THEN
    DO;                                                      /* BUILDING NC_ER_ACT */
    . DCF = RH_LENGTH + 37;
    . ACT_TEST_MU_PTR->RQ_CODE = NC_ER_ACT;
    . ACT_TEST_MU_PTR->TG_SWEEP = SWEEP;
    . ACT_TEST_MU_PTR->DSAF = PATHCB.ADJ_SA;
    . ACT_TEST_MU_PTR->NC_ER_ACT_RQ.DESTINATION_SA = ERCB.PARTNER_SA;
    . ACT_TEST_MU_PTR->NC_ER_ACT_RQ.ACT_SEQ_ID = UPM_ACT_SEQ_ID; /* PAGE 12-69    */
    . PATHCB.ACT_SEQ_ID = ACT_TEST_MU_PTR->NC_ER_ACT_RQ.ACT_SEQ_ID;
    END;
  ELSE                                                       /* BUILDING NC_ER_TEST */
    DO;
    . DCF = RH_LENGTH + 39;
    . ACT_TEST_MU_PTR->RQ_CODE = NC_ER_TEST;
    . ACT_TEST_MU_PTR->TG_SWEEP = ¬SWEEP;
    . ACT_TEST_MU_PTR->DSAF = ROUTE_TEST_RQ.ORIGINATING_ADJ_SA;
    . ACT_TEST_MU_PTR->NC_ER_TEST_RQ.DESTINATION_SA = ROUTE_TEST_RQ.DESTINATION_SA;
    . ACT_TEST_MU_PTR->NC_ER_TEST_RQ.RQ_CORRELATION = ROUTE_TEST_RQ.RQ_CORRELATION;
    . ACT_TEST_MU_PTR->NC_ER_TEST_RQ.ORIGINATING_SSCP = OSAF||OEF; /* FROM ROUTE_TEST */
    END;

  RETURN(ACT_TEST_MU_PTR);
END BUILD_NC_ER_ACT_OR_TEST;
```

```
UPM_MAX_ER_LENGTH: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS IMPLEMENTATION-DEPENDENT PROCEDURE COMPUTES  THE MAXIMUM NUMBER
  │                 OF TRANSMISSION GROUPS OVER WHICH AN NC_ER_ACT OR NC_ER_TEST REQUEST
  │                 CAN  BE  TRANSMITTED BEFORE  THE  ACTIVATION  OR TEST  PROCEDURE  IS
  │                 ABORTED.  THE MAXIMUM  ER LENGTH IS INSERTED  IN MAX_ER_LENGTH FIELD
  │                 OF THE NC_ER_ACT OR NC_ER_TEST.
  │
  │      INPUT:     NC_ER_ACT OR NC_ER_TEST
  │
  │      OUTPUT:    ASSIGNED MAX_ER_LENGTH FIELD IN REQUEST
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 BUILD_NC_ER_ACT_OR_TEST            PAGE 12-68
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  RETURN;
END UPM_MAX_ER_LENGTH;



UPM_ACT_SEQ_ID: PROCEDURE RETURNS(CHAR(10));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS   IMPLEMENTATION-DEPENDENT   PROCEDURE   GENERATES   A   UNIQUE
  │                 IDENTIFICATION VALUE TO BE PUT INTO THE NC_ER_ACT REQUEST.
  │
  │      INPUT:     NONE
  │
  │      OUTPUT:    CHARACTER STRING TO BE USED AS CORRELATION VALUE
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 BUILD_NC_ER_ACT_OR_TEST            PAGE 12-68
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  RETURN('');
END UPM_ACT_SEQ_ID;
```

```
BUILD_NC_ER_ACT_OR_TEST_REPLY: PROCEDURE(TYPE);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO BUILD THE COMMON PARTS  OF AN NC_ER_ACT_REPLY OR NC_ER_TEST_REPLY │
│                 REQUEST                                                  │
│                                                                          │
│      INPUT:     NC_ER_ACT OR NC_ER_TEST THE PARAMETER TYPE INDICATES THE VALUE TO BE │
│                 PUT INTO  THE TYPE  FIELD OF THE  REQUEST, AND  CONTAINS ONE  OF THE │
│                 FOLLOWING VALUES:                                        │
│                              • NO_REVERSE_ERN_DEFINED                    │
│                              • ER_LENGTH_ERROR                           │
│                              • ER_NOT_DEFINED                            │
│                              • TG_INOPERATIVE                            │
│                              • PRE_ER_VR_SUPPORT                         │
│                              • ER_RACE                                   │
│                              • POSITIVE_REPLY                            │
│                                                                          │
│      OUTPUT:    NC_ER_ACT_REPLY OR NC_ER_TEST_REPLY AND TGCB_PTR SPECIFYING THE TGCB │
│                 FOR  THE TG  OVER  WITH THE  REQUEST SHOULD  BE  ROUTED.  THE  INPUT │
│                 REQUEST IS DISCARDED.                                    │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 ACT_TEST_RCV                    PAGE 12-60               │
│                 ACT_TEST_SEND                   PAGE 12-59               │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL TYPE BIT(8);
  DCL ACT_TEST_MU_PTR PTR;
  DCL 1 NC_ER_ACT_TEST_RQ LIKE NC_ER_ACT_RQ BASED(ADDR(RU));
  DCL 1 NC_ER_ACT_TEST_REPLY_RQ LIKE NC_ER_ACT_REPLY_RQ BASED(ADDR(RU));

  ACT_TEST_MU_PTR = MU_PTR;                                 /* SAVE PTR TO REQUEST      */
  CREATE MU;
  MU = ACT_TEST_MU_PTR->MU;
  NC_ER_ACT_TEST_REPLY_RQ = ACT_TEST_MU_PTR->NC_ER_ACT_TEST_RQ,BY NAME;
  MUCB.DIRECTION = SEND;

                                                                        /*
                  ┌──────────────────────────────────────────────┐
                  │  TH VALUES  FOR TG_SWEEP,  NTWK_PRTY, VR_SQTI,  │
                  │  IERN, VRN, AND TPF ARE  THE SAME IN THE REPLY  │
                  │  REQUEST AS THEY WERE IN THE ORIGINAL REQUEST.  │
                  └──────────────────────────────────────────────┘
                                                                        */

  ERN = INDEX(ACT_TEST_MU_PTR->NC_ER_ACT_TEST_RQ.REV_ERN_MASK,ON);
  OSAF = NCB.NODE_SUBAREA_ADDRESS;                          /* APPENDIX A               */

  NC_ER_ACT_TEST_REPLY_RQ.MAX_PIU_SIZE = ZERO;
  NC_ER_ACT_TEST_REPLY_RQ.MAX_PIU_SIZE_FROM_ACTIVATE =
        ACT_TEST_MU_PTR->NC_ER_ACT_TEST_RQ.MAX_PIU_SIZE;
  NC_ER_ACT_TEST_REPLY_RQ.REPLY_SA = NCB.NODE_SUBAREA_ADDRESS;   /* APPENDIX A          */

  IF ACT_TEST_MU_PTR->NC_ER_ACT_TEST_RQ.RQ_CODE = NC_ER_ACT THEN
    DO;
    . NC_ER_ACT_TEST_REPLY_RQ.RQ_CODE = NC_ER_ACT_REPLY;
    . DCF = RH_LENGTH + 49;
    END;
  ELSE
    DO;
    . NC_ER_ACT_TEST_REPLY_RQ.RQ_CODE = NC_ER_TEST_REPLY;
    . DCF = RH_LENGTH + 48;
    END;
```

```
  NC_ER_ACT_REPLY_RQ.TYPE = TYPE;

  SELECT ANYORDER(TYPE);                    /* SET UP UNIQUE PORTIONS OF REPLY          */
  .
  . WHEN(NO_REVERSE_ERN_DEFINED)
  .   DO;
  .   . NC_ER_ACT_REPLY_RQ.TG_ADJ_SA = ACT_TEST_MU_PTR->TGCB.ADJ_SA;
  .   . NC_ER_ACT_REPLY_RQ.TG_NUM = ACT_TEST_MU_PTR->TGCB.TGN;
  .   END;
  .
  . WHEN(ER_LENGTH_ERROR)
  .   DO;
  .   . NC_ER_ACT_REPLY_RQ.TG_ADJ_SA = TGCB.ADJ_SA;
  .   . NC_ER_ACT_REPLY_RQ.TG_NUM = TGCB.TGN;
  .   END;
  .
  . WHEN(ER_NOT_DEFINED)
  .   DO;
  .   . NC_ER_ACT_REPLY_RQ.TG_ADJ_SA = ZERO;
  .   . NC_ER_ACT_REPLY_RQ.TG_NUM = ZERO;
  .   . CALL FIND_TGCB(ACT_TEST_MU_PTR->ROUTE_SA,ERN);        /* APPENDIX B          */
  .   END;
  .
  . WHEN(TG_INOPERATIVE)
  .   DO;
  .   . NC_ER_ACT_REPLY_RQ.TG_ADJ_SA = TGCB.ADJ_SA;
  .   . NC_ER_ACT_REPLY_RQ.TG_NUM = TGCB.TGN;
  .   . CALL FIND_TGCB(ACT_TEST_MU_PTR->ROUTE_SA,ERN);        /* APPENDIX B          */
  .   END;
  .
  . WHEN(PRE_ER_VR_SUPPORT)
  .   DO;
  .   . NC_ER_ACT_REPLY_RQ.TG_ADJ_SA = TGCB.ADJ_SA;
  .   . NC_ER_ACT_REPLY_RQ.TG_NUM = TGCB.TGN;
  .   . CALL FIND_TGCB(ACT_TEST_MU_PTR->ROUTE_SA,ERN);        /* APPENDIX B          */
  .   END;
  .
  . WHEN(ER_RACE)
  .   DO;
  .   . NC_ER_ACT_REPLY_RQ.TG_ADJ_SA = RESERVED_ZERO;
  .   . NC_ER_ACT_REPLY_RQ.TG_NUM = RESERVED_ZERO;
  .   END;
  .
  . WHEN(POSITIVE_REPLY)
  .   DO;
  .   . NC_ER_ACT_REPLY_RQ.TG_ADJ_SA = RESERVED_ZERO;
  .   . NC_ER_ACT_REPLY_RQ.TG_NUM = RESERVED_ZERO;
  .   END;
  .
  END;

  DISCARD ACT_TEST_MU_PTR->MU;

  RETURN;
END BUILD_NC_ER_ACT_OR_TEST_REPLY;



ABLE_TO_RCV_ACTVR: PROCEDURE RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO DETERMINE IF AN ER IS IN A STATE THAT ALLOWS A VR TO BE ACTIVATED
  │                 USING IT                                                     │
  │                                                                              │
  │      INPUT:     ERCB_PTR AND TGCB_PTR                                        │
  │                                                                              │
  │      OUTPUT:    BOOLEAN VALUE INDICATING WHETHER  THE ER TO BE USED BY  THE VR BEING
  │                 ACTIVATED IS  IN AN ACCEPTABLE STATE  TO CARRY TRAFFIC (YES)  OR NOT
  │                 (NO)                                                         │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 VR_RCV_CHECKS                        PAGE 12-98              │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 FSM_PATH                             PAGE 12-75              │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  IF ERCB_PTR = NULL THEN
    RETURN(NO);

  ELSE
    DO;
    . FIND PATHCB IN PATHCB_LIST WHERE(PATHCB.TG_ID = TGCB.TG_ID);
    . IF FSM_PATH = (ACTIVE | ACT_RCV) THEN                    /* PAGE 12-75       */
    .   RETURN(YES);
    . ELSE
    .   RETURN(NO);
    END;

END ABLE_TO_RCV_ACTVR;
```

```
SIGNAL_VR_MGR: PROCEDURE(SIGNAL);

                                                                              /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO  SEND  AN  ER_NOT_ACTIVATED  OR ER_ACTIVATED  SIGNAL  TO  THE  VR  │
  │                 MANAGER, INDICATING  THAT A SET OF  VR'S CAN OR CANNOT  BE ACTIVATED  │
  │                 BECAUSE THE UNDERLYING ER CAN OR CANNOT BE ACTIVATED.                 │
  │                                                                                       │
  │      INPUT:     ERCB_PTR.  THE  PARAMETER SIGNAL, EITHER  INOP OR ACT,  INDICATES IF  │
  │                 THE UNDERLYING ER HAS BEEN ACTIVATED  OR NOT AND, THEREFORE, WHETHER  │
  │                 IT CAN SUPPORT A VR.                                                  │
  │                                                                                       │
  │      OUTPUT:    ER_NOT_ACTIVATED  OR   ER_ACTIVATED  SIGNAL   TO  VR   MANAGER  WITH  │
  │                 PARM_ACT_ER ENTITY SUPPLYING ADDITIONAL INFORMATION                   │
  │                                                                                       │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                        │
  │                 FSM_ERN                              PAGE 12-73                       │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL SIGNAL CHAR(4);                     /* 'ACT' OR 'INOP' INDICATES STATUS OF ER      */

  IF ERCB.PENDING_VRNUMS = ALL_OFF THEN
    RETURN;

  CREATE PARM_ACT_ER;                                          /* PAGE 12-126          */
  PARM_ACT_ER.PARTNER_SA = ERCB.PARTNER_SA;                    /* PAGE 12-126          */
  PARM_ACT_ER.VRN_MASK = ERCB.PENDING_VRNUMS;                  /* PAGE 12-126          */
  IF SIGNAL = 'INOP' THEN
    SEND 'ER_NOT_ACTIVATED' TO VR_MGR USING(PARM_PTR = PARM_ACT_ER_PTR);  /* PAGE 12-79  */
  ELSE                                                         /* SIGNAL = 'ACT'       */
    SEND 'ER_ACTIVATED' TO VR_MGR USING(PARM_PTR = PARM_ACT_ER_PTR);      /* PAGE 12-79  */

  ERCB.PENDING_VRNUMS = ALL_OFF;

  RETURN;
END SIGNAL_VR_MGR;



ARE_ANY_PATHS_PENDING: PROCEDURE RETURNS(BIT(1));

                                                                              /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO  DETERMINE IF  ANY  NC_ER_ACT REQUESTS  HAVE  BEEN  SENT FOR  THE  │
  │                 (DSA, ERN) ON A TG OTHER THAN THE ONE CURRENTLY BEING PROCESSED       │
  │                                                                                       │
  │      INPUT:     ERCB_PTR AND PATHCB_PTR                                               │
  │                                                                                       │
  │      OUTPUT:    A  BIT  INDICATING  WHETHER  THERE  ARE  ANY  OUTSTANDING  NC_ER_ACT  │
  │                 REQUESTS (YES) OR NOT (NO)                                            │
  │                                                                                       │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                        │
  │                 INOP_RCV                             PAGE 12-44                       │
  │                 INOP_SEND                            PAGE 12-42                       │
  │                                                                                       │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                            │
  │                 FSM_PATH                             PAGE 12-75                       │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL SCAN_PATHCB_PTR PTR;

  SCAN PATHCB_LIST PTR(SCAN_PATHCB_PTR)
       UNTIL(SCAN_PATHCB_PTR ¬= PATHCB_PTR & SCAN_PATHCB_PTR->FSM_PATH = PEND_SEND);
                                                               /* PAGE 12-75          */
  SCANEND;

  IF SCAN_PATHCB_PTR = NULL THEN
    RETURN(NO);
  ELSE
    RETURN(YES);

END ARE_ANY_PATHS_PENDING;
```

```
FSM_ERN: FSM_DEFINITION CONTEXT(ERCB);

                                                                        /*
```

FUNCTION: TO RETAIN THE CURRENT STATUS OF THE ER'S ASSOCIATED WITH A (DSA, ERN) PAIR. THIS FSM IS CALLED AND QUERIED ONLY BY PROCEDURES IN THE ER MANAGER. THIS FSM REFLECTS SOME OF THE COMPOSITE STATES OF THE FSM_PATH'S RELATING TO THE (DSA, ERN). THE IMPORTANT CONDITIONS OF AN ER PERTAIN TO ITS BEING ACTIVE (I.E., ABLE TO CARRY MESSAGE UNITS), PENDING ACTIVE, OR OPERATIVE. NOT ALL POSSIBLE CONDITIONS OF THE (DSA, ERN) ARE OF INTEREST TO THE ER MANAGER--FOR EXAMPLE, THE RECEIPT OF AN NC_ER_ACT FROM ANOTHER SUBAREA NODE IS OF LITTLE SIGNIFICANCE TO THE ER MANAGER AND THEREFORE THERE IS NO SET OF STATES REFLECTING SUCH AN OCCURRENCE. A MORE COMPLETE DESCRIPTION OF THE FSM'S STATES IS GIVEN BELOW.

ALL INPUT ROWS OF THE FSM REFER TO REQUESTS OR SIGNALS THAT ARE RECEIVED (I.E., FOR REQUESTS, THE MUCB.DIRECTION INDICATOR WOULD BE "RECEIVE"). THE ROWS REFERRING TO NC_ER_INOP ARE NOT DEPENDENT ON THE DIRECTION OF THE REQUEST.

THE RESET STATE EXISTS ONLY IMMEDIATELY AFTER AN ERCB IS CREATED AND IMMEDIATELY BEFORE IT IS DESTROYED. THE ERCB IS CREATED WHEN THE FIRST PATHCB FOR IT IS TO BE CREATED AS THE RESULT OF RECEIVING AN NC_ER_OP; THE ERCB IS DESTROYED WHEN THE LAST PATHCB FOR IT IS TO BE DESTROYED AS THE RESULT OF RECEIVING AN NC_ER_INOP.

BEING IN THE OP STATE INDICATES THAT THE (DSA, ERN) IS OPERATIVE ALONG SOME NUMBER OF TG'S. WHETHER THE SET OF TG'S INCLUDES THE ONE THAT IS DEFINED FOR THIS (DSA, ERN) IS IRRELEVANT. THE ER MANAGER RECOGNIZES A (DSA, ERN) AS BEING OPERATIVE WHEN IT RECEIVES AN NC_ER_OP FOR IT. FOR THE PURPOSES OF THIS FSM, HAVING RECEIVED AN NC_ER_ACT ON ANY NUMBER OF TG'S DOES NOT AFFECT THE STATE OF THE (DSA, ERN).

THE PEND_ACT STATE IS ENTERED IF AN NC_ER_ACT HAS BEEN SENT ON EXACTLY ONE OF THE TG'S FOR THIS (DSA, ERN). THE FSM_PATH FOR ANY NUMBER OF PATHCB'S MAY REFLECT THE RECEIPT OF AN NC_ER_OP (OPERATIVE STATE) OR AN NC_ER_ACT (ACT_RCV STATE).

THE CONTEND STATE IS ENTERED WHEN MULTIPLE NC_ER_ACT REQUESTS ARE SENT OVER DIFFERENT TG'S. IF THE SUBAREA_ROUTING_LIST INDICATES THAT A (DSA, ERN) IS STATICALLY DEFINED, THIS STATE CAN NEVER BE ENTERED. IF THE (DSA, ERN) IS DYNAMICALLY DEFINED, WHEN MULTIPLE NC_ER_OP REQUESTS OVER DIFFERENT TG'S ARE RECEIVED, AN NC_ER_ACT IS SENT OVER THE FIRST TWO TG'S THAT BECOME OPERATIVE. THE FIRST POSITIVE NC_ER_ACT_REPLY RECEIVED DETERMINES WHICH TG IS USED WHEN ROUTING MESSAGE UNITS USING THIS (DSA, ERN).

THE ACTIVE STATE IS ENTERED AFTER RECEIVING A POSITIVE NC_ER_ACT_REPLY. THE SEQUENCE OF TG'S TO BE USED BY THE ER HAS BEEN SWEPT OF ALL RESIDUAL TRAFFIC; ONLY WHEN THE ER IS IN THIS STATE CAN IT SUPPORT TRAFFIC ON A VR.

REFERENCED BY THE FOLLOWING PROCEDURE(S):

ACT_SEND	PAGE 12-55
ACT_TEST_RCV	PAGE 12-60
ACT_TEST_REPLY_RCV	PAGE 12-64
ACTVR_RCV	PAGE 12-96
FSM_PATH	PAGE 12-75
INOP_RCV	PAGE 12-44
INOP_SEND	PAGE 12-42
OP_RCV	PAGE 12-40
OP_SEND	PAGE 12-39
REDUCE_REVERSE_ERN	PAGE 12-62

REFERS TO THE FOLLOWING PROCEDURE(S):

FSM_PATH	PAGE 12-75
SIGNAL_VR_MGR	PAGE 12-72
VR_MGR	PAGE 12-79


```
                                                                        */

DCL COPY_TGCB_PTR PTR;
```

INPUTS	STATE NAME ----> RESET	OP	PEND ACT	ACTIVE	CONTEND
	STATE NUMBER --> 1	2	3	4	5
NC_ER_INOP, EMPTY_PATHCB	/	1(A)	1(B)	1(A)	1(B)
NC_ER_INOP,¬EMPTY_PATHCB, TG_ID	/	-	2(C)	2	-
NC_ER_INOP,¬EMPTY_PATHCB,¬TG_ID	/	-	-	-	-
NC_ER_OP, STATIC	2	-	-	-	/
NC_ER_OP,¬STATIC	2(I)	5(D)	-	-	-
'ACTIVATE_ER',¬PATH	/	-(E)	/	/	-(F)
'ACTIVATE_ER', PATH	/	3(G)	-(F)	-(H)	/
NC_ER_ACT,¬SPRAY	/	-(J)	-(J)	-(J)	3(K)
NC_ER_ACT, SPRAY, WINNER	/	-(J)	-(J)	/	-(L)
NC_ER_ACT, SPRAY,¬WINNER	/	-(J)	-(J)	/	3(M)
NC_ER_ACT_REPLY,GOOD	/	/	4(O)	-	4(N)
NC_ER_ACT_REPLY,BAD,¬OTHERS_PENDING	/	/	2(C)	-	2(C)
NC_ER_ACT_REPLY,BAD, OTHERS_PENDING	/	/	/	-	-
'DEFINE'	/	3	/	/	/

OUTPUT CODE	FUNCTION
A	DESTROY PATHCB_LIST; REMOVE ERCB FROM ERCB_LIST DISCARD;
B	CALL SIGNAL_VR_MGR('INOP'); /* PAGE 12-72 */ DESTROY PATHCB_LIST; REMOVE ERCB FROM ERCB_LIST DISCARD;
C	CALL SIGNAL_VR_MGR('INOP'); /* PAGE 12-72 */
D	COPY_TGCB_PTR = TGCB_PTR; CALL FSM_PATH('SPRAY'); /* PAGE 12-75 */ FIND TGCB IN TGCB_LIST WHERE(TGCB.TG_ID = SUBAREA_ROUTING.TG_ID(ERCB.ER_NUM)); CALL FSM_PATH('SPRAY'); /* PAGE 12-75 */ TGCB_PTR = COPY_TGCB_PTR; SUBAREA_ROUTING.TG_ID(ERCB.ER_NUM) = ZERO;
E	SEND 'ER_NOT_ACTIVATED' TO VR_MGR USING(PARM_PTR = PARM_ACT_ER_PTR);/* PAGE 12-79 */
F	ERCB.PENDING_VRNUMS = ERCB.PENDING_VRNUMS | PARM_ACT_ER.VRN_MASK; /* PAGE 12-126 */ DISCARD PARM_ACT_ER; /* PAGE 12-126 */
G	ERCB.PENDING_VRNUMS = ERCB.PENDING_VRNUMS | PARM_ACT_ER.VRN_MASK; /* PAGE 12-126 */ DISCARD PARM_ACT_ER; /* PAGE 12-126 */ CALL FSM_PATH('ACTIVATE_ER'); /* PAGE 12-75 */
H	SEND 'ER_ACTIVATED' TO VR_MGR USING(PARM_PTR = PARM_ACT_ER_PTR); /* PAGE 12-79 */
I	SUBAREA_ROUTING.TG_ID(ERCB.ER_NUM) = TGCB.TG_ID;
J	CALL FSM_PATH; /* PAGE 12-75 */
K	SUBAREA_ROUTING.TG_ID(ERCB.ER_NUM) = TGCB.TG_ID; CALL FSM_PATH; /* PAGE 12-75 */
L	NC_ER_ACT_RQ.REV_ERN_MASK(ERCB.ER_NUM:ERCB.ER_NUM) = OFF;
M	SUBAREA_ROUTING.TG_ID(ERCB.ER_NUM) = TGCB.TG_ID; CALL FSM_PATH; /* PAGE 12-75 */ CALL FSM_PATH('CONTEND_RESEND'); /* PAGE 12-75 */
N	CALL SIGNAL_VR_MGR('ACT'); /* PAGE 12-72 */ FIND SUBAREA_ROUTING IN SUBAREA_ROUTING_LIST WHERE(SUBAREA_ROUTING.DEST_SA = ERCB.PARTNER_SA); SUBAREA_ROUTING.TG_ID(ERCB.ER_NUM) = TGCB.TG_ID;
O	CALL SIGNAL_VR_MGR('ACT'); /* PAGE 12-72 */

END FSM_ERN;

```
FSM_PATH: FSM_DEFINITION CONTEXT(PATHCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO RETAIN THE CURRENT STATE OF THE ER'S ASSOCIATED WITH A (DSA, ERN)
│                PAIR FOR A PARTICULAR TG. THE STATES OF A PATHCB ARE UNRELATED TO
│                WHETHER THE CORRESPONDING TG_ID IS THE ONE DEFINED FOR THE
│                (DSA, ERN) OR NOT.  THE STATES REFLECT COMBINATIONS OF THE RECEIPT
│                OF AN NC_ER_OP OR NC_ER_INOP, THE RECEIPT OF AN NC_ER_ACT, THE
│                TRANSMISSION OF AN NC_ER_ACT, AND THE RECEIPT OF AN NC_ER_ACT_REPLY.
│                THE STATES OF DIFFERENT PATHCB'S FOR THE SAME (DSA, ERN) ARE
│                INDEPENDENT OF EACH OTHER, EXCEPT THAT NO MORE THAN ONE PATHCB CAN
│                BE ACTIVE AT A TIME.
│
│                ALL INPUT ROWS OF THE FSM REFER TO REQUESTS OR SIGNALS THAT ARE
│                RECEIVED (I.E., FOR REQUESTS, THE MUCB.DIRECTION INDICATOR WOULD BE
│                "RECEIVE"). THE ROW REFERRING TO NC_ER_INOP ARE NOT DEPENDENT ON
│                THE DIRECTION OF THE REQUEST.
│
│                THE RESET STATE EXISTS ONLY IMMEDIATELY AFTER THE PATHCB IS CREATED
│                AS THE RESULT OF RECEIVING AN NC_ER_OP, AND JUST BEFORE IT IS
│                DESTROYED AS THE RESULT OF PROCESSING AN NC_ER_INOP.
│
│                THE OP STATE IS ENTERED WHEN AN NC_ER_OP IS RECEIVED OVER THE TG
│                IDENTIFIED BY THE PATHCB'S TG_ID.
│
│                THE PEND_SEND STATE IS ENTERED WHEN AN NC_ER_ACT HAS BEEN SENT, BUT
│                NC_ER_ACT CAN HAVE BEEN RECEIVED.
│
│                THE ACT_RCV STATE IS ENTERED WHEN AN NC_ER_ACT HAS BEEN RECEIVED AND
│                AN NC_ER_ACT_REPLY SENT, BUT NO NC_ER_ACT HAS BEEN SENT. THERE ARE
│                TWO CASES WHEN ONE SIDE OF AN ER MIGHT BE IN ACT_RCV STATE, YET THE
│                OTHER SIDE IS NOT EITHER PENDING NOR ACTIVE. ONE SUBAREA NODE SENDS
│                AN NC_ER_ACT.  A TRANSMISSION GROUP BECOMES INOPERATIVE AND THEN
│                OPERATIVE AFTER THE NC_ER_ACT PASSES.  IF NC_ER_INOP AND NC_ER_OP
│                PASS THE NC_ER_ACT AND GET TO THE DESTINATION SUBAREA NODE FIRST,
│                THAT NODE ENTERS THE ACT_RCV STATE, BUT THE ER IN THE ORIGINATION
│                SUBAREA NODE IS RESET BY THE NC_ER_INOP. THIS APPARENT MISMATCH OF
│                STATES IS NOT IMPORTANT BECAUSE THE DESTINATION SUBAREA NODE DOES
│                NOT ALLOW A VR TO USE THE ER UNTIL IT HAS SENT ITS OWN NC_ER_ACT
│                ALONG THE ROUTE.  THE OTHER SITUATION INVOLVES THE TRANSMISSION OF
│                MULTIPLE NC_ER_ACT REQUESTS BY A NODE ALLOWING DYNAMIC ER
│                DEFINITIONS.  THE ORIGINATING NODE ACCEPTS ONLY ONE OF THE
│                NC_ER_ACT_REPLY REQUESTS AND REJECTS THE OTHERS, EVEN THOUGH THE
│                DESTINATION NODE MAY HAVE ENTERED THE ACT_RCV STATE FOR ALL
│                NC_ER_ACT REQUESTS IT RECEIVED.
│
│                THE PEND_SEND_ACT_RCV STATE IS ENTERED WHEN AN NC_ER_ACT HAS BEEN
│                SENT, AND AN NC_ER_ACT HAS BEEN RECEIVED AND AN NC_ER_ACT_REPLY HAS
│                BEEN SENT.
│
│                THE ACTIVE STATE IS ENTERED WHEN AN NC_ER_ACT HAS BEEN SENT AND A
│                POSITIVE NC_ER_ACT_REPLY HAS BEEN RECEIVED AND ACCEPTED. THE TG
│                CORRESPONDING TO THIS PATHCB WILL BE USED FOR ALL MESSAGE UNITS
│                USING THIS (DSA, ERN). NO MORE THAN 1 PATHCB CAN BE ACTIVE AT ONE
│                TIME.
│
│                THE ACT_RCV_NOTDEF STATE IS ENTERED WHEN AN NC_ER_ACT WITH NO USABLE
│                REVERSE ERN'S IS RECEIVED (I.E., THE REV_ERN_MASK IS ALL OFF). IF
│                THE (DSA, ERN) DEFINITION (MAPPING TO A TG_ID) IS CHANGED SO THAT
│                THAT ER WOULD HAVE A VALID REVERSE ERN, AN NC_ER_ACT IS SENT.
│
│                THE ACT_SEND_NOTDEF STATE IS ENTERED WHEN AN NC_ER_ACT_REPLY IS
│                RECEIVED WITH A TYPE CODE INDICATING THAT NO USABLE REVERSE ERN'S
│                EXIST.  NO FURTHER NC_ER_ACT REQUESTS ARE SENT ALONG THE TG FOR THIS
│                (DSA, ERN) UNTIL THE PARTNER SUBAREA NODE FIRST SENDS AN NC_ER_ACT
│                INDICATING THAT ITS ROUTING DEFINITION TABLES HAVE BEEN CHANGED.
│
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                ABLE_TO_RCV_ACTVR                PAGE 12-71
│                ACT_TEST_REPLY_RCV               PAGE 12-64
│                ARE_ANY_PATHS_PENDING            PAGE 12-72
│                DEFINE_ER_TO_TG                  PAGE 12-33
│                FSM_ERN                          PAGE 12-73
│                INOP_RCV                         PAGE 12-44
│                INOP_SEND                        PAGE 12-42
│                OP_RCV                           PAGE 12-40
│
│     REFERS TO THE FOLLOWING PROCEDURE(S):
│                BUILD_NC_ER_ACT_OR_TEST          PAGE 12-68
│                FSM_ERN                          PAGE 12-73
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

DCL ACT_MU_PTR PTR;
```

STATE NAME ----> / STATE NUMBER --> / INPUTS	RESET 1	OP 2	PEND SEND 3	ACT RCV 4	PEND SEND ACT RCV 5	ACTIVE 6	ACT RCV NOTDEF 7	ACT SEND NOTDEF 8
NC_ER_INOP	/	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)	1(A)
NC_ER_OP	2	>	>	>	>	>	>	>
'SPRAY'	/	3(B)	/	/	/	/	/	/
'ACTIVATE_ER'	/	3(C)	/	5(C)	/	/	/	/
'CONTEND_RESEND'	/	/	/	/	-(C)	/	/	/
NC_ER_ACT, TG_ID	/	4(D)	5(D)	-(D)	-(D)	-(D)	-	4(D)
NC_ER_ACT,¬TG_ID	/	7	/	/	/	/	-	-
NC_ER_ACT_REPLY,GOOD, TG_ID	/	/	6(E)	/	6(E)	/	/	/
NC_ER_ACT_REPLY,GOOD,¬TG_ID	/	/	2	/	/	/	/	/
NC_ER_ACT_REPLY,NOT_REV	/	/	8	/	/	/	/	/
NC_ER_ACT_REPLY,BAD	/	/	2	/	/	/	/	/
'DEFINE'	>	-	>	>	>	>	3(F)	-

OUTPUT CODE	FUNCTION
A	IF SUBAREA_ROUTING.ER_SYSDEF(ERCB.ER_NUM) = DYNAMIC_DEFINITION & SUBAREA_ROUTING.TG_ID(ERCB.ER_NUM) = PATHCB.TG_ID THEN SUBAREA_ROUTING.TG_ID(ERCB.ER_NUM) = ZERO; REMOVE PATHCB FROM PATHCB_LIST DISCARD;
B	ACT_MU_PTR = BUILD_NC_ER_ACT_OR_TEST('ACT'); /* PAGE 12-68 */ ACT_MU_PTR->NC_ER_ACT_RQ.DYNAMIC_ER_DEFN = ON; FIND TGCB IN TGCB_LIST WHERE(TGCB.TG_ID = PATHCB.TG_ID); SEND ACT_MU_PTR->MU TO PC.TGC.LIST_BY_PRTY; /* CHAPTER 3 */
C	MU_PTR = BUILD_NC_ER_ACT_OR_TEST('ACT'); /* PAGE 12-68 */ SEND MU TO PC.TGC.LIST_BY_PRTY; /* CHAPTER 3 */
D	ERCB.RERN_MASK(ERN:ERN) = ON;

E	ERCB.ER_LEN = NC_ER_ACT_REPLY_RQ.ER_LENGTH;
	ERCB.RERN_MASK = ERCB.RERN_MASK | NC_ER_ACT_REPLY_RQ.REV_ERN_MASK;
	IF NC_ER_ACT_REPLY_RQ.TYPE = X'03' THEN
	ERCB.ER_VR_SUPP = PRE_ER_VR;
	ELSE
	ERCB.ER_VR_SUPP = ¬PRE_ER_VR;

F	MU_PTR = BUILD_NC_ER_ACT_OR_TEST('ACT'); /* PAGE 12-68 */ SEND MU TO PC.TGC.LIST_BY_PRTY; /* CHAPTER 3 */ CALL FSM_ERN('DEFINE'); /* PAGE 12-73 */

```
END FSM_PATH;
```

VIRTUAL ROUTE MANAGER

The VR manager activates, deactivates, and tests virtual routes. For the most part, a virtual route is activated when activation of a session requires the virtual route, and it is deactivated when there are no longer any sessions assigned to it, or when conditions in the network cause the virtual route to become inoperative. Virtual route testing is initiated by an ROUTE_TEST request sent from an SSCP. These functions are described in the following three sections: "Virtual Route Activation," "Virtual Route Deactivation," and "Virtual Route Testing."

Information pertinent to a specific virtual route is kept in the virtual route control block (VRCB), described in Appendix A. This control block exists only when the VR is in a non-reset state. Virtual routes may be between two different subarea nodes, or may be entirely within a single subarea node. For virtual routes between two different subarea nodes, the VRCB is created in each subarea when virtual route activation is initiated and destroyed when virtual route deactivation is complete. For each virtual route completely within a subarea node, a VRCB is created and the VR activated at system definition time; the VR is never deactivated.

Five FSMs are anchored in each VRCB; two of them (FSM_VR and FSM_DACTVR_DIRECTION) are described in this chapter. FSM_VR holds the activation and deactivation status of the VR; FSM_DACTVR_DIRECTION is used to determine whether this subarea may send DACTVR(Orderly). The other FSMs are described in Chapter 3.

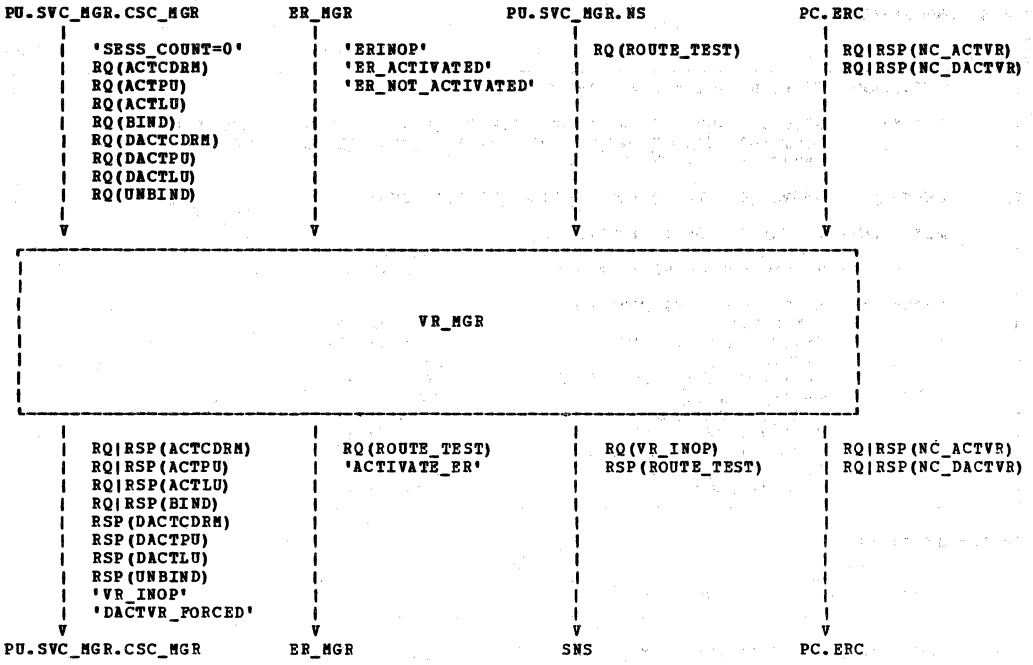


Figure 12-12. VR Manager Inputs and Outputs

```
VR_MGR: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO ROUTE SIGNALS AND PIU'S                               │
│                                                                          │
│      INPUT:     SIGNALS AND PIU'S FROM PC.ERC (CHAPTER 3), PU.SVC_MGR.CSC_MGR
│                 (CHAPTER 13), PU.SVC_MGR.NS (CHAPTER 11), THE HIGHER-LEVEL SCHEDULER
│                 (APPENDIX C), AND ER_MGR                                 │
│                                                                          │
│      OUTPUT:    SIGNAL OR PIU TO THE APPROPRIATE PROCEDURE               │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 FSM_ERM                          PAGE 12-73              │
│                 PU.SVC_MGR.PC_ROUTE_MGR.RCV      PAGE 12-13              │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 ACTVR_RCV                        PAGE 12-96              │
│                 CANCEL_VR_RESERVATION            PAGE 12-102             │
│                 DACTVR_RCV                       PAGE 12-108             │
│                 ER_ACTIVATION_TERMINATOR         PAGE 12-92              │
│                 ROUTE_TEST_RCV                   PAGE 12-113             │
│                 SEND_DACTVR_FORCED               PAGE 12-107             │
│                 SEND_DACTVR_ORDERLY              PAGE 12-106             │
│                 UPM_VR_ID_LIST_REORDER           PAGE 12-102             │
│                 VR_ID_LIST_PROCESSOR             PAGE 12-88              │
│                 VR_INOP_SEND                     PAGE 12-110             │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  SELECT ANYORDER;
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │       INPUT PIU'S FROM PC.ERC (CHAPTER 3)        │
                  └──────────────────────────────────────┘
                                                                        */
  . WHEN((INPUT(RQ) | INPUT(RSP)) & RU_CTGY = NC & RQ_CODE = NC_ACTVR)
  .   CALL ACTVR_RCV;                                    /* PAGE 12-96   */
  .
  . WHEN((INPUT(RQ) | INPUT(RSP)) & RU_CTGY = NC & RQ_CODE = NC_DACTVR)
  .   CALL DACTVR_RCV;                                   /* PAGE 12-108  */
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │   INPUT FROM PU.SVC_MGR.CSC_MGR (CHAPTER 13)     │
                  └──────────────────────────────────────┘
                                                                        */
  . WHEN(INPUT(RQ) & RU_CTGY = SC & RQ_CODE = (ACTCDRM | ACTLU | ACTPU | BIND))
  .   DO;
  .   . VR_ID_LIST_PTR = PARM_PTR;          /* PU.SVC_MGR.CSC_MGR PASSES VR_ID_LIST AS  */
  .   .                                     /* A PARAMETER.          APPENDIX A         */
  .   . CALL UPM_VR_ID_LIST_REORDER;                   /* PAGE 12-102  */
  .   . CALL VR_ID_LIST_PROCESSOR(1); /* BEGIN PROCESSING FIRST ENTRY IN VR_ID_LIST      */
  .   .                                     /* PAGE 12-88                               */
  .   END;
  .
  . WHEN(INPUT(RQ) & RU_CTGY = SC & RQ_CODE = (DACTCDRM | DACTLU | DACTPU | UNBIND))
  .   CALL CANCEL_VR_RESERVATION;                        /* PAGE 12-102  */
  .
  . WHEN(INPUT('SESS_COUNT=0'))
  .   CALL SEND_DACTVR_ORDERLY;                          /* PAGE 12-106  */
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │              INPUT FROM ER_MGR                   │
                  └──────────────────────────────────────┘
                                                                        */
  . WHEN(INPUT('ERINOP'))
  .   CALL VR_INOP_SEND;                                 /* PAGE 12-110  */
  .
  . WHEN(INPUT('ER_ACTIVATED') | INPUT('ER_NOT_ACTIVATED'))
  .   CALL ER_ACTIVATION_TERMINATOR;                     /* PAGE 12-92   */
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │       INPUT FROM PU.SVC_MGR.NS (CHAPTER 11)      │
                  └──────────────────────────────────────┘
                                                                        */
  . WHEN(INPUT(RQ) & RU_CTGY = FMD & NSC_RQ.NS_HEADER = ROUTE_TEST_HDR)
  .   CALL ROUTE_TEST_RCV;                               /* PAGE 12-113  */
                                                                        /*
                  ┌──────────────────────────────────────┐
                  │       INPUT FROM THE HIGHER-LEVEL SCHEDULER      │
                  └──────────────────────────────────────┘
                                                                        */
  . WHEN(INPUT('SEND_DACTVR_F'))
  .   CALL SEND_DACTVR_FORCED;                           /* PAGE 12-107  */
  END;

  RETURN;
END VR_MGR;
```

This page intentionally left blank

VIRTUAL ROUTE ACTIVATION

When the common session control manager (Chapter 13) receives a session activation request (ACTCDRM, ACTPU, ACTLU, or BIND), it requests that the VR manager designate the VR to be used by the session. A VR is activated only when required for the activation of a session.

Once a VR is activated, sessions can be assigned to it. This operation is relatively simple when (1) the VR control block (VRCB) has already been created, (2) the VR is in the active state, and (3) the VR satisfies the RERN requirement of the session being assigned to it (see the "Minimal ER-VR Protocol Support" section below for a description of the RERN requirement of a session). In this case, no route activation requests need flow to other nodes in the network.

On the other hand, when a VR has to be activated--that is, an ER has to be obtained to support the VR, and the VR managers at each end of the VR have to be synchronized with respect to the state of the VR and its attributes--requests must flow through the network before a session can be assigned to the VR. Activating a VR is a three-phase process. First, the ER manager must be signaled to select (and activate, if necessary) an ER to support the VR--sometimes an NC_ER_ACT request is sent into the network as a result of this signal from the VR manager to the ER manager. (The ER manager functions are discussed elsewhere in this chapter.) Second, upon notification from the ER manager that an ER for the VR is active, the VR manager sends an NC_ACTVR request to the opposite end of the VR. Third, a positive response to NC_ACTVR must be received to acknowledge that the node at the other end of the VR is ready to accept traffic on the VR. The description of the functions performed to activate a VR is presented from two perspectives: one view is that of the VR manager initiating the VR activation; the other is that of the VR manager receiving the VR activation request.

VR Activation and Class of Service

The CSC manager sends a session activation request to the VR manager with a class of service specification, called a VR identifier list, which contains a list of (VRN, TPF) pairs, each pair referring to a VR to which the session may be assigned. (See Chapters 1, 6, and 8 for a discussion of class of service.) Once, before starting to process the VR identifier list, the VR manager invokes an exit to an installation-defined UPM to allow reordering of the list. The installation-defined UPM may change the list in any or all of the following ways: reorder the (VRN, TPF) pairs, add (VRN, TPF) pairs, or delete (VRN, TPF) pairs. These changes apply only to the VR identifier list for the session activation request being processed; they do not affect the class of service specification for any subsequent session activations.

When the VR identifier list reordering is complete, the VR manager attempts to assign the session to the VR determined by the first (VRN, TPF) pair in the list and the two subareas at the ends of the VR; the two subareas are determined by the DSAF in the TH of the session activation request and by the subarea in which the VR manager resides. The pairs in the VR identifier list are examined in order to determine the VR to which the session should be assigned; the next (VRN, TPF) pair in the list is examined as a possible VR for the session only after determining that the current pair specifies a VR that is neither active nor can be activated. The VR manager satisfies a request for session assignment to a VR by setting VRCB_PTR to address the VRCB of an active VR and returning the session activation request to the CSC manager.

Locating a Suitable VRCB

VRCBs are created dynamically. For a specified VR, the VR manager first determines whether a VRCB has been created. If it has, the VR may not be in the active state, or it may not satisfy the RERN requirement for the session.

If no VRCB exists for the VR, one is created, if possible, and VR activation is attempted. In this case, the VR manager invokes the ER manager to activate an ER to support the VR. After the VR manager invokes the ER manager, the VR is in a state pending ER activation (FSM_VR is in the PEND_ER state). The VR manager adds a VR_RESERVATION entity to the VR_RESERVATION_LIST of the VRCB, indicating that processing of another session activation request is waiting for the pending event. Each VR_RESERVATION entity contains the session activation request, VR identifier list, and the index within the VR identifier list of the (VRN, TPF) pair being processed. (The VR_RESERVATION_LIST is described in Appendix A.) The entire VR_RESERVATION_LIST for a

particular VRCB is purged either when the VR becomes active or when the VR manager determines that it cannot be activated (e.g., when the ER manager replies that the ER that supports the VR cannot be activated). Individual VR_RESERVATION entities (which contain a session activation request) are discarded from the VR_RESERVATION_LIST only when a corresponding session deactivation request is received.

If it is not possible to create a VRCB because of an implementation-dependent lack of resources, the VR manager will attempt to assign the session to the next VR specified in the VR identifier list.

If the VRCB exists, and the VR is in the active state, the VR manager sets VRCB_PTR to address that VRCB, and returns the session activation request to the CSC manager.

If the VRCB exists, but the VR is not in the active state, the VR is in one of the two classes of pending states. Some pending states (e.g., PEND_ER) are possible intermediate states before the VR becomes active; other states (e.g., PEND_RESET_F_SEND) require a transition of the VR to the reset state before it can become active again.

For the former class of VR states, activation of the VR or its underlying ER is in progress already, so the VR manager adds a VR_RESERVATION entity to the VR_RESERVATION_LIST of the VRCB. For this class of pending state, a response to an NC_ACTVR request or notification from the ER manager is required. When the pending event completes, the VR manager recovers the session activation request and VR identifier list, and VR processing continues based on the current state of the VR.

If the VR is in a pending state that requires transition to the RESET state, the session cannot be assigned to the VR, and the VR manager attempts to assign the session to the next VR specified in the VR identifier list.

If no VR in the VR identifier list can be activated, the session activation request is changed into a negative response with sense code X'8013' (COS Not Available) and is returned to the CSC manager without an accompanying VRCB. The meaning of this sense code is that for each VR specified in the VR identifier list associated with the session activation request, either no ERN is designated to support the VRN, an ER could not be activated to support the VR, the VR could not be activated, or the attributes of the VR did not satisfy the RERN requirement for the session.

Requesting ER Activation

The VR manager constructs a parameter list for the ER manager indicating the destination subarea to which an ER must be activated and the VRN. The ER manager maps the VRN to an ERN and tries to activate the ER. The parameter list is sent with an ACTIVATE_ER signal.

FSM_VR moves from RESET to PEND_ER state when the ACTIVATE_ER signal is sent. In this state, the VR manager anticipates an ACT or an ER_NOT_ACTIVATED signal from the ER manager.

If an ER_ACTIVATED signal is received, the ERCB_PTR addresses the explicit route control block (ERCB) for the active ER that can be used to support the VRN to the destination subarea specified in the parameter list. Information in the ERCB is used to initialize parts of the VRCB. For instance, the VR manager records in the VRCB whether the ER supports ER-VR protocols.

Minimal ER-VR Protocol Support

If the ER includes at least one node that does not support ER-VR protocols, VR pacing will not be used on the VR; thus, there is no need either to set VR pacing values or to create a pacing queue for the VR. Also, in this case NC_ACTVR cannot be sent to the VR Manager at the other end of the VR. Therefore, the VR is considered to be active when the underlying ER becomes active; the VR manager is then able to send to the CSC manager all session activation requests held in the VR_RESERVATION_LIST. As the CSC manager receives each session activation request from the VR manager, the CSC manager assigns the session to the VR represented by the VRCB.

In this case, no NC_ACTVR request is sent to the VR manager at the other end of the virtual route, and, therefore, no VRCB is generated in that node. If the node at the other end of the virtual route supports ER-VR protocols, the virtual route control (VRC) component (Chapter 3) in that node recognizes when it receives a session activation request on a virtual route for which it has no VRCB. The VRC component then generates a VRCB, and recognizes that it is for a virtual route that contains at least one node that does not support ER-VR protocols.

When every node traversed by the ER supports ER-VR protocols, an NC_ACTVR request is sent to the opposite end of the VR. Some of the NC_ACTVR RU fields are set to indicate properties of the ER. In particular, the length of the ER is used to set minimum and maximum values for window size. The minimum is the ER length; the maximum is three times the ER length.

A session is assigned to a VR that maps to an ER having an RERN equal to 0 when (1) it is an LU-LU session, (2) the PLU is in the subarea of a PU_T4 or PU_T5 that supports ER-VR protocols, (3) the SLU is in the subarea of a PU_T4 that supports ER-VR protocols, and (4) the SLU has an active session with a control point (SSCP) that does not support ER-VR protocols. This restriction enables the SSCP that has the active SSCP-LU session with the SLU to receive LSA notification if the LU-LU session is disrupted by a VR outage.

When the LSA RU performs route outage notification, only the subareas that are routed to by way of ERs that have ERN equal to 0 are reported in the RU. In other words, in order for the loss of a subarea to be reported in LSA, the ER number used to transmit data from an LU in that subarea must be 0.

ACTIVATE VIRTUAL ROUTE (NC_ACTVR)

Flow: VR manager to VR manager (Expedited), with TG Sweep = SWEEP, at the transmission priority of the VR

Principal FSMs: FSM_VR (Page 12-121)
FSM_DACTVR_DIRECTION (Page 12-122)

NC_ACTVR activates only those VRs that are not entirely within one subarea and that contain only nodes that support ER-VR protocols.

NC_ACTVR initializes the state and attributes of the VR at each of its end nodes. The attributes specified in NC_ACTVR are minimum window size, maximum window size, initial VR sequence number, and the ERN and RERN of the underlying ER.

The NC_ACTVR response indicates that the other end of the VR either has activated the VR and is ready for traffic (positive response), or that it has not activated the VR (negative response). In the event a positive response is received, all session activation requests waiting for the VR are returned to the CSC manager, with VRCB_PTR addressing the appropriate VRCB. (The only exception to this is the case of a session activation request that requires an RERN of 0. If the VR has an RERN of 0, the session activation request is returned to the CSC manager; if not, the VR manager attempts to assign the session to the next VR in the VR identifier list.) The negative NC_ACTVR response has a sense code to indicate the reason the VR was not activated. The sense codes are:

X'080D'	NC_ACTVR Race Condition--Response Sender Wins
X'0812'	No VRCB Available
X'0815'	VR Already Active
X'0873'	VR to ER Mapping Not Defined
X'0874'	ER Not in a Valid State to Support NC_ACTVR
X'0875'	Incorrect or Undefined ER Specified for VR
X'0876'	ERN and RERN Not Compatible

When an X'080D' negative response is received, the VR is active as a result of an NC_ACTVR sent previously by the sender of the negative response. When an X'0815' negative response is received, the response is logged and the VR specified by the next entry in the VR identifier list is evaluated. When an X'0812', X'0873', X'0874', X'0875', or X'0876' negative response is received, the VR specified by the next entry in the VR identifier list is evaluated.

The NC_ACTVR receiver does three things when the VR can be activated: (1) a VRCB is created, (2) fields in the VRCB are set based on information in the NC_ACTVR, and (3) the NC_ACTVR request is changed into a response and returned to

the originator of the request. When the VR cannot be activated, a negative response is returned. A race resulting from two VR managers sending NC_ACTVR to each other is resolved by subarea address: the NC_ACTVR originated at the higher numbered subarea is the winner.

Activation Completion

The VR manager completes the activation of the VR after both ends of the VR are synchronized and the VRCBs at each end of the VR are initialized. When the VR is activated, the VR manager notifies the CSC manager by setting VRCB_PTR to address the VRCB. Other sessions can be assigned to the VR while it remains active. While the VR is active, VRC (Chapter 3) regulates the flow of its traffic.

While in a pending state trying to activate a VR, the VR manager may receive a session deactivation request (DACTCDRM, DACTLU, DACTPU, UNBIND) from the CSC manager. The VR manager scans the VR_RESERVATION_LIST of each VRCB until it finds the session activation request that corresponds to the session deactivation request. If found, the session activation request is discarded and the session deactivation request is changed to a positive response and returned to the CSC manager. If no corresponding session activation request is found, the session deactivation request is changed to a negative response (X'8005'--No Session) and returned to the CSC manager.

```
VR_ID_LIST_PROCESSOR: PROCEDURE(VR_ID_LIST_INDEX);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO FIND A VR THAT A SESSION CAN BE ASSIGNED TO BY PROCESSING THE
│                VR_ID_LIST ASSOCIATED WITH A SESSION ACTIVATION REQUEST.
│
│                IF A VR FOR THE SESSION ACTIVATION REQUEST IS ALREADY ACTIVE, THIS
│                PROCEDURE:
│
│                     (1) SETS THE GLOBAL POINTER VRCB_PTR TO POINT TO THE VRCB THAT
│                         IS TO BE USED BY PU.SVC_MGR.CSC_MGR (CHAPTER 13)
│
│                     (2) SENDS THE SESSION ACTIVATION REQUEST TO PU.SVC_MGR.CSC_MGR
│                         (CHAPTER 13)
│
│                IF A VR IS PENDING ACTIVATION, THE SESSION ACTIVATION REQUEST IS
│                ANCHORED IN THE VRCB UNTIL ACTIVATION IS COMPLETE. IN THIS CASE,
│                THE PROCEDURE RETURNS TO THE CALLING PROCEDURE.
│
│                IF A VRCB IS NOT ALREADY CREATED FOR THE VR, ONE IS CREATED AND VR
│                ACTIVATION IS ATTEMPTED. IF THE VRCB CAN BE CREATED, THIS
│                PROCEDURE:
│
│                     (1) SETS SPECIFIC FIELDS IN THE VRCB
│
│                     (2) REQUESTS THE ER MANAGER TO ACTIVATE AN ER TO SUPPORT THE VR
│
│                     (3) PLACES THE SESSION ACTIVATION REQUEST, ALONG WITH ITS
│                         VR_ID_LIST, IN THE VRCB.VR_RESERVATION_LIST
│
│                IF A VRCB IS NOT CREATED, THE PROCEDURE ATTEMPTS TO PROCESS THE NEXT
│                VR SPECIFIED IN THE VR_ID_LIST.
│
│                IF A SUITABLE VR CAN NEITHER BE FOUND NOR ACTIVATED, THE SESSION
│                ACTIVATION REQUEST IS CHANGED INTO A NEGATIVE RESPONSE, WITH SENSE
│                CODE X'8013', AND RETURNED TO PU.SVC_MGR.CSC_MGR (CHAPTER 13).
│
│                THIS PROCEDURE IS CALLED BY THE VR_MGR PROCEDURE WITH A
│                VR_ID_LIST_INDEX EQUAL TO 1, AND BY EITHER THE RELEASE_VRCB OR THE
│                CHECK_ER_SUITABILITY PROCEDURE WITH VR_ID_LIST_INDEX GREATER THAN 1.
│
│     INPUT:     ACTCDRM|ACTLU|ACTPU|BIND REQUEST AND VR_ID_LIST ENTITY. THE
│                VR_ID_LIST_INDEX PARAMETER IS AN INDEX TO THE FIRST ENTRY TO BE
│                PROCESSED IN THE VR_ID_LIST.
│
│     OUTPUT:    VRCB_PTR SET TO THE VRCB TO BE USED BY THE SESSION OR
│                -RSP(ACTCDRM|ACTLU|ACTPU|BIND) TO PU.SVC_MGR.CSC_MGR (CHAPTER 13).
│                IF AN ER MUST BE ACTIVATED OR NC_ACTVR MUST BE SENT INTO THE
│                NETWORK, THE SESSION ACTIVATION REQUEST, ALONG WITH ITS VR_ID_LIST,
│                IS ADDED TO THE VR_RESERVATION_LIST ANCHORED IN THE VRCB.
│
│     NOTES:     1. VR_ID_LIST IS DESCRIBED IN CHAPTER 8 AND APPENDIX A.
│
│                2. SENSE CODE X'8013' MEANS THE VR_ID_LIST HAS BEEN EXHAUSTED
│                   WITHOUT FINDING A VR FOR THE SESSION. SENSE CODE X'8013' IS ALSO
│                   RETURNED IF VR_ID_LIST IS EMPTY.
│
│                3. AN IMPLEMENTATION-DEPENDENT LACK OF RESOURCES CAN PREVENT
│                   CREATION OF A VRCB. THE NEXT VRID IN THE VR_ID_LIST IS CHECKED
│                   FOR ASSIGNMENT TO THE SESSION REQUESTED.
│
│                4. FSM_VR IS IN EITHER PEND_RESET STATE OR PEND_RESET_F_SEND STATE.
│                   DO NOT ADD PENDING ACTIVATION REQUESTS TO THE VR_RESERVATION LIST
│                   BECAUSE THE VR MUST BECOME RESET BEFORE IT CAN AGAIN BECOME
│                   ACTIVE AND HAVE SESSIONS ASSIGNED TO IT.
│
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):
│                CHECK_ER_SUITABILITY              PAGE 12-94
│                RELEASE_VRCB                      PAGE 12-117
│                VR_MGR                            PAGE 12-79
│
│     REFERS TO THE FOLLOWING PROCEDURE(S):
│                CHANGE_VRM_MU_TO_NEG_RSP          PAGE 12-118
│                FSM_VR                            PAGE 12-121
│                PROPER_TYPE_OF_VR                 PAGE 12-91
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

```
DCL VR_ID_LIST_INDEX FIXED BIN(8);

DO VR_ID_LIST_INDEX = VR_ID_LIST_INDEX
     BY 1 TO VR_ID_LIST.NUMBER_OF_VR_IDS;                    /* NOTE 1, APPENDIX A      */
.
. FIND VRCB IN VRCB_LIST
.      WHERE(VRCB.PARTNER_SA = DSAP &
.      VRCB.VR_ID = VR_ID_LIST.VR_ID(VR_ID_LIST_INDEX));    /* APPENDIX A              */
.
. IF VRCB_PTR ¬= NULL THEN
.   DO;
.   . CREATE VRCB;
.   .
.   . IF VRCB_PTR ¬= NULL THEN                              /* NOTE 3                  */
.   .   DO;
.   .   . VRCB.VR_ID = VR_ID_LIST.VR_ID(VR_ID_LIST_INDEX);  /* APPENDIX A              */
.   .   . VRCB.PARTNER_SA = DSAP;
.   .   .
.   .   . NEWLIST VRCB.VR_RESERVATION_LIST ENTRY_NAME(VR_RESERVATION);
.   .   .
.   .   . CREATE VR_RESERVATION;                            /* APPENDIX A              */
.   .   . VR_RESERVATION.SESSION_ACT_RQ = MU_PTR;
.   .   . VR_RESERVATION.SCBPTR = SCB_PTR;
.   .   . VR_RESERVATION.VR_LIST = VR_ID_LIST_PTR;          /* APPENDIX A              */
.   .   . VR_RESERVATION.VR_LIST_INDEX = VR_ID_LIST_INDEX;
.   .   .
.   .   . INSERT VR_RESERVATION LAST IN VR_RESERVATION_LIST;
.   .   . INSERT VRCB LAST IN VRCB_LIST;
.   .   .
.   .   . CREATE PARM_ACT_ER;                               /* PAGE 12-126             */
.   .   . PARM_ACT_ER.VRN_MASK = ALL_OFF;                   /* APPENDIX A              */
.   .   . PARM_ACT_ER.VRN_MASK(VRCB.VR_NUM:VRCB.VR_NUM) = ON;
.   .   . PARM_ACT_ER.PARTNER_SA = VRCB.PARTNER_SA;
.   .   .
.   .   . SEND 'ACTIVATE_ER' TO ER_MGR
.   .   .      USING(PARM_PTR = PARM_ACT_ER_PTR);  /* PAGE 12-31, PAGE 12-126      */
.   .   .
.   .   . CALL FSM_VR('ACTIVATE_ER');                       /* PAGE 12-121             */
.   .   . RETURN;
.   .   END;
.   END;
.
. ELSE
.   SELECT ANYORDER;
.   .
.   . WHEN((FSM_VR = ACTIVE) &                              /* PAGE 12-121             */
.   .      PROPER_TYPE_OF_VR(VR_ID_LIST_PTR))               /* PAGE 12-91              */
.   .   DO;
.   .   . SEND MU TO PU.SVC_MGR.CSC_MGR.SEND;               /* CHAPTER 13              */
.   .   . RETURN;
.   .   END;
.   .
.   . WHEN(FSM_VR = PEND_ER |                               /* PAGE 12-121             */
.   .      ((FSM_VR = PEND_ACT_SEND |                       /* PAGE 12-121             */
.   .      FSM_VR = PEND_RESET_O_SEND) &                    /* PAGE 12-121             */
.   .      PROPER_TYPE_OF_VR(VR_ID_LIST_PTR)))              /* PAGE 12-91              */
.   .   DO;
.   .   . CREATE VR_RESERVATION;                            /* APPENDIX A              */
.   .   . VR_RESERVATION.SESSION_ACT_RQ = MU_PTR;
.   .   . VR_RESERVATION.VR_LIST = VR_ID_LIST_PTR;          /* APPENDIX A              */
.   .   . VR_RESERVATION.VR_LIST_INDEX = VR_ID_LIST_INDEX;
.   .   . INSERT VR_RESERVATION LAST IN VR_RESERVATION_LIST;
.   .   . RETURN;
.   .   END;
.   .
.   . OTHERWISE;
.   .                                                       /* NOTE 4                  */
.   END;
END;

CALL CHANGE_VRN_MU_TO_NEG_RSP(X'8013');                     /* NOTE 2, PAGE 12-118     */
SEND MU TO PU.SVC_MGR.CSC_MGR.SEND;                         /* CHAPTER 13              */

RETURN;
END VR_ID_LIST_PROCESSOR;
```

This page intentionally left blank

```
PROPER_TYPE_OF_VR: PROCEDURE(PTR_TO_VR_ID_LIST) RETURNS(BIT(1));
                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │     FUNCTION:  TO DECIDE WHETHER A VR SATISFIES THE RERN REQUIREMENT OF A SESSION.
  │
  │                SOMETIMES A SESSION IS RESTRICTED TO FLOW ONLY ON A VR THAT IS
  │                SUPPORTED BY AN ER THAT HAS A RERN OF 0.  THIS RESTRICTION IS
  │                INDICATED BY THE TYPE_OF_VR FIELD IN THE VR_ID_LIST.  A SESSION MUST
  │                FLOW ON A VR THAT HAS RERN EQUAL TO 0 WHEN:
  │
  │                     (1) IT IS AN LU-LU SESSION,
  │
  │                     (2) THE PLU IS IN THE SUBAREA OF A PU_T4 OR PU_T5 THAT SUPPORTS
  │                         ER-VR PROTOCOLS,
  │
  │                     (3) THE SLU IS IN THE SUBAREA OF A PU_T4 THAT SUPPORTS ER-VR
  │                         PROTOCOLS, AND
  │
  │                     (4) THE SLU HAS AN ACTIVE SESSION WITH AN SSCP THAT DOES NOT
  │                         SUPPORT ER-VR PROTOCOLS.
  │
  │                THIS RERN RESTRICTION ENABLES THE SSCP THAT HAS THE ACTIVE SSCP-LU
  │                SESSION WITH THE SLU TO RECEIVE LOST SUBAREA NOTIFICATION IF THE
  │                LU-LU SESSION IS DISRUPTED BY A VR OUTAGE.
  │
  │     INPUT:     VRCB_PTR, AND VR_ID_LIST ENTITY ADDRESSED BY PTR_TO_VR_ID_LIST
  │                PARAMETER
  │
  │     OUTPUT:    A BIT INDICATOR OF TRUE IF THE RERN REQUIREMENT IS SATISFIED AND OF
  │                FALSE IF THE RERN REQUIREMENT IS NOT SATISFIED
  │
  │     REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                CHECK_ER_SUITABILITY              PAGE 12-94
  │                VR_ID_LIST_PROCESSOR              PAGE 12-88
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL PTR_TO_VR_ID_LIST PTR;

  IF PTR_TO_VR_ID_LIST->VR_ID_LIST.TYPE_OF_VR = RERN_MUST_BE_ZERO &  /* APPENDIX A */
      VRCB.RER_NUM ¬= ZERO THEN
    RETURN(FALSE);
  ELSE
    RETURN(TRUE);

END PROPER_TYPE_OF_VR;
```

```
ER_ACTIVATION_TERMINATOR: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO PROCESS A RESPONSE BY THE ER MANAGER TO AN ACTIVATE_ER SIGNAL  │
│                FROM THE VR MANAGER.  THE ER MANAGER SENDS AN ER_ACTIVATED OR  │
│                ER_NOT_ACTIVATED SIGNAL INDICATING WHETHER THE ER UNDERLYING A  │
│                SPECIFIC VR IS ACTIVE OR INOPERATIVE.  ALL VRCB'S AFFECTED BY THE ER  │
│                STATUS REPORTED ARE PROCESSED.                            │
│                                                                          │
│                WHEN AN ER_ACTIVATED SIGNAL IS RECEIVED FROM THE ER MANAGER AND  │
│                FSM_VR IS IN THE PEND_ER STATE, NC_ACTVR IS SENT INTO THE NETWORK.  │
│                ON THE OTHER HAND, WHEN AN ER_NOT_ACTIVATED SIGNAL IS RECEIVED FROM  │
│                THE ER MANAGER AND FSM_VR IS IN THE PEND_ER STATE, THE VRCB(S) ARE  │
│                REMOVED FROM THE VRCB_LIST AND DISCARDED BY RELEASE_VRCB.  WHEN  │
│                FSM_VR IS IN A STATE OTHER THAN PEND_ER, THE SIGNAL FROM THE ER  │
│                MANAGER PRODUCES NO ACTION.                               │
│                                                                          │
│     INPUT:     AN ER_ACTIVATED OR ER_NOT_ACTIVATED SIGNAL AND PARM_ACT_ER ENTITY  │
│                (ADDRESSED BY PARM_PTR) FROM THE ER MANAGER               │
│                                                                          │
│     OUTPUT:    NC_ACTVR TO PC.ERC (CHAPTER 3) IF THE ER IS ACTIVE, OR DESTROYED  │
│                VRCB IF IT IS NOT NEEDED FURTHER                          │
│                                                                          │
│     NOTES:     1. IN THIS CASE, PACING AND SEQUENCE NUMBERING ARE NOT USED.  │
│                                                                          │
│                2. THE VR_RESERVATION_LIST MAY BE EMPTY BECAUSE EITHER    │
│                   CHECK_ER_SUITABILITY DELETED ENTRIES THAT DID NOT SATISFY THE  │
│                   CLASS OF SERVICE REQUIREMENTS, OR SESSION DEACTIVATION REQUESTS  │
│                   CAUSING THE ENTRIES TO BE DELETED WHILE THE ER MANAGER IS DOING  │
│                   ER ACTIVATION OR WHILE THE NC_ACTVR IS FLOWING.        │
│                                                                          │
│                3. SNF_SEND_CNTR OR SNF_RCV_CNTR FIELDS MAY BE CHANGED TO START AT  │
│                   VALUES OTHER THAN 0.                                   │
│                                                                          │
│                4. THE PARAMETER LIST MAY SPECIFY MULTIPLE VRN'S THAT CAN NOW BE  │
│                   ACTIVATED AND EACH VRN MAY MAP TO UP TO THREE VR'S (ONE PER TPF  │
│                   VALUE).                                                │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                VR_MGR                          PAGE 12-79                │
│                                                                          │
│     REFERS TO THE FOLLOWING PROCEDURE(S):                                │
│                BUILD_ACTVR                     PAGE 12-116               │
│                CHECK_ER_SUITABILITY            PAGE 12-94                │
│                FSM_VR                          PAGE 12-121               │
│                SET_VR_WINDOW_SIZE              PAGE 12-95                │
│                UPM_ALLOW_SNF_OVERRIDE          PAGE 12-103               │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

```
  PARM_ACT_ER_PTR = PARM_PTR;                                        /* PAGE 12-126              */

  SCAN VRCB_LIST PTR(VRCB_PTR);
  . IF (VRCB.PARTNER_SA = PARM_ACT_ER.PARTNER_SA) &                  /* PAGE 12-126              */
  .    (FSM_VR = PEND_ER) &                                          /* PAGE 12-121              */
  .    (PARM_ACT_ER.VRN_MASK(VRCB.VR_NUM:VRCB.VR_NUM) = ON) THEN     /* PAGE 12-126              */
  .   SELECT ANYORDER;
  .   .
  .   . WHEN(INPUT('ER_ACTIVATED'))
  .   .   IF EMPTY(VRCB.VR_RESERVATION_LIST) THEN
  .   .     DO;
  .   .     . DESTROY VRCB.VR_RESERVATION_LIST;
  .   .     . REMOVE VRCB FROM VRCB_LIST DISCARD;
  .   .     END;
  .   .   ELSE
  .   .     DO;
  .   .     . VRCB.ER_NUM = ERCB.ER_NUM;
  .   .     . IF ERCB.ER_VR_SUPP = PRE_ER_VR THEN
  .   .     .   DO;                                                  /* NOTE 1                   */
  .   .     .   . VRCB.ER_VR_SUPP = PRE_ER_VR;
  .   .     .   . CALL FSM_VR('ER_ACTIVATED');                       /* PAGE 12-121              */
  .   .     .   END;
  .   .     . ELSE
  .   .     .   DO;
  .   .     .   . VRCB.RER_NUM = INDEX(ERCB.RERN_MASK,ON);
  .   .     .   . CALL CHECK_ER_SUITABILITY;                         /* PAGE 12-94               */
  .   .     .   . IF  ¬EMPTY(VRCB.VR_RESERVATION_LIST) THEN          /* NOTE 2                   */
  .   .     .   .   DO;
  .   .     .   .   . NEWLIST VRCB.Q_VR_PAC ENTRY_NAME(MU) QUEUE;
  .   .     .   .   . NEWLIST VRCB.UPM_SEGMENTS_LIST ENTRY_NAME(MU) QUEUE;
  .   .     .   .   . VRCB.ER_VR_SUPP =  ¬PRE_ER_VR;
  .   .     .   .   . VRCB.SESS_COUNT = ZERO;
  .   .     .   .   . CALL SET_VR_WINDOW_SIZE;                       /* PAGE 12-95               */
  .   .     .   .   . VRCB.SNF_SEND_CNTR = ZERO;
  .   .     .   .   . VRCB.SNF_RCV_CNTR = ZERO;
  .   .     .   .   . CALL UPM_ALLOW_SNF_OVERRIDE;                   /* NOTE 3, PAGE 12-103      */
  .   .     .   .   . CALL BUILD_ACTVR;                              /* PAGE 12-116              */
  .   .     .   .   . CALL FSM_VR('ER_ACTIVATED');                   /* PAGE 12-121              */
  .   .     .   .   END;
  .   .     .   . ELSE
  .   .     .   .   DO;
  .   .     .   .   . DESTROY VRCB.VR_RESERVATION_LIST;
  .   .     .   .   . REMOVE VRCB FROM VRCB_LIST DISCARD;
  .   .     .   .   END;
  .   .     .   END;
  .   .     END;
  .   .
  .   . WHEN(INPUT('ER_NOT_ACTIVATED'))
  .   .   CALL FSM_VR('ER_NOT_ACTIVATED');                           /* PAGE 12-121              */
  .   END;
  SCANEND;

  DISCARD PARM_ACT_ER;                                               /* PAGE 12-126              */

  RETURN;
END ER_ACTIVATION_TERMINATOR;
```

```
CHECK_ER_SUITABILITY: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO DETERMINE WHETHER THE RERN OF THE ER IS SUITABLE  │
  │                 FOR THE SESSION.                                     │
  │                                                                      │
  │                 SOMETIMES A SESSION IS RESTRICTED TO USE ONLY A VR   │
  │                 THAT IS SUPPORTED BY AN ER THAT HAS AN RERN OF 0,    │
  │                 WHICH ENABLES THE SSCP THAT HAS THE ACTIVE SSCP-LU   │
  │                 SESSION WITH THE SLU TO RECEIVE LOST SUBAREA         │
  │                 NOTIFICATION IF THE LU-LU SESSION IS DISRUPTED BY A  │
  │                 VR OUTAGE.  THIS RESTRICTION IS INDICATED BY A BIT   │
  │                 SETTING IN THE VR_ID_LIST AND IS CHECKED BY          │
  │                 PROPER_TYPE_OF_VR.                                   │
  │                                                                      │
  │                 IF THE RERN IS NOT SUITABLE FOR THE SESSION, THE     │
  │                 SESSION ACTIVATION REQUEST AND ITS ASSOCIATED        │
  │                 VR_ID_LIST ARE REMOVED FROM THE VR_RESERVATION_LIST; │
  │                 AN ATTEMPT WILL BE MADE TO ASSIGN THE SESSION TO A   │
  │                 SUBSEQUENT VR SPECIFIED IN THE VR_ID_LIST.           │
  │                                                                      │
  │      INPUT:     VRCB_PTR                                             │
  │                                                                      │
  │      OUTPUT:    POSSIBLY REDUCED VR_RESERVATION LIST IF THE ER TO BE  │
  │                 USED TO SUPPORT THE VR IS NOT SUITABLE; -RSP(ACTCDRM|│
  │                 ACTLU|ACTPU|BIND) WITH SENSE CODE X'8013' TO         │
  │                 PU.SVC_MGR.CSC_MGR (CHAPTER 13) FOR ANY SESSION      │
  │                 ACTIVATION REQUEST WHOSE VR_ID_LIST HAS BEEN         │
  │                 EXHAUSTED WITHOUT PRODUCING AN ACTIVE VR (ACTUALLY   │
  │                 SENT BY VR_ID_LIST_PROCESSOR).                       │
  │                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
  │                 ACTVR_RQ_RCV                 PAGE 12-100             │
  │                 ER_ACTIVATION_TERMINATOR     PAGE 12-92              │
  │                                                                      │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
  │                 PROPER_TYPE_OF_VR            PAGE 12-91              │
  │                 VR_ID_LIST_PROCESSOR         PAGE 12-88              │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL NEXT_VR_ID_LIST_INDEX FIXED BIN(8);
  DCL SAVED_VRCB_PTR POINTER;
  DCL SAVED_MU_PTR POINTER;

  SAVED_MU_PTR = MU_PTR;

  SCAN VRCB.VR_RESERVATION_LIST PTR(VR_RESERVATION_PTR);
  . IF ¬PROPER_TYPE_OF_VR(VR_RESERVATION.VR_LIST) THEN          /* PAGE 12-91      */
  .    DO;
  .    . MU_PTR = VR_RESERVATION.SESSION_ACT_RQ;              /* APPENDIX A      */
  .    . SCB_PTR = VR_RESERVATION.SCBPTR;                     /* APPENDIX A      */
  .    . VR_ID_LIST_PTR = VR_RESERVATION.VR_LIST;             /* APPENDIX A      */
  .    . NEXT_VR_ID_LIST_INDEX = VR_RESERVATION.VR_LIST_INDEX + 1;
  .    .
  .    . REMOVE VR_RESERVATION FROM VR_RESERVATION_LIST DISCARD;
  .    .
  .    . SAVED_VRCB_PTR = VRCB_PTR;
  .    . CALL VR_ID_LIST_PROCESSOR(NEXT_VR_ID_LIST_INDEX);   /* PAGE 12-88      */
  .    . VRCB_PTR = SAVED_VRCB_PTR;
  .    END;
  SCANEND;

 MU_PTR = SAVED_MU_PTR;

  RETURN;
END CHECK_ER_SUITABILITY;
```

```
SET_VR_WINDOW_SIZE: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO COMPUTE THE MINIMUM AND MAXIMUM  VR WINDOW SIZE VALUES BASED UPON
  │                 ER LENGTH.                                                   │
  │                                                                              │
  │      INPUT:     VRCB_PTR AND ERCB_PTR                                        │
  │                                                                              │
  │      OUTPUT:    VR WINDOW SIZE VALUES IN VRCB                                │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 ER_ACTIVATION_TERMINATOR          PAGE 12-92                 │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 UPM_VR_WINDOW_SIZE_OVERRIDE       PAGE 12-103                │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  VRCB.MAX_WINDOW_SIZE = 3 * ERCB.ER_LEN;
  VRCB.MIN_WINDOW_SIZE = ERCB.ER_LEN;
  VRCB.WINDOW_SIZE = VRCB.MIN_WINDOW_SIZE;
  VRCB.WINDOW_SIZE_CHANGE = ONE;
  CALL UPM_VR_WINDOW_SIZE_OVERRIDE;                          /* PAGE 12-103    */

  RETURN;
END SET_VR_WINDOW_SIZE;
```

```
ACTVR_RCV: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│     FUNCTION:  TO PROCESS A  RECEIVED NC_ACTVR REQUEST OR  RESPONSE.  THE PROCEDURE │
│                FINDS THE VRCB THAT IS TO BE PROCESSED, CALLS FSM_VR, AND DETERMINES │
│                WHETHER THE  NC_ACTVR IS A  REQUEST, POSITIVE RESPONSE,  OR NEGATIVE │
│                RESPONSE.                                                 │
│                                                                          │
│                FOR A REQUEST,  THE PROCEDURE ALLOCATES A VRCB,  IF NECESSARY.  WHEN │
│                STORAGE  IS  NOT  AVAILABLE,  A -RSP(X'0812')  IS  RETURNED  TO  THE │
│                NC_ACTVR  ORIGINATOR.  IF  THE  SET OF  OPTIONAL  RECEIVE CHECKS  IN │
│                VR_RCV_CHECKS FAILS,  A -RSP(NC_ACTVR) IS  RETURNED AND THE  VRCB IS │
│                RELEASED IF IT  DID NOT ALREADY EXIST BEFORE  THIS NC_ACTVR ARRIVED. │
│                THE CALL TO FSM_VR RESOLVES AN NC_ACTVR RACE, IF REQUIRED.  THE CALL │
│                TO ACTVR_RQ_RCV SETS FIELDS IN THE  VRCB BASED ON INFORMATION IN THE │
│                NC_ACTVR RU,  AND CHANGES THE REQUEST  INTO A POSITIVE  RESPONSE AND │
│                RETURNS IT TO ITS ORIGINATOR.                             │
│                                                                          │
│                FOR A  POSITIVE RESPONSE, ALL  SESSION ACTIVATION REQUESTS  THAT ARE │
│                PENDING ACTIVATION OF THE VR ARE  NOW MADE AVAILABLE TO THE CSC_MGR. │
│                IF THERE ARE NO SESSIONS WAITING FOR ACTIVATION OF THE VR, NC_DACTVR │
│                (ORDERLY) IS SENT  TO THE SENDER OF  +RSP(NC_ACTVR).  THIS PROCEDURE │
│                DISCARDS THE +RSP(NC_ACTVR).                              │
│                                                                          │
│                FOR A NEGATIVE RESPONSE, THIS PROCEDURE SETS FSM_DACTVR_DIRECTION TO │
│                REFLECT THE RECEIPT OF THE  -RSP(NC_ACTVR), AND DISCARDS THE MESSAGE │
│                UNIT.                                                     │
│                                                                          │
│     INPUT:     NC_ACTVR (REQUEST OR RESPONSE)                            │
│                                                                          │
│     OUTPUT:    WHEN AN NC_ACTVR REQUEST IS THE INPUT, EITHER AN ALLOCATED VRCB WITH │
│                FSM_VR IN ACTIVE  STATE AND A +RSP(NC_ACTVR) TO  PC.ERC (CHAPTER 3), │
│                OR A  -RSP(NC_ACTVR) TO  PC.ERC (CHAPTER  3), IS  THE OUTPUT.  WHEN │
│                +RSP(NC_ACTVR)  IS  THE  INPUT,  ACTCDRM|ACTLU|ACTPU|BIND  TO │
│                PU.SVC_MGR.CSC_MGR  (CHAPTER 13),  WITH  VRCB_PTR  SET TO  THE  VRCB │
│                (FSM_VR IS IN ACTIVE STATE) TO WHICH THE SESSION IS ASSIGNED, IS THE │
│                OUTPUT.                                                   │
│                                                                          │
│     NOTES:     1. INPUT TO FSM_VR IS ONE OF THE FOLLOWING:               │
│                                                                          │
│                          (R,RQ,NC_ACTVR,¬LOWER_SA) OR (R,RQ,NC_ACTVR,LOWER_SA) │
│                          (R,+RSP,NC_ACTVR)                               │
│                          (R,-RSP,NC_ACTVR)                               │
│                                                                          │
│                2. AN IMPLEMENTATION LACK OF RESOURCES PREVENTS CREATION OF A VRCB. │
│                                                                          │
│                3. SENSE CODE  X'0812' MEANS  NO STORAGE  IS AVAILABLE  TO CREATE  A │
│                   VRCB.                                                 │
│                                                                          │
│                4. FSM_VRPRQ_RCV  IS  INITIALIZED  TO  ALLOW  THE  FIRST  VR  PACING │
│                   RESPONSE TO BE SENT.                                  │
│                                                                          │
│                5. +RSP(NC_ACTVR) CARRIES THE FIRST VRPRS TO BE RECEIVED BY THIS END │
│                   OF THE VR.  FSM'S AND THE PACING COUNT ARE SET TO REFLECT RECEIPT │
│                   OF THE VRPRS BY THIS END OF THE  VR, AND TO ALLOW THE FIRST VRPRS │
│                   FROM THIS END OF THE VR TO  BE SENT.  THE INPUT TO FSM_VRPRQ_SEND │
│                   IS VR_PAC_RSP.                                        │
│                                                                          │
│                6. IF THE  ER SUPPORTING  THE VR  BECAME INOPERATIVE,  THE VR  TO ER │
│                   MAPPING MIGHT  HAVE BEEN CHANGED  AND THEN ANOTHER  NC_ACTVR SENT │
│                   OUT  ON  A  DIFFERENT  ER.  THIS  CHECK  ASCERTAINS  WHETHER  THE │
│                   RSP(ACTVR) USES THE ER SPECIFIED IN THE VRCB.         │
│                                                                          │
│     REFERENCED BY THE FOLLOWING PROCEDURE(S):                            │
│                VR_MGR                              PAGE 12-79            │
│                                                                          │
│     REFERS TO THE FOLLOWING PROCEDURE(S):                                │
│                ACTVR_RQ_RCV                        PAGE 12-100           │
│                CHANGE_ACTVR_TO_NEG_RSP             PAGE 12-99            │
│                FSM_DACTVR_DIRECTION                PAGE 12-122           │
│                FSM_ERN                             PAGE 12-73            │
│                FSM_VR                              PAGE 12-121           │
│                SEND_DACTVR_ORDERLY                 PAGE 12-106           │
│                UPM_SEND_VRPRS                      PAGE 12-103           │
│                VR_ACTIVATED                        PAGE 12-101           │
│                VR_RCV_CHECKS                       PAGE 12-98            │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */

DCL SAVED_MU_PTR PTR;
```

```
FIND VRCB IN VRCB_LIST
     WHERE(VRCB.PARTNER_SA = OSAF & VRCB.VR_ID = VRID);

SELECT ANYORDER;
. WHEN(INPUT(RQ))
.   IF VRCB_PTR = NULL THEN
.     DO;
.     . CREATE VRCB;
.     . IF VRCB_PTR = NULL THEN
.     .   DO;                                              /* NOTE 2                  */
.     .   . CALL CHANGE_ACTVR_TO_NEG_RSP(X'0812');         /* NOTE 3                  */
.     .   . SEND MU TO PC.ERC;                             /* CHAPTER 3               */
.     .   END;
.     . ELSE
.     .   DO;
.     .   . INSERT VRCB IN VRCB_LIST;
.     .   . VRCB.VR_ID = VRID;
.     .   . VRCB.PARTNER_SA = OSAF;
.     .   . IF VR_RCV_CHECKS = NG THEN                     /* PAGE 12-98              */
.     .   .   REMOVE VRCB FROM VRCB_LIST DISCARD;
.     .   . ELSE
.     .   .   DO;
.     .   .   . NEWLIST VRCB.VR_RESERVATION_LIST ENTRY_NAME(VR_RESERVATION);
.     .   .   . CALL FSM_VR;                               /* NOTE 1, PAGE 12-121     */
.     .   .   END;
.     .   END;
.     END;
.   ELSE
.     IF VR_RCV_CHECKS = OK THEN                           /* PAGE 12-98              */
.       CALL FSM_VR;                                       /* NOTE 1, PAGE 12-121     */
.
. WHEN(INPUT(RSP) & RTI = POSITIVE)
.   DO;
.   . IF VRCB_PTR ¬= NULL THEN
.   .   DO;
.   .   . FIND ERCB IN ERCB_LIST
.   .   .      WHERE(ERCB.PARTNER_SA = VRCB.PARTNER_SA & ERCB.ER_NUM = VRCB.ER_NUM);
.   .   . IF FSM_ERN = ACTIVE &                           /* PAGE 12-73              */
.   .   .      ERCB.RERN_MASK(ERN:ERN) = ON THEN          /* NOTE 6                  */
.   .   .   DO;
.   .   .   . CALL FSM_VR;                                /* NOTE 1, PAGE 12-121     */
.   .   .   . IF FSM_VR = ACTIVE THEN                     /* PAGE 12-121             */
.   .   .   .   IF ¬EMPTY(VRCB.VR_RESERVATION_LIST) THEN
.   .   .   .     DO;
.   .   .   .     . CALL FSM_DACTVR_DIRECTION;            /* PAGE 12-122             */
.   .   .   .     . CALL VR_ACTIVATED;                    /* PAGE 12-101             */
.   .   .   .     . CALL FSM_SET_CWRI('RESET');           /* NOTE 5, CHAPTER 3       */
.   .   .   .     . CALL FSM_VRPRQ_SEND;                  /* NOTE 5, CHAPTER 3       */
.   .   .   .     . CALL FSM_VRPRQ_RCV('FIRST_VRPRS');    /* NOTE 4, CHAPTER 3       */
.   .   .   .     . CALL UPM_SEND_VRPRS;                  /* NOTE 4, PAGE 12-103     */
.   .   .   .     . VRCB.PACING_COUNT = VRCB.WINDOW_SIZE; /* NOTE 5                  */
.   .   .   .     END;
.   .   .   .   ELSE
.   .   .   .     DO;
.   .   .   .     . SAVED_MU_PTR = MU_PTR;
.   .   .   .     . CALL SEND_DACTVR_ORDERLY;             /* PAGE 12-106             */
.   .   .   .     . MU_PTR = SAVED_MU_PTR;
.   .   .   .     END;
.   .   .   END;
.   .   END;
.   . DISCARD MU;
.   END;
.
. WHEN(INPUT(RSP) & RTI = NEGATIVE)
.   DO;
.   . IF VRCB_PTR ¬= NULL THEN
.   .   DO;
.   .   . FIND ERCB IN ERCB_LIST
.   .   .      WHERE(ERCB.PARTNER_SA = VRCB.PARTNER_SA & ERCB.ER_NUM = VRCB.ER_NUM);
.   .   . IF FSM_ERN = ACTIVE &                           /* PAGE 12-73              */
.   .   .      ERCB.RERN_MASK(ERN:ERN) = ON THEN          /* NOTE 6                  */
.   .   .   DO;
.   .   .   . CALL FSM_DACTVR_DIRECTION;                  /* PAGE 12-122             */
.   .   .   . CALL FSM_VR;                                /* NOTE 1, PAGE 12-121     */
.   .   .   END;
.   .   END;
.   . DISCARD MU;
.   END;
END;

  RETURN;
END ACTVR_RCV;
```

```
VR_RCV_CHECKS: PROCEDURE RETURNS(BIT(1));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO PERFORM OPTIONAL CHECKS FOR THE RECEIVER OF AN NC_ACTVR.  │
   │                                                                              │
   │      INPUT:     NC_ACTVR REQUEST AND VRCB_PTR                                │
   │                                                                              │
   │      OUTPUT:    OK OR NG, INDICATING WHETHER THE  NC_ACTVR IS ACCEPTABLE OR NOT.  IF
   │                 THE NC_ACTVR  IS NOT ACCEPTABLE,  A -RSP(NC_ACTVR)  WITH APPROPRIATE
   │                 SENSE  CODE IS  SENT  BACK TO  THE  NC_ACTVR  ORIGINATOR VIA  PC.ERC
   │                 (CHAPTER 3).                                                 │
   │                                                                              │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
   │                 ACTVR_RCV                          PAGE 12-96                │
   │                                                                              │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
   │                 ABLE_TO_RCV_ACTVR                  PAGE 12-71                │
   │                 CHANGE_ACTVR_TO_NEG_RSP            PAGE 12-99                │
   │                 VRN_TO_ERN_MAP                     PAGE 12-124               │
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL ER_NUM BIT(4);

  IF VRN_TO_ERN_MAP(VRCB.PARTNER_SA,VRCB.VR_NUM,ER_NUM) = ¬EXIST THEN /* PAGE 12-124  */
    DO;                                 /* VIRTUAL ROUTE NOT DEFINED                */
    . CALL CHANGE_ACTVR_TO_NEG_RSP(X'0873');                    /* PAGE 12-99       */
    . SEND MU TO PC.ERC;                                        /* CHAPTER 3        */
    . RETURN(NG);
    END;

  IF NC_ACTVR_RQ.RCV_ERN_MASK(ER_NUM:ER_NUM) = OFF THEN
    DO;                                 /* INCORRECT OR UNDEFINED ER REQUESTED      */
    . CALL CHANGE_ACTVR_TO_NEG_RSP(X'0875');                    /* PAGE 12-99       */
    . SEND MU TO PC.ERC;                                        /* CHAPTER 3        */
    . RETURN(NG);
    END;


  FIND ERCB IN ERCB_LIST WHERE(ERCB.PARTNER_SA = VRCB.PARTNER_SA & ERCB.ER_NUM = ER_NUM);


  IF ERCB_PTR = NULL  |  ¬ABLE_TO_RCV_ACTVR  THEN               /* PAGE 12-71       */
    DO;                                 /* ER NOT IN A VALID STATE                  */
    . CALL CHANGE_ACTVR_TO_NEG_RSP(X'0874');                    /* PAGE 12-99       */
    . SEND MU TO PC.ERC;                                        /* CHAPTER 3        */
    . RETURN(NG);
    END;


  IF ERCB.RERN_MASK(ERN:ERN) = OFF THEN
    DO;                                 /* NONREVERSIBLE ER REQUESTED               */
    . CALL CHANGE_ACTVR_TO_NEG_RSP(X'0876');                    /* PAGE 12-99       */
    . SEND MU TO PC.ERC;                                        /* CHAPTER 3        */
    . RETURN(NG);
    END;

  RETURN(OK);
END VR_RCV_CHECKS;
```

```
CHANGE_ACTVR_TO_NEG_RSP: PROCEDURE(SENSE_CODE);

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO CHANGE AN NC_ACTVR REQUEST TO A  -RSP(NC_ACTVR) AND SET UP THE TH
  │                 PROPERLY
  │
  │      INPUT:     NC_ACTVR REQUEST, AND SENSE_CODE PARAMETER  GIVING THE SENSE CODE TO
  │                 BE PUT INTO THE NEGATIVE RESPONSE
  │
  │      OUTPUT:    -RSP(NC_ACTVR)  WITH APPROPRIATE  SENSE  CODE.   THE INPUT  NC_ACTVR
  │                 REQUEST UNIT IS OVERWRITTEN BY THE RESPONSE UNIT.
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 ACTVR_RCV                          PAGE 12-96
  │                 FSM_VR                             PAGE 12-121
  │                 VR_RCV_CHECKS                      PAGE 12-98
  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):
  │                 SWAP_ORIGIN_DEST                   PAGE 12-119
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL SENSE_CODE BIT(32);

  CALL CHANGE_MU_TO_NEG_RSP(SENSE_CODE);                  /* APPENDIX B    */

  CALL SWAP_ORIGIN_DEST;                                  /* PAGE 12-119   */

  ERN = INDEX(NC_ACTVR_RQ.RCV_ERN_MASK,ON);

  RETURN;
END CHANGE_ACTVR_TO_NEG_RSP;
```

```
ACTVR_RQ_RCV: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO  SET SPECIFIC  FIELDS  IN A  VRCB,  SOME OF  WHICH  ARE BASED  ON  │
 │                 INFORMATION  OBTAINED FROM  THE NC_ACTVR  REQUEST, AND  TO SEND  THE  │
 │                 +RSP(NC_ACTVR).  THIS PROCEDURE  IS EXECUTED ONLY BY  THE VR MANAGER  │
 │                 THAT RECEIVES AN NC_ACTVR REQUEST.                   │
 │                                                                      │
 │      INPUT:     NC_ACTVR AND VRCB_PTR                                │
 │                                                                      │
 │      OUTPUT:    UPDATED VRCB, AND NC_ACTVR TO PC.ERC (CHAPTER 3)     │
 │                                                                      │
 │      NOTES:     1. SINCE PROCESSING  PASSED VR_RCV_CHECKS  SUCCESSFULLY, THE  VRN TO  │
 │                    ERN MAPPING MUST EXIST.                           │
 │                                                                      │
 │                 2. INPUT TO FSM_DACTVR_DIRECTION IS (S,+RSP,ACTVR).  │
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 ACTVR_RCV                       PAGE 12-96           │
 │                 FSM_VR                          PAGE 12-121          │
 │                                                                      │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
 │                 CHANGE_VRN_MU_TO_POS_RSP        PAGE 12-118          │
 │                 CHECK_ER_SUITABILITY            PAGE 12-94           │
 │                 FSM_DACTVR_DIRECTION            PAGE 12-122          │
 │                 VR_ACTIVATED                    PAGE 12-101          │
 │                 VRN_TO_ERN_MAP                  PAGE 12-124          │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  EXIST = VRN_TO_ERN_MAP(VRCB.PARTNER_SA,VRCB.VR_NUM,VRCB.ER_NUM);  /* NOTE 1, PAGE 12-124   */
  VRCB.RER_NUM = ERN;

  NEWLIST VRCB.Q_VR_PAC ENTRY_NAME(MU) QUEUE;
  NEWLIST VRCB.UPM_SEGMENTS_LIST ENTRY_NAME(MU) QUEUE;

  VRCB.MAX_WINDOW_SIZE = NC_ACTVR_RQ.MAX_WINDOW_SIZE;
  VRCB.MIN_WINDOW_SIZE = NC_ACTVR_RQ.MIN_WINDOW_SIZE;
  VRCB.WINDOW_SIZE = NC_ACTVR_RQ.MIN_WINDOW_SIZE;
  VRCB.WINDOW_SIZE_CHANGE = ONE;

  VRCB.PACING_COUNT = ZERO;

  VRCB.ER_VR_SUPP =  ¬PRE_ER_VR;

  VRCB.SESS_COUNT = ZERO;

  VRCB.SNF_SEND_CNTR = NC_ACTVR_RQ.VR_SEND_SEQ_NO;
  VRCB.SNF_RCV_CNTR = NC_ACTVR_RQ.VR_SEND_SEQ_NO;

  CALL CHECK_ER_SUITABILITY;                          /* PAGE 12-94            */
  IF  ¬EMPTY(VRCB.VR_RESERVATION_LIST) THEN
    CALL VR_ACTIVATED;                                /* PAGE 12-101           */

  CALL CHANGE_VRN_MU_TO_POS_RSP(TRUNCATE);            /* PAGE 12-118           */
  VRPRS = VR_PAC_RSP;
  CALL FSM_DACTVR_DIRECTION;                          /* NOTE 2, PAGE 12-122   */
  SEND MU TO PC.ERC;                                  /* CHAPTER 3             */


  RETURN;
END ACTVR_RQ_RCV;
```

```
VR_ACTIVATED: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO PROCESS PENDING SESSION ACTIVATION REQUESTS WHEN  │
 │                 ACTIVATED TO SUPPORT THE SESSIONS. IF THERE ARE ANY  │
 │                 ACTIVATION REQUESTS AWAITING THE ACTIVATION OF THE VR│
 │                 RETURNED TO PU.SVC_MGR.CSC_MGR (CHAPTER 13).         │
 │                                                                      │
 │      INPUT:     VRCB_PTR                                             │
 │                                                                      │
 │      OUTPUT:    ACTDCRM|ACTLU|ACTPU|BIND TO PU.SVC_MGR.CSC_MGR (CHAPTER 13) WITH
 │                 VRCB_PTR SET TO THE VRCB TO WHICH THE SESSION IS ASSIGNED
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 ACTVR_RCV                       PAGE 12-96           │
 │                 ACTVR_RQ_RCV                    PAGE 12-100          │
 │                 DACTVR_RCV                      PAGE 12-108          │
 │                 FSM_VR                          PAGE 12-121          │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL SAVED_MU_PTR PTR;

  SAVED_MU_PTR = MU_PTR;

  SCAN VRCB.VR_RESERVATION_LIST PTR(VR_RESERVATION_PTR);
  .
  . MU_PTR = VR_RESERVATION.SESSION_ACT_RQ;          /* APPENDIX A        */
  . SCB_PTR = VR_RESERVATION.SCBPTR;                 /* APPENDIX A        */
  .
  . DISCARD VR_RESERVATION.VR_LIST->VR_ID_LIST;      /* APPENDIX A        */
  .
  . REMOVE VR_RESERVATION FROM VR_RESERVATION_LIST DISCARD;  /* APPENDIX A */
  .
  . SEND MU TO PU.SVC_MGR.CSC_MGR.SEND;              /* CHAPTER 13        */
  .
  SCANEND;

  MU_PTR = SAVED_MU_PTR;

  RETURN;
END VR_ACTIVATED;
```

```
CANCEL_VR_RESERVATION: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO CANCEL THE ASSIGNMENT TO A VR OF A SESSION DURING SESSION │
   │                 ACTIVATION.  THIS PROCEDURE SEARCHES THROUGH THE SESSION ACTIVATION │
   │                 REQUESTS AWAITING VR ACTIVATION.  IF THE SESSION ACTIVATION REQUEST │
   │                 THAT CORRESPONDS TO THE INPUT SESSION DEACTIVATION REQUEST IS FOUND, │
   │                 THE SESSION ACTIVATION REQUEST IS DISCARDED, AND THE SESSION │
   │                 DEACTIVATION REQUEST IS CHANGED TO A +RSP, THEN SENT TO │
   │                 PU.SVC_MGR.CSC_MGR (CHAPTER 13); OTHERWISE, THE SESSION DEACTIVATION │
   │                 REQUEST IS CHANGED TO A -RSP AND SENT TO PU.SVC_MGR.CSC_MGR (CHAPTER │
   │                 13).                                                 │
   │                                                                      │
   │      INPUT:     DACTCDRM|DACTLU|DACTPU|UNBIND REQUEST                │
   │                                                                      │
   │      OUTPUT:    EITHER -RSP(DACTCDRM|DACTLU|DACTPU|UNBIND) WITH SENSE CODE X'8005' │
   │                 OR     +RSP(DACTCDRM|DACTLU|DACTPU|UNBIND) TO PU.SVC_MGR.CSC_MGR │
   │                 (CHAPTER 13)                                         │
   │                                                                      │
   │      NOTE:      SENSE CODE X'8005' MEANS NO SESSION ACTIVATION REQUEST WAS FOUND FOR │
   │                 THE SESSION DEACTIVATION REQUEST.                    │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 VR_MGR                          PAGE 12-79           │
   │                                                                      │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
   │                 CHANGE_VRM_MU_TO_NEG_RSP        PAGE 12-118          │
   │                 CHANGE_VRM_MU_TO_POS_RSP        PAGE 12-118          │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  SCAN VRCB_LIST PTR(VRCB_PTR);
  .
  . FIND VR_RESERVATION IN VRCB.VR_RESERVATION_LIST
  .     WHERE(VR_RESERVATION.SCBPTR = SCB_PTR);          /* APPENDIX A      */
  .
  . IF VR_RESERVATION_PTR ¬= NULL THEN
  .   DO;                          /* FOUND CORRESPONDING ENTRY             */
  .    . DISCARD VR_RESERVATION.SESSION_ACT_RQ->MU;
  .    . DISCARD VR_RESERVATION.VR_LIST->VR_ID_LIST;
  .    . REMOVE VR_RESERVATION FROM VR_RESERVATION_LIST DISCARD;
  .    . CALL CHANGE_VRM_MU_TO_POS_RSP(TRUNCATE);       /* PAGE 12-118     */
  .    . SEND MU TO PU.SVC_MGR.CSC_MGR.SEND;            /* CHAPTER 13      */
  .    . RETURN;
  .   END;
  .
  SCANEND;

  CALL CHANGE_VRM_MU_TO_NEG_RSP(X'8005');               /* NOTE, PAGE 12-118 */
  SEND MU TO PU.SVC_MGR.CSC_MGR.SEND;                   /* CHAPTER 13      */

  RETURN;
END CANCEL_VR_RESERVATION;



UPM_VR_ID_LIST_REORDER: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE IS AN INSTALLATION-DEFINED UPM THAT ALLOWS │
   │                 REORDERING, ADDING TO, OR DELETING FROM THE VR_ID_LIST, WHICH │
   │                 RESULTS FROM CLASS OF SERVICE NAME RESOLUTION FOR THIS SESSION │
   │                 ACTIVATION REQUEST.  THE NUMBER OF ENTRIES IN THE LIST AFTER THE │
   │                 MODIFICATIONS ARE MADE ARE THE SAME AS OR LESS THAN THE NUMBER OF │
   │                 ENTRIES BEFORE THE MODIFICATIONS.  NO MODIFICATIONS ARE ALLOWED FOR │
   │                 SSCP-BASED SESSIONS.                                 │
   │                                                                      │
   │      INPUT:     VR_ID_LIST                                           │
   │                                                                      │
   │      OUTPUT:    AN ALTERED VR_ID_LIST                                │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 VR_MGR                          PAGE 12-79           │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  RETURN;
END UPM_VR_ID_LIST_REORDER;
```

```
UPM_ALLOW_SNF_OVERRIDE: PROCEDURE:

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE  IS  AN INSTALLATION-DEFINED  UPM  THAT  ALLOWS  THE │
   │                 SNF_SEND_CNTR OR SNF_RCV_CNTR  FIELDS OF A VRCB TO BE  SET TO VALUES │
   │                 OTHER THAN 0.                                        │
   │                                                                      │
   │      INPUT:     VRCB_PTR                                             │
   │                                                                      │
   │      OUTPUT:    POSSIBLY ALTERED SNF_SEND_CNTR OR SNF_RCV_CNTR FIELDS IN VRCB │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 DACTVR_RCV                          PAGE 12-108      │
   │                 ER_ACTIVATION_TERMINATOR            PAGE 12-92       │
   │                 FSM_VR                              PAGE 12-121      │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  RETURN;
END UPM_ALLOW_SNF_OVERRIDE;



UPM_VR_WINDOW_SIZE_OVERRIDE: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS   PROCEDURE  IS   AN  INSTALLATION-DEFINED   UPM  THAT   ALLOWS │
   │                 MODIFICATION OF THE VR PACING  VALUES BASED ON INSTALLATION-SUPPLIED │
   │                 ALGORITHMS.                                          │
   │                                                                      │
   │      INPUT:     VRCB_PTR AND ERCB_PTR                                │
   │                                                                      │
   │      OUTPUT:    POSSIBLY ALTERED VRCB PACING VALUES                  │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 SET_VR_WINDOW_SIZE                  PAGE 12-95       │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  RETURN;
END UPM_VR_WINDOW_SIZE_OVERRIDE;



UPM_SEND_VRPRS: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE IS  AN INSTALLATION-DEFINED  UPM  THAT TRIGGERS  THE │
   │                 SENDING OF THE FIRST VR PACING RESPONSE FROM THIS END OF THE VR.      │
   │                                                                      │
   │      INPUT:     RSP(NC_ACTVR) AND VRCB_PTR                           │
   │                                                                      │
   │      OUTPUT:    INITIALIZED FSM_VRPRQ_RCV                            │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 ACTVR_RCV                           PAGE 12-96       │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  RETURN;
END UPM_SEND_VRPRS;
```

VIRTUAL ROUTE DEACTIVATION

A VR is deactivated normally after the last session assigned to the VR is deactivated. Normal deactivation is accomplished by a NC_DACTVR request and response flowing over the VR.

A VR is also deactivated when the underlying ER becomes inoperative. In this case it is not possible to send NC_DACTVR over the VR. Instead, the VR managers at each end of the ER recognize that the VR is inoperative, cause session outage notification to be sent on the active sessions assigned to the VR, and, when the session outage notification is complete, deactivate the VR by destroying the VRCB.

DEACTIVATE VIRTUAL ROUTE (NC_DACTVR)

Flow: VR manager to VR manager (Expedited), with TG Sweep = SWEEP, at the transmission priority of the VR

Principal FSMs:

NC_DACTVR deactivates only those VRs that are not entirely within one subarea, and that contain only nodes that support explicit and virtual routes. There are two types of NC_DACTVR: Orderly and Forced.

At any given time, only one of the VR managers controlling the VR can send NC_DACTVR(Orderly). If NC_DACTVR has never been sent on the VR, the VR manager that sent NC_ACTVR is allowed to send NC_DACTVR. If NC_DACTVR has been sent, the VR manager that sent the latest negative response to NC_DACTVR is allowed to send NC_DACTVR.

The VR manager sends NC_DACTVR(Orderly) when the CSC manager signals it that there are no longer any sessions assigned to the VR. Upon receiving NC_DACTVR(Orderly), a VR manager responds positively or negatively. A negative response indicates that this VR manager has one or more sessions assigned to the VR, and causes the VR to remain active; a positive response indicates that this VR manager has no sessions assigned to the VR and causes the VR to be reset.

After sending NC_DACTVR(Orderly), but before receiving its response, the VR manager may process a session activation request for assignment to the VR. In this case the session activation request is placed in the VR_RESERVATION list of the VRCB. If a negative NC_ACTVR response is received, the VR is active and the session is assigned to it. If a positive NC_ACTVR response is received, the VR is reset and NC_ACTVR is sent to re-activate the VR; the session activation request is retained in VR_RESERVATION list until the NC_ACTVR response is received.

Either of the VR managers controlling the VR is allowed to send NC_DACTVR(Forced). The VR manager sends NC_DACTVR(Forced) to reset the VR unconditionally, and only when there are no longer any active sessions assigned to the VR. Upon receiving NC_DACTVR(Forced) a VR manager responds positively. If the receiving VR manager has any sessions assigned to the VR, it signals CSC manager to perform session outage notification, and when all sessions assigned to the VR are terminated, sends a positive response.

```
SEND_DACTVR_ORDERLY: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO PROCESS  A SESS_COUNT=0  SIGNAL FROM  PU.SVC_MGR.CSC_MGR (CHAPTER
 │                 13).  THIS  INPUT SPECIFIES  THAT THERE ARE  NO LONGER  ANY SESSIONS
 │                 ASSIGNED  TO  THE VIRTUAL  ROUTE;  IF  OTHER CONDITIONS  ALLOW,  THE
 │                 VIRTUAL ROUTE IS RESET BY SENDING NC_DACTVR(ORDERLY).
 │
 │                 • IF THE VR IS LOCAL TO THIS SUBAREA, THE VR REMAINS ACTIVE.
 │
 │                 • IF THE VR INCLUDES NODE(S) THAT DO  NOT SUPPORT ER'S AND VR'S, THE
 │                   VR IS RESET AND THE VRCB RELEASED WITHOUT SENDING NC_DACTVR.
 │
 │                 • IF THE VR INCLUDES  ONLY NODES THAT DO SUPPORT ER'S  AND VR'S, THE
 │                   ACTION TAKEN DEPENDS  ON THE STATE OF  THE VR AND ON  WHETHER THIS
 │                   END OF THE VR IS ALLOWED TO SEND NC_DACTVR(ORDERLY).  IF THE STATE
 │                   OF THE VR IS ACTIVE AND IF THIS  IS THE NODE THAT SENT NC_ACTVR OR
 │                   THE LATEST NEGATIVE RESPONSE TO NC_DACTVR(ORDERLY), THEN THIS NODE
 │                   SENDS NC_DACTVR(ORDERLY).  IF  THE STATE OF FSM_VR  INDICATES THAT
 │                   SESSION OUTAGE NOTIFICATION IS BEING PERFORMED BECAUSE A NC_DACTVR
 │                   (FORCED) WAS  RECEIVED OR BECAUSE  THE VR BECAME  INOPERATIVE, THE
 │                   SESS_COUNT=0 SIGNAL INDICATES THAT SESSION OUTAGE NOTIFICATION HAS
 │                   BEEN  COMPLETED;  THEREFORE, THE  VR  IS  RESET  AND THE  VRCB  IS
 │                   RELEASED.
 │
 │      INPUT:     SESS_COUNT=0 SIGNAL AND VRCB_PTR
 │
 │      OUTPUT:    NC_DACTVR(ORDERLY) TO PC.ERC (CHAPTER 3)
 │
 │      NOTE:      THE FSM INPUT ROW IS ONE OF THE FOLLOWING:
 │
 │                          ('SESS_COUNT=0', CAN_SEND_O, ¬PRE_VR, ¬LOCAL)
 │                          ('SESS_COUNT=0', ¬PRE_VR, ¬LOCAL)
 │                          ('SESS_COUNT=0', PRE_VR, ¬LOCAL)
 │                          ('SESS_COUNT=0', LOCAL)
 │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                 ACTVR_RCV                       PAGE 12-96
 │                 VR_MGR                          PAGE 12-79
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 BUILD_NC_TH_RH                  PAGE 12-123
 │                 FSM_DACTVR_DIRECTION            PAGE 12-122
 │                 FSM_VR                          PAGE 12-121
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  CALL FSM_VR('SESS_COUNT=0');                          /* NOTE, PAGE 12-121      */

  RETURN;
END SEND_DACTVR_ORDERLY;
```

```
SEND_DACTVR_FORCED: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO PROCESS A SEND_DACTVR_F SIGNAL  INDICATING THAT THE VIRTUAL ROUTE
  │                 SHOULD POSSIBLY  BE RESET.   THE HIGHER-LEVEL  SCHEDULER SENDS  THIS
  │                 SIGNAL FOR EACH VR IDENTIFIED IN THE VRCB LIST.
  │
  │                 • IF THE VR INCLUDES NODES THAT  DO NOT SUPPORT ER-VR PROTOCOLS, THE
  │                   VR IS RESET WITHOUT SENDING NC_DACTVR.
  │
  │                 • IF THE VR INCLUDES ONLY NODES THAT DO SUPPORT ER-VR PROTOCOLS, AND
  │                   THE STATE OF THE VR IS ACTIVE, OR NC_DACTVR(ORDERLY) HAS BEEN SENT
  │                   BUT THE RESPONSE NOT YET RECEIVED, THEN NC_DACTVR(FORCED) IS SENT.
  │                   IF THE PREVIOUSLY ACTIVATED VR IS IN ANY OTHER STATE, IT IS IN THE
  │                   PROCESS OF BEING RESET, AND THERE IS NO NEED TO SEND NC_DACTVR.
  │
  │      INPUT:     SEND_DACTVR_F SIGNAL AND VRCB_PTR FROM THE HIGHER-LEVEL SCHEDULER
  │
  │      OUTPUT:    NC_DACTVR(FORCED) TO PC.ERC (CHAPTER 3) IN FSM_VR
  │
  │      NOTE:      THE FSM INPUT ROW IS ONE OF THE FOLLOWING:
  │                           ('SEND_DACTVR_F', ¬PRE_VR)
  │                           ('SEND_DACTVR_F', PRE_VR)
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 VR_MGR                              PAGE 12-79
  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):
  │                 BUILD_NC_TH_RH                      PAGE 12-123
  │                 FSM_VR                              PAGE 12-121
  │                 UPM_SEND_DACTVR_FORCED              PAGE 12-108
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  IF VRCB.PARTNER_SA ¬= NCB.NODE_SUBAREA_ADDRESS &          /* NOT LOCAL VR      */
       UPM_SEND_DACTVR_FORCED = YES  THEN                   /* PAGE 12-108       */
    CALL FSM_VR('SEND_DACTVR_F');                           /* NOTE, PAGE 12-121 */

  RETURN;
END SEND_DACTVR_FORCED;
```

```
DACTVR_RCV: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO HANDLE RECEIPT OF NC_DACTVR REQUEST OR RESPONSE,  │
  │                 AND THE RESULTANT DEACTIVATION OF A VIRTUAL ROUTE.   │
  │                                                                      │
  │      INPUT:     NC_DACTVR REQUEST OR RESPONSE FROM PC.ERC (CHAPTER 3) │
  │                                                                      │
  │      OUTPUT:    +RSP(NC_DACTVR), -RSP(NC_DACTVR), OR NC_ACTVR TO     │
  │                 PC.ERC (CHAPTER 3)                                   │
  │                                                                      │
  │      NOTES:     1. RECEIVING NC_DACTVR WHEN THE VRCB DOES NOT EXIST  │
  │                    IS A SHOULD-NOT-OCCUR CONDITION.                  │
  │                                                                      │
  │                 2. THE FSM INPUT ROW IS ONE OF THE FOLLOWING:        │
  │                                                                      │
  │                          (R, RQ, DACTVR_O, NO_SESS)                  │
  │                          (R, RQ, DACTVR_O,¬NO_SESS)                  │
  │                          (R, RQ, DACTVR_F, NO_SESS)                  │
  │                          (R, RQ, DACTVR_F,¬NO_SESS)                  │
  │                          (R, +RSP, DACTVR, WAIT_SESS)                │
  │                          (R, +RSP, DACTVR,¬WAIT_SESS)                │
  │                          (R, -RSP, DACTVR)                           │
  │                                                                      │
  │                 3. THIS STATEMENT DISCARDS A RECEIVED NC_DACTVR      │
  │                    RESPONSE.                                         │
  │                                                                      │
  │                 4. THIS IS THE CASE WHEN                             │
  │                                                                      │
  │                       A) NC_DACTVR(ORDERLY) WAS SENT,                │
  │                       B) SESSION ACTIVATION REQUESTS HAVE BEEN PUT IN │
  │                          VRCB.VR_RESERVATION_LIST, AND               │
  │                       C) A -RSP(NC_DACTVR) HAS BEEN RECEIVED.        │
  │                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
  │                 VR_MGR                          PAGE 12-79           │
  │                                                                      │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
  │                 BUILD_ACTVR                     PAGE 12-116          │
  │                 FSM_VR                          PAGE 12-121          │
  │                 UPM_ALLOW_SNF_OVERRIDE          PAGE 12-103          │
  │                 VR_ACTIVATED                    PAGE 12-101          │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  FIND VRCB IN VRCB_LIST WHERE(VRCB.PARTNER_SA = OSAF & VRCB.VR_ID = VRID);

  IF VRCB_PTR = NULL THEN
    IF RRI = RQ THEN
      DO;
      . CALL UPM_LOG('DACTVR_WITHOUT_VRCB');              /* NOTE 1, APPENDIX B   */
      . DISCARD MU;
      END;
    ELSE
      DISCARD MU;                                         /* RESPONSE UNIT        */

  ELSE
    DO;
    . CALL FSM_VR;                                        /* NOTE 2, PAGE 12-121  */
    . IF MU_PTR ¬= NULL THEN                              /* IF RESPONSE NOT SENT */
    .   DISCARD MU;                                       /* NOTE 3               */
    END;

  RETURN;
END DACTVR_RCV;



UPM_SEND_DACTVR_FORCED: PROCEDURE RETURNS(BIT(1));

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE IS AN INSTALLATION-DEFINED UPM THAT   │
  │                 DETERMINES IF THE VR ASSOCIATED WITH THE CURRENT     │
  │                 VRCB SHOULD BE DEACTIVATED.                          │
  │                                                                      │
  │      INPUT:     VRCB                                                 │
  │                                                                      │
  │      OUTPUT:    YES, INDICATING THAT THE VR SHOULD BE DEACTIVATED OR, │
  │                 NO, INDICATING THAT THE VR SHOULD NOT BE DEACTIVATED │
  │                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
  │                 SEND_DACTVR_FORCED              PAGE 12-107          │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  RETURN(YES);

END UPM_SEND_DACTVR_FORCED;
```

VIRTUAL ROUTE INOPERATIVE (VR_INOP)

Flow: VR manager to CP (Normal)

Principal FSM: FSM_VR (Page 12-121)

VR_INOP may be sent for each CP-PU session in which SDT has flowed, to notify the CPs that one or more VRs have become inoperative. This information may be passed to a network operator. When a TG becomes inoperative, all ERs using that TG become inoperative. To identify these ERs, NC_ER_INOP requests are sent using fan-out propagation. When an ER becomes inoperative, all VRs supported by that ER become inoperative. The ER manager in each node at the end of a newly inoperative ER constructs an PARM_ER_INOP parameter list and sends it to the VR manager in the same node. For more detail see the discussion of NC_ER_INOP in the "ER manager" section of this chapter.

Optionally, a VR_INOP request is sent to appropriate CPs as the result of an ERINOP signal from the ER manager to the VR manager. For each VR assigned to a newly inoperative ER, the VR manager changes the state of FSM_VR to reflect the inoperative condition of the VR, notifies PU.SVC_MGR.CSC_MGR to initiate session outage notification for any sessions assigned to the VR, and prepares an entry for VR_INOP.

The VR_INOP request carries a code specifying the type of routing interruption--unexpected or controlled--along with the addresses of the subareas on each end of the failing TG, TGN of the failing TG, and a list of entries identifying the VRs ending in this node that have become inoperative because of a routing interruption on the TG. Each VR is identified by the subarea address at the other end of the VR, the VRID, and the ERN assigned to this VR for transmitting from this node.

```
VR_INOP_SEND: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO  PROCESS A  ERINOP SIGNAL  FROM THE  ER MANAGER.   THE SIGNAL  IS │
  │                 ACCOMPANIED BY  A PARM_ER_INOP PARAMETER LIST  CONTAINING (DSA, ERN) │
  │                 PAIRS,  EACH OF  WHICH SPECIFIES  AN  INOPERATIVE ER.   FOR EACH  VR │
  │                 SUPPORTED  BY THE  INOPERATIVE  ER, THIS  PROCEDURE  SENDS A  VRINOP │
  │                 SIGNAL TO PU.SVC_MGR.CSC_MGR (CHAPTER 13) TO INITIATE SESSION OUTAGE │
  │                 NOTIFICATION, CHANGES FSM_VR STATE TO REFLECT THE INOPERATIVE NATURE │
  │                 OF THE VR, AND  PREPARES AN ENTRY TO BE SENT  IN VR_INOP IDENTIFYING │
  │                 THE VR.  AFTER PROCESSING ALL (DSA, ERN) PAIRS, THIS PROCEDURE SENDS │
  │                 THE NEWLY BUILT VR_INOP REQUEST TO THE APPROPRIATE CP'S.             │
  │                                                                                      │
  │      INPUT:     ERINOP SIGNAL AND PARM_ER_INOP STRUCTURE FROM THE ER MANAGER          │
  │                                                                                      │
  │      OUTPUT:    VRINOP SIGNAL  TO PU.SVC_MGR.CSC_MGR  (CHAPTER 13),  VR_INOP REQUEST │
  │                 FOR EACH CP-PU SESSION IN WHICH SDT HAS FLOWED.                      │
  │                                                                                      │
  │      NOTES:     1. PU.SVC_MGR.CSC_MGR USES THE VRCB_PTR ONLY AS AN IDENTIFIER OF THE │
  │                    VR THAT  BECAME INOPERATIVE,  PU.SVC_MGR.CSC_MGR DOES  NOT ACCESS │
  │                    THE VRCB ITSELF.                                                  │
  │                                                                                      │
  │                 2. THIS SCAN  SENDS AN  VR_INOP REQUEST  FOR EACH  CP-PU SESSION  IN │
  │                    WHICH SDT HAS FLOWED.   IF VR_INOP IS RECEIVED BY A  CP THAT DOES │
  │                    NOT SUPPORT ER-VR PROTOCOLS, THAT CP DISCARDS IT.                 │
  │                                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                                       │
  │                 VR_MGR                              PAGE 12-79                       │
  │                                                                                      │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                           │
  │                 BUILD_NS_RQN_RH                     PAGE 12-124                      │
  │                 FSM_VR                              PAGE 12-121                      │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL VR_INOP_PTR PTR;
  DCL ER_INOP_CNT BIT(8);
  DCL ER_NUM BIT(4);

  PARM_ER_INOP_PTR = PARM_PTR;                                  /* PAGE 12-127    */

  CREATE MU;
  CALL BUILD_NS_RQN_RH(MU_PTR);                                 /* PAGE 12-124    */
  VR_INOP_RQ = PARM_ER_INOP, BY NAME;                           /* PAGE 12-127    */
  VR_INOP_RQ.NS_HEADER = VR_INOP_HDR;
  VR_INOP_RQ.FORMAT = FORMAT1;
  VR_INOP_RQ.CNT_VR_FIELD = 0;

  DO ER_INOP_CNT = 1 TO PARM_ER_INOP.CNT_ER_FIELD;              /* PAGE 12-127    */
  . SCAN VRCB_LIST PTR(VRCB_PTR);
  . . IF VRCB.PARTNER_SA = PARM_ER_INOP.SA(ER_INOP_CNT) &       /* PAGE 12-127    */
  . .       PARM_ER_INOP.MASK(ER_INOP_CNT,VRCB.ER_NUM:VRCB.ER_NUM) = ON THEN /* PAGE 12-127 */
  . .    DO;
  . .    . SEND 'VRINOP' TO PU.SVC_MGR.CSC_MGR.SON;           /* NOTE 1, CHAPTER 13 */
  . .    . VR_INOP_RQ.CNT_VR_FIELD = VR_INOP_RQ.CNT_VR_FIELD + 1;
  . .    . VR_INOP_RQ.SA(VR_INOP_RQ.CNT_VR_FIELD) = VRCB.PARTNER_SA;
  . .    . VR_INOP_RQ.VR_ID(VR_INOP_RQ.CNT_VR_FIELD) = VRCB.VR_ID;
  . .    . VR_INOP_RQ.MASK(VR_INOP_RQ.CNT_VR_FIELD) = ALL_OFF;
  . .    . VR_INOP_RQ.MASK(VR_INOP_RQ.CNT_VR_FIELD,VRCB.ER_NUM:VRCB.ER_NUM) = ON;
  . .    . CALL FSM_VR('VRINOP');                              /* PAGE 12-121    */
  . .    END;
  . SCANEND;
  END;

  DISCARD PARM_ER_INOP;                                         /* PAGE 12-127    */

  DCF = RH_LENGTH + 15 + VR_INOP_RQ.CNT_VR_FIELD * 8;
  VR_INOP_PTR = MU_PTR;

  NRCB_PTR = LOCATE_NODE_RESOURCE(NCB.PU_EA);                   /* APPENDIX B     */
  SCAN NRCB.CP_INDIRECT_LIST PTR(CP_INDIRECT_PTR);              /* NOTE 2, APPENDIX A */
  . CPCB_PTR = CP_INDIRECT.CP_ENTRY_PTR;                        /* APPENDIX A     */
  . IF FSM_CP_SESS_SDT = ACTIVE THEN                            /* CHAPTER 11     */
  .    DO;
  .    . CREATE MU;
  .    . MU = VR_INOP_PTR->MU;          /* COPY VR_INOP REQUEST INTO CURRENT MU */
  .    . SCB_PTR = CPCB.CP_SCB_ID;                              /* APPENDIX A     */
  .    . SEND MU TO SNS.SEND;                                   /* OPTIONAL, CHAPTER 6 */
  .    END;
  SCANEND;

  DISCARD VR_INOP_PTR->MU;

  RETURN;
END VR_INOP_SEND;
```

VIRTUAL ROUTE TESTING

Testing of virtual routes is initiated by the ROUTE_TEST request, which is sent from an SSCP to a PU. It specifies another subarea in the network and a list of VRNs or ERNs. If the list is of ERNs, all explicit routes using those ERNs for PIUs flowing from the subarea of the PU receiving the ROUTE_TEST request to the subarea specified in the request are to be tested; the virtual routes supported by those explicit routes are also to be tested. If the list is of VRNs, all virtual routes assigned those VRNs and connecting the subarea of the PU receiving the ROUTE_TEST and the subarea specified in the request are to be tested, along with the explicit routes underlying those virtual routes.

Testing involves two different activities. The first is reporting the states, as known in the node receiving the ROUTE_TEST request, of the ERs and VRs. These states are determined by the VR manager and reported to the SSCP by the ROUTE_TEST response. The second activity is determining the condition of ERs by sending NC_ER_TEST. This investigation of the ERs is done by the ER manager and is reported by ER_TESTED requests sent to appropriate CPs. (See the discussion of NC_ER_TEST and ER_TESTED in the "ER manager" section of this chapter.)

ROUTE TEST (ROUTE_TEST)

Flow: SSCP to PU_T4|5 (Normal)

Principal FSMs: FSM_VR (Page 12-121)

The VR manager checks FSM_VR and FSM_ER for the states of routes specified in the ROUTE_MASK field of ROUTE_TEST, and reports these states (e.g., active, not defined, pending deactivation) in the response to ROUTE_TEST. The VR manager also checks the Test Code byte of ROUTE_TEST; if this byte specifies that NC_ER_TEST requests are to be sent on the ERs, the VR manager sends a copy of the ROUTE_TEST request to the ER manager to generate and send the NC_ER_TEST requests.

```
ROUTE_TEST_RCV: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO BUILD AND SEND RSP(ROUTE_TEST).    IF ROUTE_TEST CALLS FOR │
  │                 EXPLICITLY TESTING THE UNDERLYING ER'S, THE ER MANAGER IS CALLED
  │                 (WITH AN INPUT OF ROUTE_TEST) TO PERFORM THAT FUNCTION.      │
  │                                                                              │
  │      INPUT:     ROUTE_TEST AND SCB_PTR                                       │
  │                                                                              │
  │      OUTPUT:    RSP(ROUTE_TEST) TO SNS.SEND, AND DISCARDED ROUTE_TEST        │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 VR_MGR                              PAGE 12-79               │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 BUILD_NS_RQN_RH                     PAGE 12-124              │
  │                 CHANGE_VRN_MU_TO_POS_RSP            PAGE 12-118              │
  │                 SET_ER                              PAGE 12-65               │
  │                 SET_VR                              PAGE 12-114              │
  │                 TEST_SEND                           PAGE 12-56               │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL(TESTRQ_PTR,TESTRSP_PTR) PTR;
  DCL (VR_NUM,ER_NUM) BIT(4);

  TESTRQ_PTR = MU_PTR;                      /* ADDRESSABILITY TO ROUTE_TEST RQ  */

  CREATE MU;
  MU = TESTRQ_PTR->MU;                                      /* COPY RQ INTO RSP */
  CALL CHANGE_VRN_MU_TO_POS_RSP(NO_TRUNCATE);               /* PAGE 12-118      */

  ROUTE_TEST_RSP.NS_HEADER = ROUTE_TEST_HDR;
  ROUTE_TEST_RSP.FORMAT = FORMAT1;
  ROUTE_TEST_RSP.CNT_ROUTE_DATA = 0;

  SELECT ANYORDER(TESTRQ_PTR->ROUTE_TEST_RQ.TEST_TYPE);
  .
  . WHEN(TEST_VRS)
  .   DO VR_NUM = 0 TO NCB.MAX_VR_NUM;                     /* APPENDIX A       */
  .   . IF TESTRQ_PTR->ROUTE_TEST_RQ.ROUTE_MASK(VR_NUM:VR_NUM) = ON THEN
  .   .   CALL SET_VR(TESTRQ_PTR->ROUTE_TEST_RQ.DESTINATION_SA,VR_NUM);
  .   .                                                   /* PAGE 12-114      */
  .   END;
  .
  . WHEN(TEST_ERS | TEST_DEFINED_ERS)
  .   DO ER_NUM = 0 TO NCB.MAX_ER_NUM;                     /* APPENDIX A       */
  .   . IF TESTRQ_PTR->ROUTE_TEST_RQ.ROUTE_MASK(ER_NUM:ER_NUM) = ON THEN
  .   .   CALL SET_ER(TESTRQ_PTR->ROUTE_TEST_RQ.DESTINATION_SA,ER_NUM);
  .   .                                                   /* PAGE 12-65       */
  .   END;
  END;

  DCF = RH_LENGTH + 5 + ROUTE_TEST_RSP.CNT_ROUTE_DATA * 8;
  SEND MU TO SNS.SEND;                                      /* CHAPTER 6        */


                                                                                /*
                  ┌──────────────────────────────────────┐
                  │  DECIDE WHETHER TO TEST THE ER'S.    │
                  └──────────────────────────────────────┘
                                                                                */
  MU_PTR = TESTRQ_PTR;                      /* ADDRESS ROUTE_TEST_RQ            */
  IF ROUTE_TEST_RQ.TEST_CODE ¬= DO_NOT_TEST_ER THEN
    CALL TEST_SEND;                                         /* PAGE 12-56       */

  DISCARD MU;                                               /* ROUTE_TEST_RQ    */

  RETURN;
END ROUTE_TEST_RCV;
```

```
SET_VR: PROCEDURE(DEST_SA,VR_NUM);
                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO  FILL  IN THE  APPROPRIATE  FIELD  OF  THE ROUTE_DATA  FIELDS  IN │
   │                 RSP(ROUTE_TEST) FOR ALL VR'S HAVING A GIVEN VRN.  THE ER INFORMATION │
   │                 IN THE ROUTE_DATA FIELD REFERS TO THE  ER THAT IS DEFINED TO SUPPORT │
   │                 THE VR.                                                  │
   │                                                                          │
   │      INPUT:     RSP(ROUTE_TEST); DEST_SA  IS THE DESTINATION  SUBAREA OF THE  VR AND │
   │                 VR_NUM IS THE VR NUMBER OF THE VRS WHOSE STATUS IS TO BE DETERMINED │
   │                                                                          │
   │      OUTPUT:    COMPLETED ROUTE_DATA FIELDS OF RSP(ROUTE_TEST)           │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 ROUTE_TEST_RCV                  PAGE 12-113              │
   │                                                                          │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
   │                 FIND_ER_STATUS                  PAGE 12-66               │
   │                 UPM_SET_VR_STATUS               PAGE 12-114              │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL DEST_SA BIT(32);
  DCL VR_NUM BIT(4);
  DCL ER_NUM BIT(4);
  DCL ER_STATUS BIT(8);
  DCL ADJ_SA BIT(32);


  CALL FIND_ER_STATUS(DEST_SA,VR_NUM,ER_NUM,ER_STATUS,ADJ_SA);   /* PAGE 12-66          */

  SCAN VRCB_LIST PTR(VRCB_PTR);
  . IF VRCB.VR_NUM = VR_NUM THEN
  .   DO;
  .   . ROUTE_TEST_RSP.CNT_ROUTE_DATA = ROUTE_TEST_RSP.CNT_ROUTE_DATA + 1;
  .   . ROUTE_TEST_RSP.VR_ID(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = VRCB.VR_ID;
  .   . CALL UPM_SET_VR_STATUS;                          /* PAGE 12-114          */
  .   . ROUTE_TEST_RSP.ER_NUM(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = ER_NUM;
  .   . ROUTE_TEST_RSP.ER_STATUS(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = ER_STATUS;
  .   . ROUTE_TEST_RSP.ORIGINATING_ADJ_SA(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = ADJ_SA;
  .   END;
  SCANEND;

  RETURN;
END SET_VR;



UPM_SET_VR_STATUS: PROCEDURE;
                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO DETERMINE THE STATUS OF A VR AS RECORDED IN ITS VRCB  │
   │                                                                          │
   │      INPUT:     RSP(ROUTE_TEST) AND VRCB_PTR                             │
   │                                                                          │
   │      OUTPUT:    THE VR_STATUS FIELD  IN THE ROUTE_DATA FIELD  OF THE RSP(ROUTE_TEST) │
   │                 IS FILLED IN.                                            │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 FIND_VR_STATUS                  PAGE 12-115              │
   │                 SET_VR                          PAGE 12-114              │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */

  RETURN;
END UPM_SET_VR_STATUS;
```

```
FIND_VR_STATUS: PROCEDURE(DEST_SA,VR_NUM,ER_STATUS);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO FILL IN THE APPROPRIATE FIELDS OF ONE ROUTE_DATA FIELD IN │
   │                 RSP(ROUTE_TEST) FOR EVERY VR THAT USES A GIVEN VRN       │
   │                                                                          │
   │      INPUT:     RSP(ROUTE_TEST), ERCB_PTR, AND PATHCB_PTR; DEST_SA IS THE │
   │                 DESTINATION SUBAREA OF THE VR, VR_NUM IS THE VRN, AND ER_STATUS IS │
   │                 THE STATUS OF THE UNDERLYING ER                          │
   │                                                                          │
   │      OUTPUT:    COMPLETED ROUTE_DATA FIELD OF RSP(ROUTE_TEST)            │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 SET_ER                          PAGE 12-65               │
   │                                                                          │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
   │                 UPM_SET_VR_STATUS               PAGE 12-114              │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL DEST_SA BIT(32);
  DCL VR_NUM BIT(4);
  DCL ER_STATUS BIT(8);

  SCAN VRCB_LIST PTR(VRCB_PTR);
  . IF VRCB.VR_NUM = VR_NUM THEN
  .   DO;
  .   . ROUTE_TEST_RSP.CNT_ROUTE_DATA = ROUTE_TEST_RSP.CNT_ROUTE_DATA + 1;
  .   . ROUTE_TEST_RSP.VR_ID(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = VRCB.VR_ID;
  .   . CALL UPM_SET_VR_STATUS;                          /* PAGE 12-114          */
  .   . ROUTE_TEST_RSP.ER_NUM(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = ERCB.ER_NUM;
  .   . ROUTE_TEST_RSP.ER_STATUS(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = ER_STATUS;
  .   . ROUTE_TEST_RSP.ORIGINATING_ADJ_SA(ROUTE_TEST_RSP.CNT_ROUTE_DATA) = PATHCB.ADJ_SA;
  .   END;
  SCANEND;

  RETURN;
END FIND_VR_STATUS;
```

```
BUILD_ACTVR: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO BUILD AN NC_ACTVR REQUEST                                 │
  │                                                                              │
  │      INPUT:     VRCB_PTR AND ERCB_PTR                                        │
  │                                                                              │
  │      OUTPUT:    NC_ACTVR REQUEST                                             │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 DACTVR_RCV                      PAGE 12-108                  │
  │                 ER_ACTIVATION_TERMINATOR        PAGE 12-92                   │
  │                 FSM_VR                          PAGE 12-121                  │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 BUILD_NC_TH_RH                  PAGE 12-123                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  CREATE MU;

  CALL BUILD_NC_TH_RH(MU_PTR);                          /* PAGE 12-123        */
  VRN = VRCB.VR_NUM;
  IERN = VRN;
  ERN = VRCB.ER_NUM;
  TPF = VRCB.TP_FIELD;
  DSAF = VRCB.PARTNER_SA;
  DR1I = ON;                                            /* DEFINITE RESPONSE  */
  DR2I = OFF;
  DCF = RH_LENGTH + 19;

  RQ_CODE = NC_ACTVR;
  NC_ACTVR_RQ.FORMAT = FORMAT1;

  NC_ACTVR_RQ.RCV_ERN_MASK = ERCB.RERN_MASK;
  NC_ACTVR_RQ.SEND_ERN_MASK(VRCB.ER_NUM:VRCB.ER_NUM) = ON;

  NC_ACTVR_RQ.VR_SEND_SEQ_NO = VRCB.SNF_RCV_CNTR;

  NC_ACTVR_RQ.MAX_WINDOW_SIZE = VRCB.MAX_WINDOW_SIZE;
  NC_ACTVR_RQ.MIN_WINDOW_SIZE = VRCB.MIN_WINDOW_SIZE;

  NC_ACTVR_RQ.MAX_SEND_PIU_LENGTH = NO_RESTRICTION;
  NC_ACTVR_RQ.MAX_RCV_PIU_LENGTH = NO_RESTRICTION;

  RETURN;
END BUILD_ACTVR;



BUILD_DACTVR: PROCEDURE(TYPE);

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO BUILD AN NC_DACTVR REQUEST                                │
  │                                                                              │
  │      INPUT:     VRCB_PTR, AND PARAMETER TYPE INDICATING  WHETHER THE NC_DACTVR IS OF
  │                 TYPE ORDERLY OR FORCED                                       │
  │                                                                              │
  │      OUTPUT:    NC_DACTVR                                                    │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 FSM_VR                          PAGE 12-121                  │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 BUILD_NC_TH_RH                  PAGE 12-123                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  DCL TYPE BIT(8);

  CREATE MU;

  CALL BUILD_NC_TH_RH(MU_PTR);                          /* PAGE 12-123        */
  VRN = VRCB.VR_NUM;
  IERN = VRN;
  ERN = VRCB.ER_NUM;
  TPF = VRCB.TP_FIELD;
  DSAF = VRCB.PARTNER_SA;
  DR1I = ON;                                            /* DEFINITE RESPONSE  */
  DR2I = OFF;
  DCF = RH_LENGTH + 5;

  RQ_CODE = NC_DACTVR;
  NC_DACTVR_RQ.FORMAT = FORMAT1;
  NC_DACTVR_RQ.TYPE = TYPE;

  RETURN;
END BUILD_DACTVR;
```

```
RELEASE_VRCB: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  THIS  PROCEDURE REMOVES  A  VRCB FROM  VRCB_LIST  AND PROCESSES  ALL  │
   │                SESSION ACTIVATION REQUESTS THAT ARE  AWAITING THE ACTIVATION OF THE  │
   │                VR, BY INVOKING THE VR_ID_LIST_PROCESSOR.                            │
   │                                                                                     │
   │     INPUT:     VRCB_PTR                                                             │
   │                                                                                     │
   │     OUTPUT:    NULL  VRCB_PTR AND  -RSP(ACTCDRM|ACTLU|ACTPU|BIND)  FOR ANY  SESSION │
   │                ACTIVATION  REQUEST  WHOSE  VR_ID_LIST HAS  BEEN  EXHAUSTED  WITHOUT │
   │                PRODUCING AN ACTIVE VR                                               │
   │                                                                                     │
   │     NOTES:     1. AN EMPTY VRCB.VR_RESERVATION_LIST IS HANDLED PROPERLY.            │
   │                                                                                     │
   │                2. THE  SESSION ACTIVATION  REQUEST (ACTCDRM|ACTLU|ACTPU|BIND)  FROM │
   │                   THE VR_RESERVATION IS MADE THE CURRENT MESSAGE UNIT.              │
   │                                                                                     │
   │                3. VR_ID_LIST_PROCESSOR WILL CHANGE VRCB_PTR.                        │
   │                                                                                     │
   │                4. WHEN   THIS    PROCEDURE   IS   INVOKED,    THE   VRCB    HAS   A │
   │                   VR_RESERVATION_LIST HEADER.                                       │
   │                                                                                     │
   │     REFERENCED BY THE FOLLOWING PROCEDURE(S):                                       │
   │                PSM_VR                              PAGE 12-121                      │
   │                                                                                     │
   │     REFERS TO THE FOLLOWING PROCEDURE(S):                                           │
   │                VR_ID_LIST_PROCESSOR                PAGE 12-88                       │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  DCL NEXT_VR_ID_LIST_INDEX BIT(8);
  DCL SAVED_VRCB_PTR POINTER;
  DCL SAVED_MU_PTR POINTER;

  SAVED_MU_PTR = MU_PTR;

  SCAN VRCB.VR_RESERVATION_LIST PTR(VR_RESERVATION_PTR);        /* NOTE 1        */
  .
  . MU_PTR = VR_RESERVATION.SESSION_ACT_RQ;                     /* NOTE 2        */
  . SCB_PTR = VR_RESERVATION.SCBPTR;
  . VR_ID_LIST_PTR = VR_RESERVATION.VR_LIST;
  . NEXT_VR_ID_LIST_INDEX = VR_RESERVATION.VR_LIST_INDEX + 1;
  .
  . REMOVE VR_RESERVATION FROM VR_RESERVATION_LIST DISCARD;
  .
  . SAVED_VRCB_PTR = VRCB_PTR;                                  /* NOTE 3        */
  . CALL VR_ID_LIST_PROCESSOR(NEXT_VR_ID_LIST_INDEX);           /* PAGE 12-88    */
  . VRCB_PTR = SAVED_VRCB_PTR;
  SCANEND;

  MU_PTR = SAVED_MU_PTR;

  DESTROY VRCB.VR_RESERVATION_LIST;                             /* NOTE 4        */
  DESTROY VRCB.UPM_SEGMENTS_LIST;
  DESTROY VRCB.Q_VR_PAC;

  REMOVE VRCB FROM VRCB_LIST DISCARD;

  RETURN;
END RELEASE_VRCB;
```

```
CHANGE_VRM_MU_TO_POS_RSP: PROCEDURE(TRUNCATION);

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO CHANGE A REQUEST PROCESSED BY THE VR MANAGER TO A │
  │                 RESPONSE AND SET UP THE TH PROPERLY                  │
  │                                                                      │
  │      INPUT:     REQUEST MU AND TRUNCATION PARAMETER INDICATING WHETHER│
  │                 TRUNCATE THE RESPONSE UNIT                           │
  │                                                                      │
  │      OUTPUT:    POSITIVE RESPONSE MU AND VRCB_PTR THE INPUT REQUEST  │
  │                 OVERWRITTEN BY THE RESPONSE UNIT.                    │
  │                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
  │                 ACTVR_RQ_RCV                 PAGE 12-100             │
  │                 CANCEL_VR_RESERVATION        PAGE 12-102             │
  │                 FSM_VR                       PAGE 12-121             │
  │                 ROUTE_TEST_RCV               PAGE 12-113             │
  │                                                                      │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
  │                 SWAP_ORIGIN_DEST             PAGE 12-119             │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL TRUNCATION BIT(1);

  FIND VRCB IN VRCB_LIST
       WHERE(VRCB.PARTNER_SA = OSAF  &  VRCB.VR_ID = VRID);

  CALL CHANGE_MU_TO_POS_RSP(TRUNCATION);                  /* APPENDIX B   */

  CALL SWAP_ORIGIN_DEST;                                  /* PAGE 12-119  */

  ERN = VRCB.ER_NUM;

  RETURN;
END CHANGE_VRM_MU_TO_POS_RSP;



CHANGE_VRM_MU_TO_NEG_RSP: PROCEDURE(SENSE_CODE);

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO CHANGE A REQUEST PROCESSED BY THE VR MANAGER TO A │
  │                 RESPONSE AND SET UP THE TH PROPERLY                  │
  │                                                                      │
  │      INPUT:     REQUEST MU, AND SENSE_CODE PARAMETER GIVING THE SENSE│
  │                 INTO THE NEGATIVE RESPONSE                           │
  │                                                                      │
  │      OUTPUT:    NEGATIVE RESPONSE MU THE INPUT REQUEST UNIT IS OVERWRI│
  │                 RESPONSE UNIT.                                       │
  │                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
  │                 CANCEL_VR_RESERVATION        PAGE 12-102             │
  │                 FSM_VR                       PAGE 12-121             │
  │                 VR_ID_LIST_PROCESSOR         PAGE 12-88              │
  │                                                                      │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
  │                 SWAP_ORIGIN_DEST             PAGE 12-119             │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL SENSE_CODE BIT(32);

  FIND VRCB IN VRCB_LIST
       WHERE(VRCB.PARTNER_SA = OSAF  &  VRCB.VR_ID = VRID);

  CALL CHANGE_MU_TO_NEG_RSP(SENSE_CODE);                  /* APPENDIX B   */

  CALL SWAP_ORIGIN_DEST;                                  /* PAGE 12-119  */

  ERN = VRCB.ER_NUM;

  RETURN;
END CHANGE_VRM_MU_TO_NEG_RSP;
```

```
SWAP_ORIGIN_DEST: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO EXCHANGE THE ORIGIN (OSAF AND OEF) AND DESTINATION (DSAF AND DEF)
  │                 ADDRESS FIELDS OF THE TH
  │
  │      INPUT:     REQUEST UNIT PROCESSED BY VR MANAGER
  │
  │      OUTPUT:    SWAPPED ORIGIN AND DESTINATION ADDRESS FIELDS OF THE TH
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 CHANGE_ACTVR_TO_NEG_RSP          PAGE 12-99
  │                 CHANGE_VRH_RU_TO_NEG_RSP         PAGE 12-118
  │                 CHANGE_VRH_RU_TO_POS_RSP         PAGE 12-118
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL SA BIT(32);
  DCL EA BIT(16);

  SA = OSAF;
  OSAF = DSAF;
  DSAF = SA;

  EA = OEF;
  OEF = DEF;
  DEF = EA;

  RETURN;
END SWAP_ORIGIN_DEST;
```

```
FSM_VR: FSM_DEFINITION CONTEXT(VRCB);

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │     FUNCTION:  TO HOLD THE ACTIVATION AND DEACTIVATION STATUS OF THE VIRTUAL ROUTE.
 │
 │                RESET STATE IS ENTERED IMMEDIATELY UPON THE CREATION OF THE VRCB AND
 │                IMMEDIATELY PRIOR TO THE DESTRUCTION OF THE VRCB.
 │
 │                PEND_ER AND PEND_ACT STATES ARE ENTERED DURING THE ACTIVATION OF THE
 │                VR.  PEND_ER IS  ENTERED WHEN WAITING FOR THE ER  MANAGER TO SPECIFY
 │                WHETHER THE  ER SUPPORTING THIS VR  IS ACTIVE.  PEND_ACT  IS ENTERED
 │                WHEN NC_ACTVR IS SENT.
 │
 │                ACTIVE STATE IS THE ONLY STATE IN  WHICH SEQUENCED PIU'S CAN BE SENT
 │                ON THE VR.
 │
 │                PEND_RESET_O_SEND IS ENTERED WHEN  NC_DACTVR(ORDERLY) IS SENT.  ONCE
 │                THIS STATE IS ENTERED, THE VR CAN AGAIN BECOME ACTIVE OR RESET.
 │
 │                PEND_RESET_F_SEND  IS   ENTERED  WHEN  NC_DACTVR(FORCED)  IS  SENT.
 │                PEND_RESET_F_RCV IS  ENTERED WHEN NC_DACTVR(FORCED) IS  RECEIVED AND
 │                THERE ARE SESSIONS  ASSIGNED TO THE VR.   PEND_RESET_INOP IS ENTERED
 │                WHEN THE VR BECOMES INOPERATIVE.  ONCE  ANY OF THESE THREE STATES IS
 │                ENTERED, THE VR WILL BE RESET.
 │
 │                THE INPUT SIGNAL  'PRE_VR_ACT' IS SENT BY PC.VRC WHEN  IT RECEIVES A
 │                SESSION ACTIVATION  REQUEST ON A ROUTE  THAT CONTAINS NODES  THAT DO
 │                NOT SUPPORT ER-VR  PROTOCOLS, AND THEREFORE WAS  NOT ACTIVATED USING
 │                THOSE PROTOCOLS.
 │
 │     REFERENCED BY THE FOLLOWING PROCEDURE(S):
 │                ACTVR_RCV                          PAGE 12-96
 │                DACTVR_RCV                         PAGE 12-108
 │                ER_ACTIVATION_TERMINATOR           PAGE 12-92
 │                SEND_DACTVR_FORCED                 PAGE 12-107
 │                SEND_DACTVR_ORDERLY                PAGE 12-106
 │                VR_ID_LIST_PROCESSOR               PAGE 12-88
 │                VR_INOP_SEND                       PAGE 12-110
 │
 │     REFERS TO THE FOLLOWING PROCEDURE(S):
 │                ACTVR_RQ_RCV                       PAGE 12-100
 │                BUILD_ACTVR                        PAGE 12-116
 │                BUILD_DACTVR                       PAGE 12-116
 │                CHANGE_ACTVR_TO_NEG_RSP            PAGE 12-99
 │                CHANGE_VRM_MU_TO_NEG_RSP           PAGE 12-118
 │                CHANGE_VRM_MU_TO_POS_RSP           PAGE 12-118
 │                FSM_DACTVR_DIRECTION               PAGE 12-122
 │                RELEASE_VRCB                       PAGE 12-117
 │                UPM_ALLOW_SNF_OVERRIDE             PAGE 12-103
 │                VR_ACTIVATED                       PAGE 12-101
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAME -----> / STATE NUMBER ---> / INPUT	RESET 1	PEND ER 2	PEND ACT SEND 3	ACTIVE 4	PEND RESET O_SEND 5	PEND RESET F_SEND 6	PEND RESET F_RCV 7	PEND RESET INOP 8
'ACTIVATE_ER'	2	/	/	/	/	/	/	/
'ER_ACTIVATED', ¬PRE_VR	>	3(A)	>	>	>	>	>	>
'ER_ACTIVATED', PRE_VR	>	4(K)	>	>	>	>	>	>
'ER_NOT_ACTIVATED'	>	1(H)	>	>	>	>	>	>
R, RQ, ACTVR, LOWER_SA	4(P)	4(P)	-(B)	-(C)	-(C)	-(C)	/	-(C)
R, RQ, ACTVR, ¬LOWER_SA	4(P)	4(P)	4(P)	-(C)	-(C)	-(C)	/	-(C)
R, -RSP, ACTVR, 0815	/	/	1(L)	/	/	/	/	-
R, -RSP, ACTVR, ¬0815	/	-	1(H)	-	/	/	/	-
R, +RSP, ACTVR	-	-	4	/	/	/	/	-
R, RQ, DACTVR_O, NO_SESS	/	/	/	1(E)	/	-	/	/
R, RQ, DACTVR_O, ¬NO_SESS	/	/	/	-(G)	/	-	/	/
R, RQ, DACTVR_F, NO_SESS	/	/	/	1(E)	1(E)	-(E)	/	/
R, RQ, DACTVR_F, ¬NO_SESS	/	/	/	7(F)	/	/	/	/
R, +RSP, DACTVR, WAIT_SESS	/	/	/	/	3(N)	-	-	-
R, +RSP, DACTVR, ¬WAIT_SESS	/	/	/	/	1(H)	1(H)	-	-
R, -RSP, DACTVR	/	/	/	/	4(M)	-	-	-
'SEND_DACTVR_F', ¬PRE_VR	/	/	/	6(J)	6(J)	-	-	-
'SEND_DACTVR_F', PRE_VR	/	/	/	1(H)	/	/	-	-
'SESS_COUNT=0', CAN_SEND_O, ¬PRE_VR, ¬LOCAL	>	>	>	5(D)	-	-	1(R)	1(H)
'SESS_COUNT=0', ¬PRE_VR, ¬LOCAL	>	>	>	-	-	-	1(R)	1(H)
'SESS_COUNT=0', ¬LOCAL	>	>	>	1(H)	/	/	/	1(H)
'SESS_COUNT=0', LOCAL	/	/	/	-	/	/	/	/
'VRINOP'	-	8	8	8	8	8	-	-
'PRE_VR_ACT'	4	/	/	/	/	/	/	/

```
| OUTPUT | FUNCTION                                                                                    |
| CODE   |                                                                                             |
|--------|---------------------------------------------------------------------------------------------|
| A      | SEND MU TO PC.ERC;                                                    /* CHAPTER 3    */|
|--------|---------------------------------------------------------------------------------------------|
| B      | CALL CHANGE_ACTVR_TO_NEG_RSP(X'080D');      /* RACE CONDITION, PAGE 12-99          */|
|        | SEND MU TO PC.ERC;                                                    /* CHAPTER 3    */|
|--------|---------------------------------------------------------------------------------------------|
| C      | CALL CHANGE_ACTVR_TO_NEG_RSP(X'0815');      /* VR ALREADY ACTIVE, PAGE 12-99       */|
|        | SEND MU TO PC.ERC;                                                    /* CHAPTER 3    */|
|        | CALL UPM_LOG('VR ALREADY ACTIVE');                                    /* APPENDIX B   */|
|--------|---------------------------------------------------------------------------------------------|
| D      | CALL BUILD_DACTVR(ORDERLY);                                           /* PAGE 12-116  */|
|        | SEND MU TO PC.ERC;                                                    /* CHAPTER 3    */|
|--------|---------------------------------------------------------------------------------------------|
| E      | CALL CHANGE_VRM_MU_TO_POS_RSP(TRUNCATE);                              /* PAGE 12-118  */|
|        | SEND MU TO PC.ERC;                                                    /* CHAPTER 3    */|
|        | CALL RELEASE_VRCB;                                                    /* PAGE 12-117  */|
|--------|---------------------------------------------------------------------------------------------|
| F      | SEND 'DACTVR_FORCED' TO PU.SVC_MGR.CSC_MGR.SON;                       /* CHAPTER 13   */|
|--------|---------------------------------------------------------------------------------------------|
| G      | CALL CHANGE_VRM_MU_TO_NEG_RSP(X'0872');     /* SESSIONS STILL ON VR, PAGE 12-118 */|
|        | CALL FSM_DACTVR_DIRECTION;                                            /* PAGE 12-122  */|
|        | SEND MU TO PC.ERC;                                                    /* CHAPTER 3    */|
|        | CALL VR_ACTIVATED;                                                    /* PAGE 12-101  */|
|--------|---------------------------------------------------------------------------------------------|
| H      | CALL RELEASE_VRCB;                                                    /* PAGE 12-117  */|
|--------|---------------------------------------------------------------------------------------------|
| J      | CALL BUILD_DACTVR(FORCED);                                            /* PAGE 12-116  */|
|        | SEND MU TO PC.ERC;                                                    /* CHAPTER 3    */|
|--------|---------------------------------------------------------------------------------------------|
| K      | CALL VR_ACTIVATED;                                                    /* PAGE 12-101  */|
|--------|---------------------------------------------------------------------------------------------|
| L      | CALL UPM_LOG('VR ALREADY ACTIVE');                                    /* APPENDIX B   */|
|        | DISCARD MU;                                                                         |
|        | CALL RELEASE_VRCB;                                                    /* PAGE 12-117  */|
|--------|---------------------------------------------------------------------------------------------|
| M      | CALL FSM_DACTVR_DIRECTION;                                            /* PAGE 12-122  */|
|        | CALL VR_ACTIVATED;                                                    /* PAGE 12-101  */|
|--------|---------------------------------------------------------------------------------------------|
| N      | DISCARD MU;                                                           /* RSP(DACTVR)  */|
|        | VRCB.SNF_SEND_CNTR = ZERO;                                                          |
|        | VRCB.SNF_RCV_CNTR = ZERO;                                                           |
|        | CALL UPM_ALLOW_SNF_OVERRIDE;                                          /* PAGE 12-103  */|
|        | CALL BUILD_ACTVR;                                                     /* PAGE 12-117  */|
|        | SEND MU TO PC.ERC;                                                    /* CHAPTER 3    */|
|--------|---------------------------------------------------------------------------------------------|
| P      | CALL ACTVR_RQ_RCV;                                                    /* PAGE 12-100  */|
|--------|---------------------------------------------------------------------------------------------|
| R      | CALL BUILD_DACTVR(FORCED);                                            /* PAGE 12-117  */|
|        | CALL CHANGE_MU_TO_POS_RSP(TRUNCATE);                                  /* APPENDIX B   */|
|        | SEND MU TO PC.ERC;                                                    /* CHAPTER 3    */|
|        | CALL RELEASE_VRCB;                                                    /* PAGE 12-117  */|
|--------|---------------------------------------------------------------------------------------------|
```

```
END FSM_VR;
```

```
FSM_DACTVR_DIRECTION: FSM_DEFINITION CONTEXT(VRCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS   FINITE-STATE   MACHINE   IS   USED   TO   DETERMINE   WHETHER │
│                 NC_DACTVR(ORDERLY) MAY BE  SENT FROM THIS END OF  THE VIRTUAL ROUTE. │
│                 NC_DACTVR(ORDERLY)  CANNOT  BE  SENT  WHEN  IN  RESET  STATE  OR  IN │
│                 CANNOT_SEND STATE.  NC_DACTVR(ORDERLY) CAN BE  SENT WHEN IN CAN_SEND │
│                 STATE.                                                   │
│                                                                          │
│      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
│                 ACTVR_RCV                          PAGE 12-96            │
│                 ACTVR_RQ_RCV                       PAGE 12-100           │
│                 FSM_VR                             PAGE 12-121           │
│                 SEND_DACTVR_ORDERLY                PAGE 12-106           │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAME ---->	RESET	CAN SEND	CANNOT SEND
STATE NUMBER -->	1	2	3
INPUT			
S, +RSP, ACTVR	3	3	-
R, -RSP, ACTVR	3	3	-
R, +RSP, ACTVR	2	-	2
S, -RSP, DACTVR	2	-	2
R, -RSP, DACTVR	3	3	-

```
END FSM_DACTVR_DIRECTION;
```

```
BUILD_NC_TH_RH: PROCEDURE(MSG_PTR);

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO BUILD  THE TRANSMISSION  HEADER AND REQUEST  HEADER OF  A NETWORK
 │                 CONTROL PIU                                          │
 │                                                                      │
 │      INPUT:     MSG_PTR ADDRESSES A REQUEST                          │
 │                                                                      │
 │      OUTPUT:    INITIALIZED TH AND  RH FIELDS IN THE  REQUEST.  UNINITIALIZED FIELDS
 │                 ARE INDICATED IN COMMENTS.                           │
 │                                                                      │
 │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
 │                 BUILD_ACTVR                        PAGE 12-116       │
 │                 BUILD_DACTVR                       PAGE 12-116       │
 │                 BUILD_NC_ER_ACT_OR_TEST            PAGE 12-68        │
 │                 INOP_SEND                          PAGE 12-42        │
 │                 OP_SEND                            PAGE 12-39        │
 │                 SEND_DACTVR_FORCED                 PAGE 12-107       │
 │                 SEND_DACTVR_ORDERLY                PAGE 12-106       │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL MSG_PTR PTR;

                                                                        /*
                    ┌──────────────────────────────────┐
                    │            TH FIELDS             │
                    └──────────────────────────────────┘
                                                                        */
  MSG_PTR->FID = 4;
                                                    /* TG SWEEP SET ELSEWHERE   */
  MSG_PTR->ER_VR_SUPP_IND =  ¬PRE_ER_VR;
  MSG_PTR->VR_PAC_CNT_IND =  ¬PAC_CNT_0;
  MSG_PTR->NTWK_PRTY =  ¬N_PRTY;
                                    /* IERN, ERN, VRN, TPF SET ELSEWHERE        */
  MSG_PTR->VR_CWI = INC_WS;
  MSG_PTR->TG_NONFIFO_IND = FIFO;
  MSG_PTR->VR_SQTI = NSEQ_NSUP;
  MSG_PTR->VRPRQ =  ¬VR_PAC_RQ;
  MSG_PTR->VRPRS =  ¬VR_PAC_RSP;
                                                    /* VR_CWRI SET ELSEWHERE    */
  MSG_PTR->VR_RWI =  ¬RESET_WS;
  MSG_PTR->VR_SNF_SEND = RESERVED_ZERO;
                                                    /* DSAF SET ELSEWHERE       */
  MSG_PTR->OSAF = NCB.NODE_SUBAREA_ADDRESS;
  MSG_PTR->DEF = 0;                                 /* PU ELEMENT ADDRESS       */
  MSG_PTR->OEF = 0;                                 /* PU ELEMENT ADDRESS       */
  MSG_PTR->SNAI = SNA;
  MSG_PTR->BBIUI = BBIU;                            /* MPF                      */
  MSG_PTR->EBIUI = EBIU;
  MSG_PTR->EFI = EXPEDITED;
  MSG_PTR->SNF = RESERVED_ZERO;

                                                                        /*
                    ┌──────────────────────────────────┐
                    │            RH FIELDS             │
                    └──────────────────────────────────┘
                                                                        */
  MSG_PTR->RRI = RQ;
  MSG_PTR->RU_CTGY = NC;
  MSG_PTR->FI = ON;
  MSG_PTR->SDI =  ¬SD;
  MSG_PTR->BCI = BC;
  MSG_PTR->ECI = EC;
  MSG_PTR->DR1I =  ¬DR1;            /* WILL BE SET FOR NC_ACTVR AND NC_DACTVR   */
  MSG_PTR->DR2I =  ¬DR2;                            /* BY THE VR MANAGER        */
  MSG_PTR->ERI =  ¬ER;
  MSG_PTR->QRI = RESERVED_ZERO;
  MSG_PTR->PI =  ¬PAC;
  MSG_PTR->BBI =  ¬BB;
  MSG_PTR->EBI =  ¬EB;
  MSG_PTR->CDI =  ¬CD;
  MSG_PTR->CSI = RESERVED_ZERO;
  MSG_PTR->EDI = RESERVED_ZERO;
  MSG_PTR->PDI = RESERVED_ZERO;

  MSG_PTR->MUCB.DIRECTION = SEND;

  RETURN;
END BUILD_NC_TH_RH;
```

```
BUILD_NS_RQN_RH: PROCEDURE(NS_MU_PTR);

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO BUILD THE RH OF AN RQN NETWORK SERVICES REQUEST.  │
  │                                                                      │
  │      INPUT:     MESSAGE UNIT ADDRESSED BY NS_MU_PTR PARAMETER        │
  │                                                                      │
  │      OUTPUT:    MESSAGE UNIT WITH RH FIELDS SET                      │
  │                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
  │                 NS_ER_INOP_SEND                 PAGE 12-47           │
  │                 ROUTE_TEST_RCV                  PAGE 12-113          │
  │                 TESTED_SEND                     PAGE 12-63           │
  │                 VR_INOP_SEND                    PAGE 12-110          │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL NS_MU_PTR PTR;

  NS_MU_PTR->RRI = RQ;
  NS_MU_PTR->RU_CTGY = FMD;
  NS_MU_PTR->FI = ON;
  NS_MU_PTR->SDI =  ¬SD;
  NS_MU_PTR->BCI = BC;
  NS_MU_PTR->ECI = EC;
  NS_MU_PTR->DR1I =  ¬DR1;
  NS_MU_PTR->DR2I =  ¬DR2;
  NS_MU_PTR->ERI =  ¬ER;
  NS_MU_PTR->QRI =  ¬QR;
  NS_MU_PTR->PI =  ¬PAC;
  NS_MU_PTR->BBI =  ¬BB;
  NS_MU_PTR->EBI =  ¬EB;
  NS_MU_PTR->CDI =  ¬CD;
  NS_MU_PTR->CSI =  CODE0;
  NS_MU_PTR->EDI =  ¬ED;
  NS_MU_PTR->PDI =  ¬PD;

  NS_MU_PTR->MUCB.DIRECTION = SEND;

  RETURN;
END BUILD_NS_RQN_RH;



VRN_TO_ERN_MAP: PROCEDURE(DEST_SA,VR_NUM,ER_NUM) RETURNS(BIT(1));

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO DETERMINE THE ER NUMBER OF THE ER THAT SUPPORTS A VR │
  │                                                                      │
  │      INPUT:     DEST_SA IS THE SUBAREA  ADDRESS OF THE NODE AT THE  OTHER END OF THE │
  │                 VR AND VR_NUM IS THE VR NUMBER THAT IS TO BE SUPPORTED │
  │                                                                      │
  │      OUTPUT:    THE ER NUMBER OF THE ER DEFINED TO SUPPORT THE VR (SECOND PARAMETER) │
  │                 TO THE DESTINATION SUBAREA NODE (FIRST PARAMETER) IS PUT INTO ER_NUM │
  │                 AND A BIT VALUE  OF EXIST OR ¬EXIST IS RETURNED  TO INDICATE WHETHER │
  │                 OR NOT THERE IS A VR TO ER MAPPING.                  │
  │                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
  │                 ACT_SEND                        PAGE 12-55           │
  │                 ACTVR_RQ_RCV                    PAGE 12-100          │
  │                 FIND_ER_STATUS                  PAGE 12-66           │
  │                 TEST_SEND                       PAGE 12-56           │
  │                 VR_RCV_CHECKS                   PAGE 12-98           │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL DEST_SA BIT(32);
  DCL VR_NUM BIT(4);
  DCL ER_NUM BIT(4);

  FIND ERN_MAP IN ERN_MAP_LIST WHERE(ERN_MAP.DEST_SA = DEST_SA); /* APPENDIX A          */

  IF ERN_MAP_PTR ¬= NULL THEN                                    /* APPENDIX A          */
    DO;
    . ER_NUM = ERN_MAP.ER_NUM(VR_NUM);                           /* APPENDIX A          */
    . RETURN(EXIST);
    END;
  ELSE
    RETURN(¬EXIST);

END VRN_TO_ERN_MAP;
```

```
ERN_TO_VRN_MAP: PROCEDURE(DEST_SA,VRN_MASK,ER_NUM) RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  TO DETERMINE THE VR NUMBERS OF  VR'S SUPPORTED BY ER'S DESIGNATED BY
  │                 AN ER NUMBER                                                 │
  │                                                                              │
  │      INPUT:     DEST_SA IS THE SUBAREA  ADDRESS OF THE NODE AT THE  OTHER END OF THE
  │                 ER AND  ER_NUM IS  THE ER  NUMBER CORRESPONDING  TO SOME  SET OF  VR
  │                 NUMBERS                                                      │
  │                                                                              │
  │      OUTPUT:    THE  SET  OF  VR  NUMBERS CORRESPONDING  TO  THE  ER  NUMBER  (THIRD
  │                 PARAMETER) TO THE DESTINATION SUBAREA  NODE (FIRST PARAMETER) IS PUT
  │                 INTO VRN_MASK (SECOND PARAMETER) AND A  BIT VALUE OF EXIST OR ¬EXIST
  │                 IS RETURNED TO INDICATE WHETHER OR NOT  THE DESIRED VR TO ER MAPPING
  │                 EXISTS.                                                      │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 SET_ER                                  PAGE 12-65           │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL DEST_SA BIT(32);
  DCL VRN_MASK BIT(16);
  DCL ER_NUM BIT(4);
  DCL VR_NUM BIT(4);


  FIND ERN_MAP IN ERN_MAP_LIST WHERE(ERN_MAP.DEST_SA = DEST_SA); /* APPENDIX A      */

  IF ERN_MAP_PTR ¬= NULL THEN                                    /* APPENDIX A      */
    DO;
    . VRN_MASK = ALL_OFF;
    . DO VR_NUM = 0 TO NCB.MAX_VR_NUM;                           /* APPENDIX A      */
    . . IF ERN_MAP.ER_NUM(VR_NUM) = ER_NUM THEN                  /* APPENDIX A      */
    . .    VRN_MASK(VR_NUM:VR_NUM) = ON;
    . END;
    . RETURN(EXIST);
    END;
  ELSE
    RETURN(¬EXIST);

END ERN_TO_VRN_MAP;
```

```
FSM_INPUT_DEFINITION::

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │     THE SYMBOLS  USED IN THE "INPUTS"  COLUMN OF THE STATE-TRANSITION  MATRICES ARE  │
 │     DEFINED BELOW.                                                           │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

      'ACTIVATE_ER'       FSMINPUT = 'ACTIVATE_ER'
      ACTVR               RQ_CODE = NC_ACTVR;
      BAD                 NC_ER_ACT_REPLY_RQ.TYPE = (X'01' | X'02' | X'04' | X'05' | X'06');
      CAN_SEND_O          FSM_DACTVR_DIRECTION = CAN_SEND;
      'CONTEND_RESEND'    FSMINPUT = 'CONTEND_RESEND'
      DACTVR              RQ_CODE = NC_DACTVR;
      DACTVR_F            RQ_CODE = NC_DACTVR & NC_DACTVR_RQ.TYPE = FORCED;
      DACTVR_O            RQ_CODE = NC_DACTVR & NC_DACTVR_RQ.TYPE = ORDERLY;
      'DEFINE'            FSMINPUT = 'DEFINE'
      EMPTY_PATHCB        EMPTY(PATHCB_LIST) = YES;
      'ER_ACTIVATED'        FSMINPUT = 'ER_ACTIVATED'
      'ER_NOT_ACTIVATED'   FSMINPUT = 'ER_NOT_ACTIVATED'
      GOOD                NC_ER_ACT_REPLY_RQ.TYPE = (X'00'|X'03');
      LOCAL               VRCB.PARTNER_SA = NCB.NODE_SUBAREA_ADDRESS;
      LOWER_SA            OSAF < NCB.NODE_SUBAREA_ADDRESS;
      NC_ER_ACT           RQ_CODE = NC_ER_ACT;
      NC_ER_ACT_REPLY     RQ_CODE = NC_ER_ACT_REPLY;
      NC_ER_INOP          RQ_CODE = NC_ER_INOP;
      NC_ER_OP            RQ_CODE = NC_ER_OP;
      NO_SESS             VRCB.SESS_COUNT = 0;
      NOT_REV             NC_ER_ACT_REPLY_RQ.TYPE = (X'02');
      OTHERS_PENDING      ARE_ANY_PATHS_PENDING = YES;        /* PAGE 12-72  */
      PATH                PATHCB_PTR ¬= NULL;
      PRE_VR              VRCB.ER_VR_SUPP = PRE_ER_VR;
      'PRE_VR_ACT'        FSMINPUT = 'PRE_VR_ACT';
      R                   MUCB.DIRECTION = RECEIVE;
      RACE                NC_ER_ACT_REPLY_RQ.TYPE = (X'01');
      'RESET'             FSMINPUT = 'RESET';
      RQ                  RRI = RQ;
      +RSP                RRI = RSP & RTI = POS;
      -RSP                RRI = RSP & RTI = NEG;
      S                   MUCB.DIRECTION = SEND;
      'SEND_DACTVR_F'     FSMINPUT = 'SEND_DACTVR_F';
      'SESS_COUNT=0'      FSMINPUT = 'SESS_COUNT=0'
      SPRAY               NC_ER_ACT_RQ.DYNAMIC_ER_DEFN = YES;
      'SPRAY'             FSMINPUT = 'SPRAY'
      STATIC              SUBAREA_ROUTING.ER_SYSDEF(ERCB.ER_NUM) = STATIC_DEFINITION;
      TG_ID               SUBAREA_ROUTING.TG_ID(ERCB.ER_NUM) = PATHCB.TG_ID;
      'VRINOP'            FSMINPUT = 'VRINOP';
      WAIT_SESS           ¬EMPTY(VRCB.VR_RESERVATION_LIST);
      WINNER              NCB.NODE_SUBAREA_ADDRESS > NC_ER_ACT_RQ.ORIGINATING_SA;
      0815                SNC = X'0815';


       END FSM_INPUT_DEFINITION;
```

```
/*
+-------------------------------------------------------------------+
|                                                                   |
|              EXPLICIT ROUTE ACTIVATION PARAMETERS (PARM_ACT_ER)   |
|                                                                   |
|   FUNCTION:  THIS ENTITY CONTAINS  THE INFORMATION PASSED BETWEEN |
|              AND ER  MANAGER IN ORDER TO  ACTIVATE THE EXPLICIT   |
|              ONE  OR  SEVERAL VIRTUAL  ROUTES.  AN  INSTANCE OF  |
|              CREATED BY THE VR MANAGER WHEN IT INVOKES THE ER MANAGER TO ACTIVATE
|              THE ER  THAT SUPPORTS THE SPECIFIED  VR TO THE  DESTINATION SUBAREA;
|              THE  ER  MANAGER  DESTROYS  THE  PARM_ACT_ER  AFTER  INITIATING  THE
|              ACTIVATION  PROCESS FOR  THE  UNDERLYING  ER.  AFTER  ATTEMPTING  TO
|              ACTIVATE THE UNDERLYING ER, THE  ER MANAGER CREATES ANOTHER INSTANCE
|              OF  PARM_ACT_ER  TO  SIGNAL  THE  VR  MANAGER  WHETHER  OR  NOT  THE
|              UNDERLYING ER  HAS BEEN ACTIVATED, AND  WHICH VR'S IT  WILL SUPPORT.
|              THE  VR  MANAGER  DESTROYS  THE  PARM_ACT_ER  AFTER  PROCESSING  ITS
|              CONTENTS.                                            |
|                                                                   |
+-------------------------------------------------------------------+
*/

DCL PARM_ACT_ER_PTR PTR;

ENTITY(PARM_ACT_ER),
      2 PARTNER_SA                  BIT(32),     /* SUBAREA NODE AT OTHER END OF THE ER     */
      2 VRN_MASK                    BIT(16);     /* VIRTUAL ROUTE NUMBER MASK               */
```

```
/*
┌──────────────────────────────────────────────────────────────────────┐
│ FUNCTION:  THIS ENTITY PROVIDES PARAMETERS PASSED FROM THE ER MANAGER TO THE VR │
│            MANAGER REGARDING ONE OR MORE INOPERATIVE ER'S.           │
└──────────────────────────────────────────────────────────────────────┘
*/


DCL PARM_ER_INOP_PTR PTR;

ENTITY(PARM_ER_INOP),
      2 REASON_CODE BIT(8),          /* X'01' UNEXPECTED ROUTING INTERRUPTION   */
                                     /*       OVER A TRANSMISSION GROUP,        */
                                     /*       E.G., THE LAST ACTIVE LINK        */
                                     /*       IN A TG HAS FAILED                */
                                     /* X'02' CONTROLLED ROUTING INTERRUPTION,  */
                                     /*       E.G., THE RESULT OF DISCONTACT    */
      2 ORIGINATING_SA BIT(32),      /* ADDRESS OF THE PU THAT ORIGINATED       */
                                     /* THE NC_ER_INOP                          */
      2 TG_ADJ_SA BIT(32),           /* SUBAREA ADDRESS ON OTHER END            */
                                     /* OF THE TRANSMISSION GROUP THAT          */
                                     /* HAD THE ROUTING INTERRUPTION            */
      2 TG_NUM BIT(8),               /* TGN OF THE TRANSMISSION GROUP           */
                                     /* THAT HAD THE ROUTING INTERRUPTION       */
      2 CNT_ER_FIELD BIT(8),         /* NUMBER OF DESTINATION SUBAREAS THAT     */
                                     /* ARE ON THE ER'S USING THE ABOVE TG      */
      2 ER_FIELD (1:REFER (CNT_ER_FIELD)),
        3 SA BIT(32),                /* SUBAREA ADDRESS OF A DESTINATION        */
                                     /* ROUTED TO USING AN ER REQUIRING THE     */
                                     /* TG THAT HAD THE ROUTING INTERRUPTION    */
        3 MASK BIT(16);              /* EXPLICIT ROUTE NUMBER MASK:             */
                                     /* A BIT IS 1 IF THE CORRESPONDING ERN     */
                                     /* IS INOPERATIVE                          */
                                     /* (BIT 0 CORRESPONDS TO ERN 0,            */
                                     /* BIT 1 TO ERN 1, AND SO FORTH).          */
```

COMMON SESSION CONTROL MANAGER

Each PU.SVC_MGR contains a component, called common session control manager, that is invoked for activation and deactivation of all locally supported half-sessions and boundary function supported half-sessions.

Common session control manager (CSC manager) is composed of five protocol machines--CSC_MGR.SEND, CSC_MGR.BF_SEND, CSC_MGR.RCV, CSC_MGR.BF_RCV, and CSC_MGR.SON--as shown in Figure 13-2. All activation and deactivation requests (ACTCDRM, ACTLU, ACTPU, BIND, DACTCDRM, DACTLU, DACTPU, and UNBIND) and their responses destined for locally supported half-sessions or boundary function supported half-sessions are directed to CSC manager by path control (Chapter 3) or by the services managers. The flow of requests/responses through CSC manager for the support of paired half-sessions is shown in Figure 13-3. The flow of requests/responses through CSC manager for the support of paired half sessions with one half-session being supported by boundary function is shown in Figure 13-4.

CSC_MGR.SEND, CSC_MGR.BF_SEND, CSC_MGR.RCV, and CSC_MGR.BF_RCV perform the following functions:

- Verify the activation or deactivation request or response is valid for the NAU (e.g., an ACTLU is prevented from being sent from any NAU except the SSCP, and an ACTLU to any NAU other than an LU receives a negative response).

- Perform checking of some parameters (TS profile and FM profile related) contained in the activation or deactivation request or response.

- Make state dependent checks on received or sent requests or responses.

- Create and destroy a session control block (SCB), if necessary. At least two session control blocks exist for a session, one at the primary half-session and one at the secondary half-session. If the secondary half-session is supported by boundary function a third session control block exists for the boundary function supporting the secondary half-session.

- Call the appropriate FSMs to update the state of the locally supported half-session or boundary function supported half-session.

A session control block (SCB) exists for each locally supported half-session or boundary function supported half-session that is not reset. The SCB provides storage for half-session variables and for pointers to the half-session lists, queues and FSM states. When an activation request is sent, the SCB for the sending half-session is created by CSC_MGR.SEND, unless one already exists from a previous activation. When an activation request is received, the SCB for the receiving half-session (local or BF supported) is created by CSC_MGR.RCV, unless one already exists from a previous activation. The SCB is deleted when the session is deactivated.

The SESS FSMs are used for session activation and deactivation and are contained within this chapter. These FSMs are actually contained within the locally supported half-session or boundary function supported half-session but are defined here, as CSC manager is the only protocol machine that updates the states of these FSMs. The SESS FSMs may cause the deletion of the SCB when there is a transition from a nonreset state to the reset state.

When a SESS FSM is called by CSC manager with an activation request or a positive response to an activation request, as the state transition occurs, another composite protocol machine is invoked, session activation parameters, see the section "Session Activation Parameters Prococol Machine (SESSACT)," later in this chapter. When a SESS FSM is called by CSC manager with a response to a deactivation request or a reset signal, the SCB is discarded. Discarding of the SCB has the effect of resetting all locally supported half-session or boundary function supported hal.-session FSMs, removing all entries from the queues and lists, and resetting all variables.

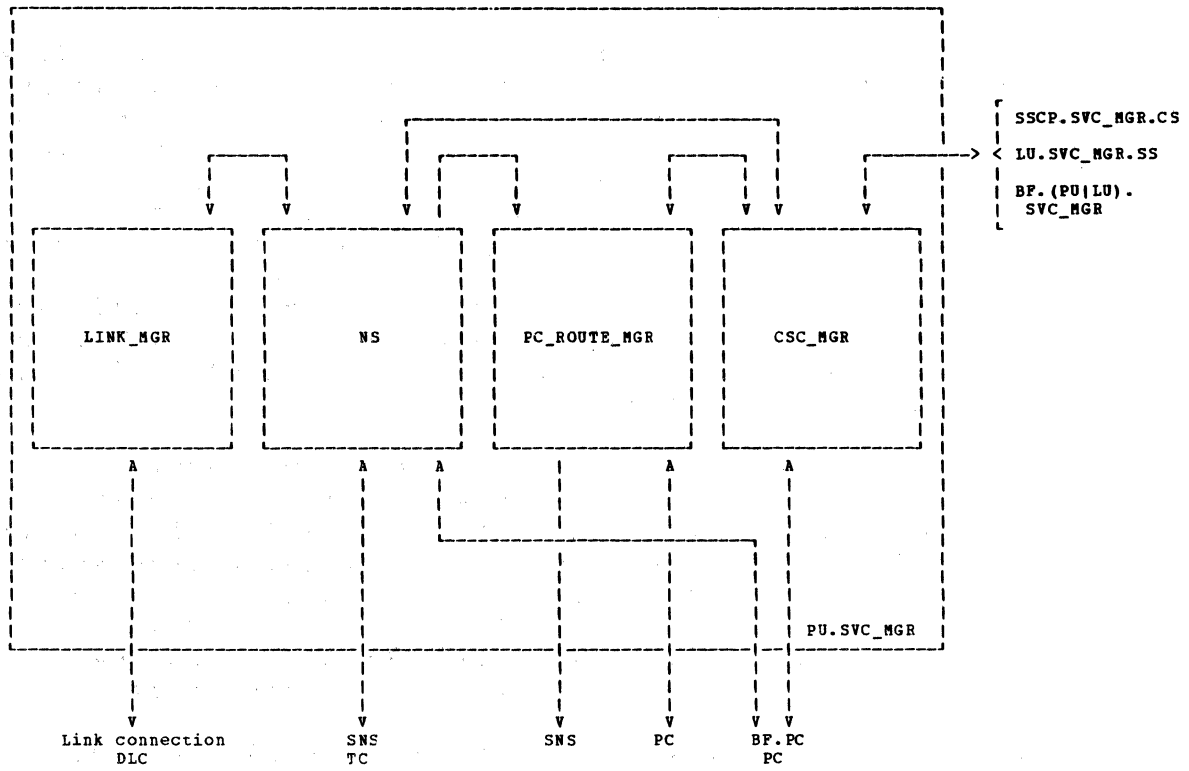


Figure 13-1. Overview of PU.SVC_MGR

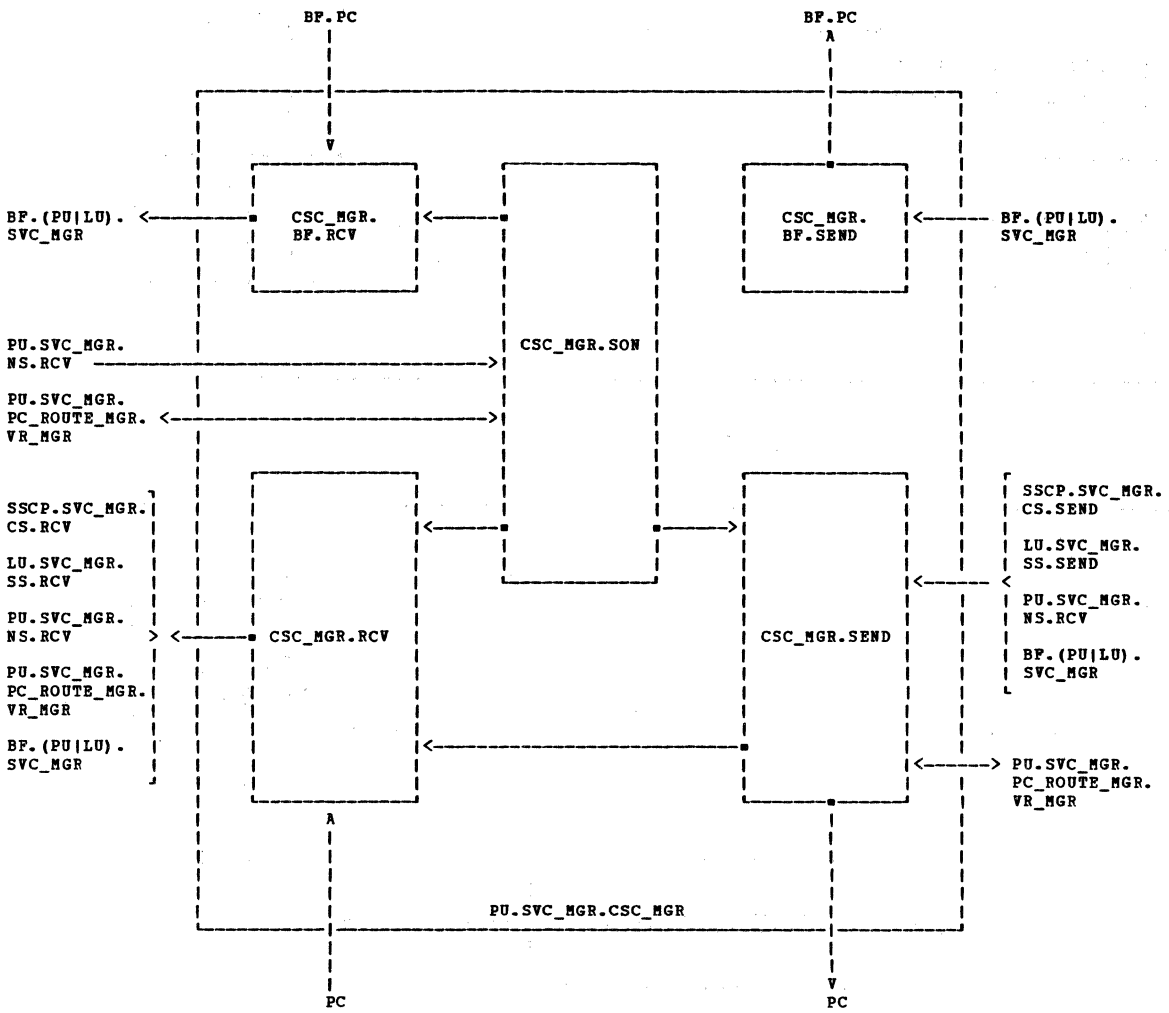
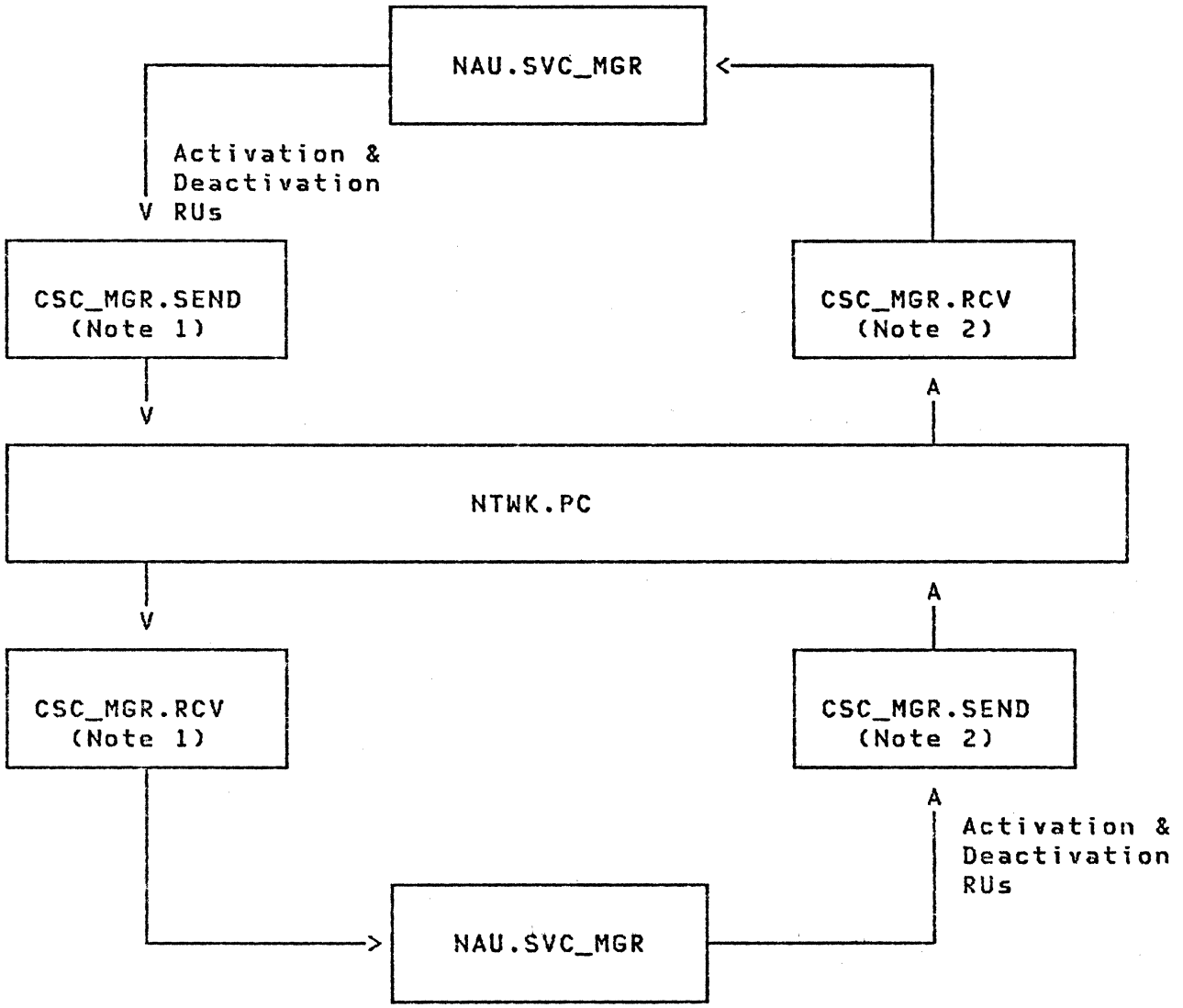


Figure 13-2. Structure of PU.SVC_MGR.CSC_MGR

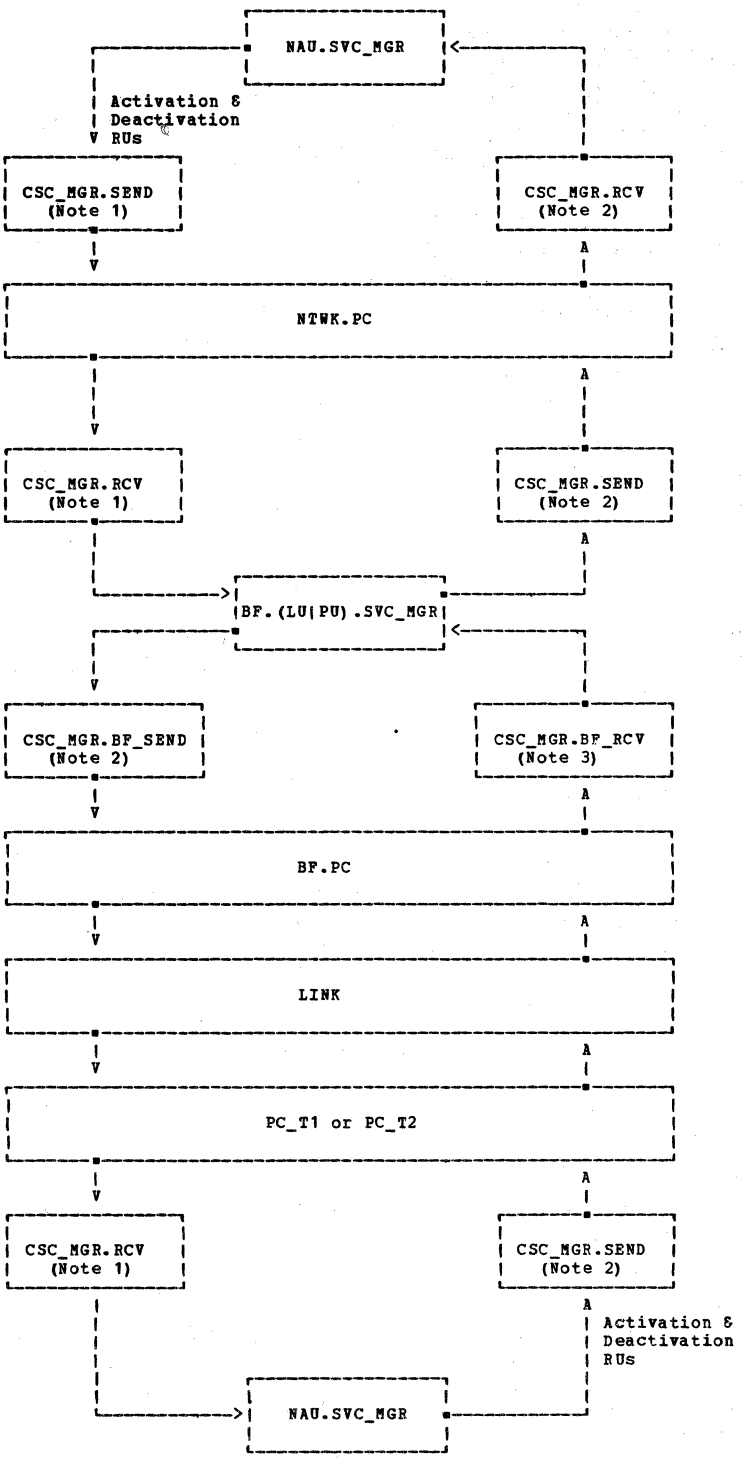


NOTES:

1. Session control block is created on sending or receiving a valid activation request.

2. Session control block is discarded upon sending or receiving a deactivation response via the SESS FSM, or by receiving a negative response to an activation request.

Figure 13-3. Typical flow through CSC_MGR (for locally supported half-sessions)



NOTES:

1. Session control block is created on sending or receiving a valid activation request.
2. Session control block is discarded upon sending or receiving a deactivation response via the SESS FSM, or receiving a negative response to an activation request.
3. Session control blocks are neither created nor destroyed in this instance.

Figure 13-4. Typical flow through CSC_MGR (local and boundary function supported half-sessions).

SESSION OUTAGE NOTIFICATION PROCESSING

Session outage notification (SON) notifies half-sessions of network failures by driving the half-sessions into reset state. A given session may be restarted from this reset state, depending upon the specific cause of the outage. Session deactivation RUs (DACTCDRM, DACTLU, DACTPU, and UNBIND) perform session outage notification, and inform the NAU of the type of outage. The SON RUs are generated by a component of CSC manager called CSC_MGR.SON.

The network notifies CSC_MGR.SON of conditions that may disrupt traffic flow between half-sessions. These network states may be the result of failures (e.g., link outage) or they may be caused by the specific actions of an SSCP (e.g., deactivation of an SSCP-PU session). Depending upon the specific cause of the outage, CSC_MGR.SON identifies the affected sessions and sends session deactivation requests to each half-session that is accessible to CSC_MGR. Session outage notification for each session always flows along the path that the session used.

The following conditions result in session outage notification processing:

- Virtual route inoperative: Failure of an explicit route (see Chapter 12) causes NC_ER_INOP RUs to propagate along explicit routes and notify virtual route managers (VR_MGR) having virtual routes using the inoperative explicit route. The virtual route is declared inoperative, thus disrupting sessions that were using it. The VR_MGR notifies CSC_MGR.SON of this condition.

- Virtual route deactivated: A virtual route is unconditionally deactivated, thus disrupting sessions that were using it. The VR_MGR notifies CSC_MGR.SON of this condition.

- Route extension inoperative: The link connecting a PU_T4|5 node to a PU_T1|2 node becomes inoperative, thus disrupting sessions that were using it. The PU.SVC_MGR (Chapter 11) notifies CSC_MGR.SON of these conditions.

- Hierarchical Reset or SSCP Gone: A hierarchical reset takes place because: 1) an SSCP attempts to reactivate its session with a PU or LU but the PU or LU responds Cold, thus triggering the hierarchical reset of underlying sessions; 2) an SSCP deactivates its session with a PU or LU, thus triggering the reset of any underlying sessions. CSC_MGR.SEND (via the FSMs that it calls) notifies CSC_MGR.SON of this condition.

CSC_MGR.SON performs the processing for session outage notification conditions as described in the following sections.

Virtual Route Inoperative

For each session that uses the VR that has become inoperative, CSC_MGR.SON generates a deactivation request (DACTCDRM, DACTLU, DACTPU, or UNBIND) and sends it to the half-session in its subarea, with the cause of the deactivation indicating VRINOP. CSC_MGR.SON identifies all the affected sessions by finding the SCBs having the same VRID as the inoperative VR. The response to the deactivation request is intercepted by CSC_MGR.SEND and discarded, since the request was generated by CSC_MGR. CSC_MGR.SON is entered as a result of the VR_MGR setting the VRCB_PTR to point to the inoperative VR, and sending "VRINOP" to CSC_MGR.SON.

Virtual Route Deactivated

The CSC manager processing is identical to the virtual route inoperative case. CSC_MGR.SON is entered as a result of the VR_MGR sending "DACTVR_FORCED" (with the VRCB_PTR) to CSC_MGR.SON.

Route Extension Inoperative

For each LU-LU session that uses the route extension that has become inoperative, CSC_MGR.SON generates a deactivation request and sends it to the half-session that is still accessible, with the cause of deactivation indicating REX_INOP. The response to the UNBIND terminates in common session control manager (CSC_MGR.SEND). SSCP-based sessions using the inoperative route extension are reset without explicit session outage notification to the SSCP's half-sessions. (The SSCP's half-sessions are reset by the SSCP.SVC_MGR as a result of receiving an INOP from the PU_T4|5 node.) CSC_MGR.SON is entered as a result of the PU.SVC_MGR (Chapter 11) sending "REX_INOP(ALS_EA)" to CSC_MGR.SON.

Hierarchical Reset or SSCP Gone

For each LU-LU session that is to be reset as part of the hierarchical reset (because of DACTPU or DACTLU, or, a Cold response to ACTLU or ACTPU), CSC_MGR.SON generates an UNBIND request. See Figure 13-5 for more details.

SSCP Gone

DACTPU received by a PU_T4|5 node does not result in the resetting of any underlying LU-LU sessions.

DACTPU received by a PU_T2 node or BF supporting a PU_T1 node results in the deactivation of the SSCP-LU and LU-LU sessions for all LUs belonging to the reset hierarchy of the PU. CSC_MGR.SON generates and sends a deactivation request to each SSCP-LU and LU-LU half-session in its node. CSC_MGR.SON in a PU_T4|5 node, providing BF support for the node of the subject PU, generates and sends an UNBIND request to the primary half-session of each LU-LU session of each LU belonging to the subject PU. For BF supporting a PU_T1 node, UNBIND is sent to the secondary half-session in the PU_T1.

DACTLU results in the deactivation of all LU-LU sessions of the subject LU. In a PU_T1|2 node, CSC_MGR.SON generates and sends an UNBIND to each LU-LU half-session in its node, for the subject LU. In a PU_T4|5 node, CSC_MGR.SON generates and sends an UNBIND to both half-sessions of each LU-LU session for the subject LU, in its node. If the subject LU is in a PU_T1|2 node, the CSC_MGR.SON in the PU_T4|5 node (providing BF support) generates and sends an UNBIND to the primary half-session of each LU-LU session of the subject LU.

CSC_MGR.SON is scheduled when CSC_MGR.SEND, while processing RSP(DACTPU|DACTLU), sends "SSCP_GONE" to CSC_MGR.SON.

The sending of RSP(ACTPU,Cold) by a PU_T2 node or the BF supporting a PU_T1 node results in the deactivation of the SSCP-LU and LU-LU sessions for all LUs belonging to the reset hierarchy of the PU. CSC_MGR.SON in the PU_T2 node generates and sends a deactivation request to each SSCP-LU and LU-LU half-session in its node. CSC_MGR.SON in a PU_T4|5 node providing BF support for the PU_T1|2 node, generates and sends an UNBIND to the primary half-session of each LU-LU session of each LU belonging to the subject PU_T1|2 node. For BF supporting PU_T1 nodes, UNBIND is sent to the secondary half-session.

The sending of RSP(ACTPU,Cold) by a PU_T4|5 node results in the deactivation of the SSCP-LU and LU-LU sessions for all LUs belonging to the reset hierarchy of the PU. CSC_MGR.SON in the PU_T4|5 node generates and sends the deactivation requests.

The sending of RSP(ACTLU,Cold) by an LU in a PU_T1|2 node, results in the deactivation of all LU-LU sessions of the subject LU. CSC_MGR.SON in the PU_T1|2 node generates and sends an UNBIND to each LU-LU half-session in its node for the subject LU. CSC_MGR.SON in a PU_T4|5 node providing BF support for the PU_T1|2 node LU, generates and sends an UNBIND to the primary half-session of each LU-LU session of the subject LU.

The sending of RSP(ACTLU,Cold) by an LU in a PU_T4|5 node also results in the deactivation of all LU-LU sessions of the subject LU. CSC_MGR.SON in the PU_T4|5 node generates and sends an UNBIND to each half-session of each LU-LU session for the subject LU.

In each case above, CSC_MGR.SON is entered when CSC_MGR.SEND, while processing RSP(ACTPU,Cold) or RSP(ACTLU,Cold), sends "HIERARCHICAL_RESET" to CSC_MGR.SON.

EVENT	ACTION
RSP(ACTLU,Cold) sent by PU_T1|2	CSC_MGR.SON in the node providing BF support, resets the secondary LU-LU half-session by sending UNBIND to itself. RSP(UNBIND) is discarded in FSM.
RSP(ACTLU,Cold) received by BF	CSC_MGR.SON in the node providing BF support, sends UNBIND to the PLUs; receipt of RSP(UNBIND) resets the BF. RSP(UNBIND) is not forwarded to the SLU.
RSP(ACTLU,Cold) received by SSCP	No effect.
RSP(ACTPU,Cold) sent by PU_T2	PU_T2 resets secondary LU-LU half-session by sending UNBIND to itself. The secondary SSCP-LU half-sessions are reset by sending DACTLU to itself. Responses are discarded by the FSMs.
RSP(ACTPU,Cold) sent by CSC_MGR.SON in the BF node supporting a PU_T1	CSC_MGR.SON in the node providing BF support, sends UNBIND to the PLUs; receipt of RSP(UNBIND) resets the BF. CSC_MGR.SON in the node providing BF support, sends UNBIND to the SLUs and sends DACTLU to the secondary SSCP-LU half-session.
RSP(ACTPU,Cold) received by BF from a PU_T2	CSC_MGR.SON in the node providing BF support, sends UNBIND to the PLUs; receipt of RSP(UNBIND) resets the BF. CSC_MGR.SON in the node providing BF support, creates and sends DACTLU to itself resetting the SSCP-BF.LU session. The DACTLU is discarded in the BF FSMs
RSP(ACTPU,Cold) received by SSCP	The SSCP.SVC_MGR sends DACTLU(Cleanup) to the PU_T1|2 node resetting the primary SSCP-LU half-session. The DACTLU(Cleanup) is discarded by the BF
DACTLU received by PU_T1|2	Same as sending RSP(ACTLU,Cold).
DACTLU received by BF	Same as receiving RSP(ACTLU,Cold) from PU_T1|2.
DACTLU sent by SSCP	Is not involved in SON.
RSP(ACTLU,Cold) sent by an LU in a subarea node	Send UNBIND to the PLUs, send UNBIND to itself to reset the secondary LU-LU half-sessions. The RSP(UNBIND) generated by the SLU is discarded in the SLU's FSM.
RSP(ACTLU,Cold) received by SSCP from LU in a subarea node.	Is not involved in SON
RSP(ACTPU,Cold) sent by a PU in a subarea node.	Send DACTLU to all LUs having an active session with the PU's SSCP. Send UNBIND to all the LUs having an active session with this LU. Send UNBIND to this LU for each UNBIND sent to the partner LUs. The UNBIND sent to this LU is discarded by the FSM.
RSP(DACTPU) sent by PU_T4|5	Underlying LU-LU sessions are not reset. All underlying SSCP-LU, SSCP-BF.LU, and SSCP-BF.PU sessions are reset by sending DACTLU and DACTPU
DACTPU received by the BF supporting a PU_T1.	Same as sending RSP(ACTPU,Cold) on behalf of a PU_T1.
RSP(DACTPU) received by BF supporting a PU_T2.	Same as BF receiving RSP(ACTPU,Cold) from PU_T2.
RSP(DACTPU) sent by PU_T2.	The PU_T2 resets all secondary LU-LU half-sessions by sending UNBIND to itself. The PU_T2 resets all secondary SSCP-LU half-sessions by sending DACTLU to itself.

NOTES:

1. It is not uncommon for CSC_MGR to send deactivation requests to itself. The reason CSC_MGR does this is to route the request through the appropriate SVC_MGR, allowing the SVC_MGR to update its FSMs.

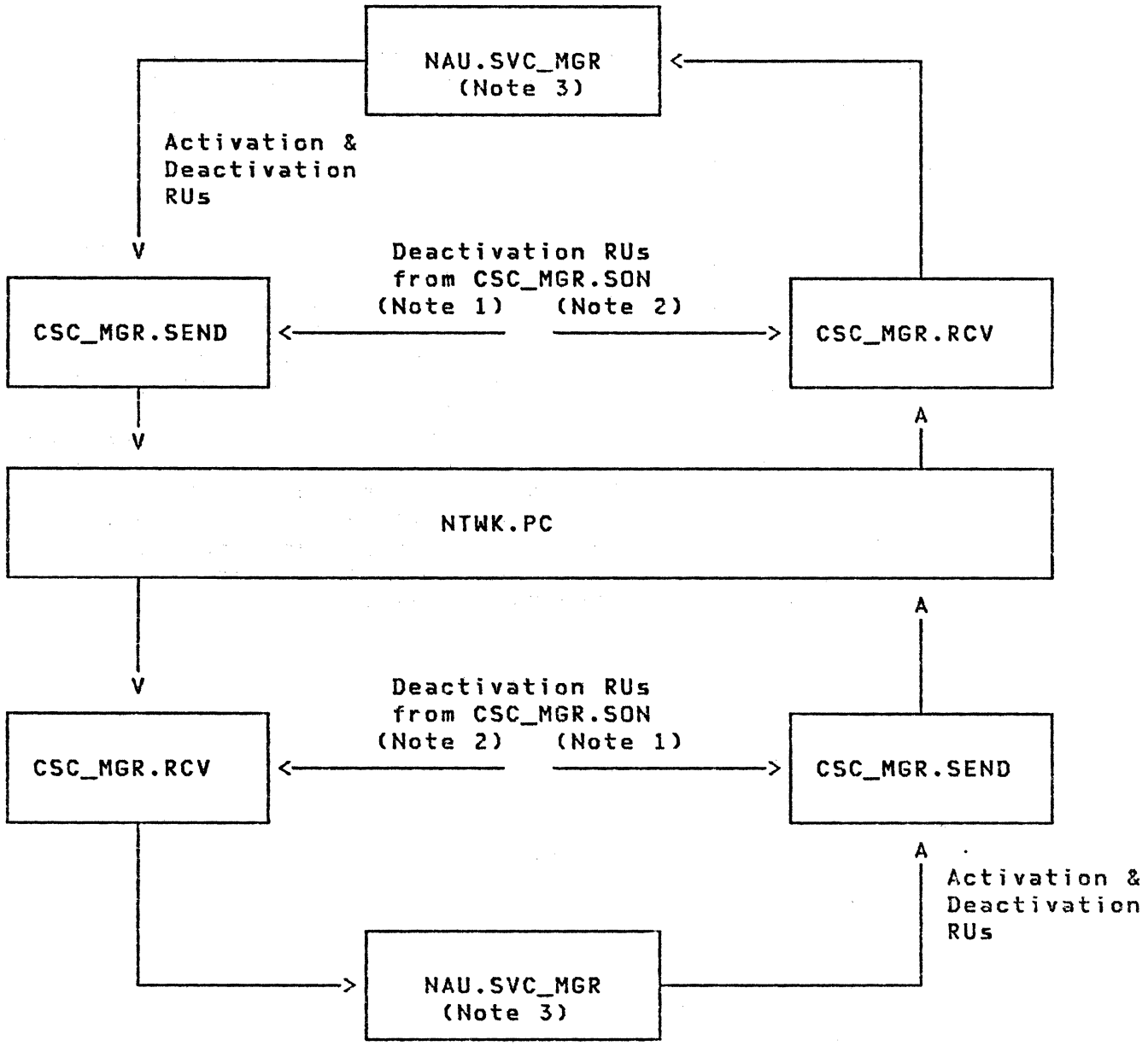
2. ACTPU and DACTPU are not sent to PU_T1s.

Figure 13-5. Reset table for the signals SSCP_GONE and HIERARCHICAL_RESET.

SESSION ACTIVATION PARAMETERS PROTOCOL MACHINE (SESSACT)

CSC_MGR.SEND|RCV obtains the SCB for the locally supported or boundary function supported half-session; SESSACT retains the session activation parameters carried on the session activation request and response, and initializes the states of the half-session.

The SCB (Appendix A) has a common format for all half-sessions. A portion of the SCB corresponds to that portion of the BIND RU containing the FM, TS, and PS profile and usage fields. The SESSACT procedures map the parameters from the activation request and response into portions of the SCB. SESSACT.TC_INITIALIZE (Chapter 4) and SESSACT.DFC_INITIALIZE (Chapter 5) are called by SESSACT response-sending and -receiving procedures. These SESSACT procedures use the information contained in the SCB to set up parameters for TC and DFC use while the session is active; these parameters are also saved in the SCB. For example, the maximum RU size parameters are encoded in the activation RUs; instead of decoding these parameters each time they are needed while the session is active, they are decoded once and the results are saved.



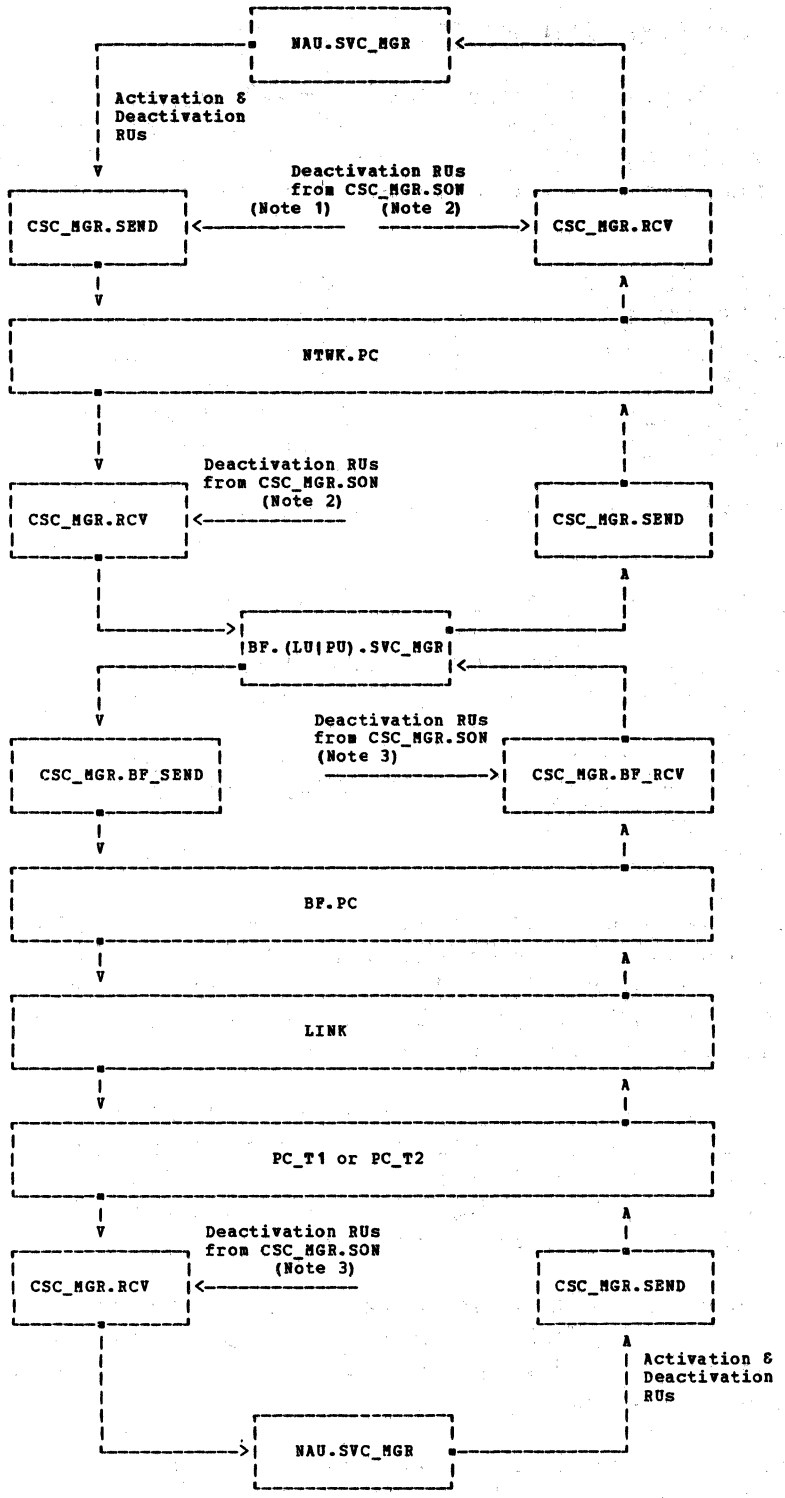
NOTES:

1. The session outage notification cause on these deactivation RUs is HIERARCHICAL_RESET or SSCP_GONE.

2. The session outage notification cause on these deactivation RUs is VRINOP, DACTVR_FORCED, or SESSION_OVERRIDE.

3. The NAUs within this figure are contained in a PU_T4|5 node. For NAUs supported by boundary function see Figure 13-7.

Figure 13-6. Flow through CSC_MGR (PU_T4|5 NAUs with SON)



NOTES:

1. Deactivation request with session outage notification cause of HIERARCHICAL_RESET or SSCP_GONE.

2. Deactivation request with session outage notification cause of VRINOP, or DACTVR_FORCED.

3. Deactivation request with session outage notification cause of REX_INOP, HIERARCHICAL_RESET, or SSCP_GONE.

Figure 13-7. Flow through CSC_MGR (local and boundary function half-sessions) with session outage notification.

SESSION ACTIVATION AND DEACTIVATION PROTOCOLS

The activation status of each half-session is indicated by the state of an appropriate FSM. This section includes the session-status (SESS) FSM definitions for SSCP-PU, SSCP-LU, and LU-LU primary and secondary half-sessions; their names are:

The session-status FSM for both SSCP-SSCP half-sessions is also define d:

The session status for SSCP-PU, SSCP-LU, and LU-LU boundary function supported half-sessions are also defined:

Along with the session activation RU, the SSCP.SVC_MGR or LU.SVC_MGR passes to CSC manager the Class of Service Name and virtual route identifier list (see Appendix A) to be used by the PU.SVC_MGR.PC_ROUTE_MGR.VR_MGR (Chapter 12) in assigning a virtual route (VR) for the new session.

The following is an overview of the use of Class of Service name and virtual route identifier list in the activation of a session. The detailed logic is described later in this chapter and under the VR manager. The following is from the point of view of the primary half-session.

- CSC_MGR, having received a session activation request from the NAU.SVC_MGR, creates an SCB, and sends the activation request to the VR manager for the assignment of a virtual route.

- An installation specific algorithm reorders the virtual route list using the COS name and the network address pair for the session. The VR manager selects the first available virtual route, activates it if it is not already active, and sets a global pointer (VRCB_PTR) pointing to the virtual route control block. The virtual route control block represents the virtual route to be used by the session being activated. See VR manager (Chapter 12) for more details.

CSC manager builds the session control block (SCB), associates it with the assigned virtual route control block (VRCB), and sends the session activation request to path control. If the VR manager cannot assign a virtual route, the VR manager returns a negative response to CSC_MGR. CSC manager forwards the -RSP to the SSCP.SVC_MGR or LU.SVC_MGR.

ACTIVATE CROSS-DOMAIN RESOURCE MANAGER (ACTCDRM)
DEACTIVATE CROSS-DOMAIN RESOURCE MANAGER (DACTCDRM)

Flow: From SSCP to SSCP (Expedited)

Principal FSM:
FSM_SESS_SSCP_SSCP_PRI_OR_SEC (Page 13-91)

ACTCDRM is sent from one SSCP to another SSCP to activate a session between them and to exchange information about the SSCPs (such as contents ID). By sending ACTCDRM, a half-session indicates its intention to assume the role of primary; the half-session receiving ACTCDRM is requested to assume the role of secondary.

Since any SSCP may send ACTCDRM, it may happen that two SSCPs send ACTCDRM to each other at the same time. In this case, each CSC manager receives an ACTCDRM request from the other SSCP before it receives the response for the ACTCDRM request it sent for its own SSCP. To resolve this contention situation, each CSC manager compares the SSCP ID of the ACTCDRM it has sent to the SSCP ID of the ACTCDRM received. The sender of the greater SSCP ID is the ACTCDRM contention winner; that CSC manager sends a NAU Contention negative response (X'080D') to the other SSCP's ACTCDRM. If the two ACTCDRMs traversed the same virtual route, the receiver of the ACTCDRM containing the greater SSCP ID processes the request as if it had never sent ACTCDRM. If the two ACTCDRMs traversed different virtual routes, the receiver of the greater SSCP ID generates and sends a DACTCDRM (with an SON code = X'10') to the SSCP contention winner over the same virtual route on which the contention-losing ACTCDRM was sent. DACTCDRM is sent to reset a half-session in the case of an inoperative virtual route. Subsequently, the receiver of the contention-winning ACTCDRM receives and discards the negative response to the losing ACTCDRM. If the SSCP IDs are equal, both SSCPs send and receive negative responses; the contention is then resolved by the network operators.

When an SSCP is attempting to establish a session, the ACTCDRM may be in the VR reservation list waiting for the activation of a VR. It is possible for the session partner SSCP (the destination of the ACTCDRM in the reservation list), to send an ACTCDRM. When CSC manager receives the ACTCDRM, CSC manager accepts that ACTCDRM, even if it would be the contention loser, unless the virtual route has already been established. If the VR has already been established, the normal contention situation processing occurs.

If the activation request/response sequence identifier in the SCB is less than the one in the ACTCDRM, the received request is more recent and the session is to be overridden.

For ACTCDRM, the session is overridden by the SSCP.SVC_MGR performing the resynchronization of the SSCP-SSCP session (see Chapter 7).

The type of session activation requested may be either Cold or ERP (error recovery procedure). The type of session activation actually performed by the SSCP is indicated on the response. If Cold is requested, only Cold session activation is allowed. If ERP is requested, either ERP or Cold session activation may be performed. The parameters and rules to be used while the SSCP-SSCP session is active are indicated by the FM and TS profiles (see Appendix F) and the TS Usage field.

The CDRM control vector and the Activation Request/Response Sequence Identifier control vector are carried in ACTCDRM and RSP(ACTCDRM); RSP(ACTCDRM) may also carry the Vector Keys Not Recognized control vector. The CDRM control vector contains the CDRM profile number and CDRM Usage field (see Appendixes E and F). They are exchanged between SSCPs in order to convey the functional capability of each SSCP to the other.

The Activation Request/Response Sequence Identifier control vector for ACTCDRM is created by the SSCP.SVC_MGR. The Activation Request/Response Sequence Identifier is used by the receiver of ACTCDRM to determine whether the current ACTCDRM supersedes a previously received ACTCDRM or RSP(ACTCDRM) from the same sender (see Appendix E).

A new Activation Request/Response Sequence Identifier control vector is generated by the SSCP.SVC_MGR receiving ACTCDRM and placed in the RSP(ACTCDRM). The Activation Request/Response Sequence Identifier control vector is used by the receiver of RSP(ACTCDRM) to determine whether the current RSP(ACTCDRM) supersedes a previously received ACTCDRM or RSP(ACTCDRM) from the same sender. Generating a new Activation Request/Response Sequence Identifier control vector for RSP(ACTCDRM) facilitates the restart of SSCP-SSCP sessions that have failed due to routes becoming inoperative.

The Vector Keys Not Recognized control vector in RSP(ACTCDRM) specifies those control vector key values that were received in ACTCDRM but not recognized by the receiver.

ACTCDRM(Cold) may not be sent by an SSCP unless the SESS_SSCP_SSCP_PRI_OR_SEC FSM is in the reset state. On the receive side, ACTCDRM(Cold) causes the reset of the sessions belonging to the reset hierarchy of the secondary SSCP, the deactivation of any cross-domain LU-LU sessions between LUs in the domains of the two SSCPs, and the purging of queued INIT and CDINIT requests for LU-LU sessions between the two domains. The LU-LU half-sessions are deactivated by an SSCP by sending:

- CLEANUP to its LUs associated with a PU_T4|5 node and acting as SLUs.

- CTERM(Cleanup) to its LUs associated with a PU_T4|5 node and acting as PLUs.

- DACTLU or ACTLU(Cold) to its LUs associated with a PU_T1|2 node (relying on the fact these LUs have a session limit of one).

The positive response to ACTCDRM conveys the same type of information as the ACTCDRM request. If the SSCP that sent ACTCDRM does not accept the information conveyed on the positive response, it sends DACTCDRM indicating an invalid activation parameter (with reason code set to 0821, 0833, or 0835), to deactivate the session and to indicate to the sender of the response the field that was unacceptable.

DACTCDRM is sent to deactivate an SSCP-SSCP session. The type of deactivation is indicated in the request as follows:

- Normal end of session: The response to this type of DACTCDRM is accompanied by the resetting of the sessions belonging to the reset hierarchy of the SSCPs.

- Invalid activation parameter: The results are the same as for normal end of session.

- Session outage notification: DACTCDRM(type = SON) resets the SSCP-SSCP' session, and also results in the resynchronization of the two SSCPs with respect to LU-LU sessions and requests for LU-LU sessions between the two domains. The SSCP.SVC_MGR (Chapter 8) performs the SSCP-SSCP resynchronization.

Cleanup can be indicated in the DACTCDRM request, as an SON cause (see Appendix E), when an SSCP is resetting its SSCP-SSCP half-session before receiving the partner SSCP's response to DACTCDRM. In this case, CSC_MGR.SEND in the sending node, generates the response to DACTCDRM and sends the response to CSC_MGR.RCV in the same node; CSC_MGR.RCV forwards the response to the SSCP.SVC_MGR that sent DACTCDRM(Cleanup), resetting its SSCP-SSCP half-session. CSC_MGR.SEND also forwards the DACTCDRM to path control (Chapter 3).

ACTIVATE PHYSICAL UNIT (ACTPU)
DEACTIVATE PHYSICAL UNIT (DACTPU)

Flow: From SSCP to PU (Expedited)

Principal FSMs:

ACTPU is sent from an SSCP to a PU to activate a session between the SSCP and the PU, and to obtain certain information about the PU (such as contents ID). The SSCP assumes the role of primary NAU, while the PU assumes the role of secondary.

The type of session activation requested may be either Cold or ERP (error recovery procedure) where Cold would reset the SSCP-PU primary and secondary session subtrees, while ERP is requesting the activation of the SSCP-PU session without resetting the SSCP-PU primary and secondary subtrees. The type of session activation actually performed by the PU is indicated on the response. If Cold is requested, only Cold session activation is allowed. If ERP is requested, either ERP or Cold session activation may be performed, depending upon the functional capabilities of the receiving PU. The parameters and rules to be used while the SSCP-PU session is active are indicated by the FM profile and TS profile (see Appendix F).

ACTPU contains a six-byte SSCP ID field, which can be used by the receiving PU in determining the validity of the ACTPU.

The Activation Request/Response Sequence Identifier control vector and the SSCP-PU Session Capabilities control vectors are carried by format 3 of ACTPU, while the corresponding response carries the Activation Request/Response Sequence Identifier control vector and may also carry the Vector Keys Not Recognized control vector.

The Activation Request/Response Sequence Identifier control vector for ACTPU is generated by the (SSCP|PUCP).SVC_MGR. The Activation Request/Response Sequence Identifier control vector is used by the receiver of ACTPU to determine whether the current ACTPU supersedes a previously received ACTPU from the same sender. If the activation request/response sequence identifier in the SCB is less than the one in the ACTPU, the received ACTPU is more recent and the session is to be overridden. For ACTPU, the session is overridden by the PU.SVC_MGR performing lost control point hierarchical reset for the SSCP-PU session (see Chapter 11).

The SSCP-PU Session Capabilities control vector identifies the functional level of the SSCP to the PU (see CSC_MGR.SON and Appendix E for details). The Activation Request/Response Sequence Identifier control vector for ACTPU is echoed in the RSP(ACTPU) by CSC manager in the node receiving the ACTPU.

When an ACTPU is addressed to a PU_T4|5, and the Cold response is returned to the SSCP, all boundary function FSM session subtrees involving that SSCP are reset. When an ACTPU is addressed to a PU_T1|2 node supported by a boundary function, the boundary function also processes the ACTPU and the response to ACTPU, to update its control information. ACTPU and DACTPU are handled completely by the boundary function support for PU_T1s.

The activation of the SSCP-PU session on the part of the PU is signaled by a positive response.

ACTPU(Cold) is used to activate the SSCP-PU session and to reset the SSCP-PU primary and secondary session subtrees.

Cold is returned on the response to ACTPU if (1) Cold was requested, or (2) ERP was requested, but the PU could not activate the SSCP-PU session without resetting the SSCP-PU secondary session subtree. When the Cold response is returned to the SSCP by the PU, all FSMs in the reset hierarchy of the PU are reset. Upon receipt of the Cold response, all FSMs in the reset hierarchy of the SSCP-PU session.

When an ACTPU (Cold or ERP) is addressed to a PU_T2 supported by a boundary function and a positive Cold response is returned, CSC manager in the node providing boundary function support performs the necessary session outage notification as a result of resetting the SSCP-PU session subtree. If the PU returns a positive ERP response, the boundary function establishes the session parameters retained from the ACTPU being responded to without .esetting the subtree.

ACTPU(ERP) is used to activate or to resynchronize the SSCP-PU session without affecting other sessions. An ERP response indicates that the SSCP-PU data traffic and PU services subtrees were reset without affecting other half-sessions.

If the PU returns a positive ERP response, the boundary function establishes the session parameters retained from the ACTPU being responded to without resetting the subtree.

DACTPU is sent to deactivate the session between the SSCP and the PU. The type of deactivation is indicated in the request as follows:

- Final use, physical connection may be broken: The response to this type of DACTPU, in a PU_T1|2 node, is accompanied by the resetting of the FSMs in the reset hierarchy of the PU and SSCP. The response to this type of DACTPU in a PU_T4|5 node does not reset the FSMs.

- Not final use, physical connection should not be broken: This is the same process as for final use.

- Session outage notification: The cause specified for session outage notification is contained in DACTPU. All of these causes reset only the subject session, i.e., the SSCP-PU session; the hierarchical reset is not performed.

Cleanup can be indicated in the DACTPU request, as an SON cause (see Appendix E), when an SSCP is resetting its SSCP-PU half-session before receiving the PU's response to DACTPU. In this case, CSC_MGR.SEND, in the node containing the SSCP, generates a response to the DACTPU, and sends the response to CSC_MGR.RCV located in the same SSCP's node; CSC_MGR.RCV forwards the response to the SSCP.SVC_MGR, resetting the SSCP-PU primary half-session. CSC_MGR.SEND in the node containing the SSCP also forwards the DACTPU to path control (Chapter 3).

When DACTPU(Cleanup) is addressed to a PU_T1|2 node supported by a boundary function, CSC_MGR.RCV in that node generates a response to DACTPU, and sends the response to itself. The receipt of the response resets the boundary function supporting the secondary half-session. CSC_MGR.RCV also forwards the DACTPU to path control (Chapter 3).

When DACTPU is addressed to a PU_T1|2 node supported by a boundary function, CSC manager in the node providing boundary function support also performs session outage notification depending upon the type of the DACTPU.

ACTIVATE LOGICAL UNIT (ACTLU)
DEACTIVATE LOGICAL UNIT (DACTLU)

Flow: From SSCP to LU (Expedited)

Principal FSMs: FSM_SESS_CP_LU_PRI (Page 13-93)
FSM_SESS_CP_LU_SEC (Page 13-93)
FSM_SESS_BF_CP_LU (Page 13-97)

ACTLU is sent from an SSCP to an LU to activate a session between the SSCP and the LU, and to establish common session parameters. The SSCP assumes the role of primary NAU, while the LU assumes the role of secondary.

The type of session activation requested may be either Cold or ERP (error recovery procedure). The type of session activation actually performed by the LU is indicated on the response. If Cold is requested, only Cold session activation is allowed. If ERP is requested, either ERP or Cold session activation may be performed.

The parameters and rules to be used while the SSCP-LU session is active are indicated by the FM profile and TS profile (see Appendix F).

For ACTLU to be validly received, the PU providing local support for the LU must have an active half-session with the SSCP that sent the ACTLU.

The ACTLU response carries the SSCP-LU Session Capabilities control vector and the LU-LU Session Services Capabilities control vector that include fields that specify the capabilities of the LU, e.g., the maximum RU size allowed on the normal flows, the ability of the LU to accept unsolicited FMD requests from the SSCP, and the capability of the LU to act as a secondary for an LU-LU session (see Appendix E for details). The vectors also provide SSCP-LU (and BF) resynchronization capability by specifying the LU-LU session limit and the LU-LU session count.

When an ACTLU or DACTLU is addressed to an LU supported by a boundary function, the boundary function also processes the request, and the response, to update its control information for the LU.

The activation of the SSCP-LU session on the part of the LU is signaled by a positive response.

ACTLU(Cold) is used to activate the SSCP-LU session and to reset the SSCP-LU primary and secondary session subtrees.

Cold is returned on the response to ACTLU if (1) Cold was requested, or (2) ERP was requested, but the LU could not activate the SSCP-LU session without resetting the SSCP-LU secondary session subtree. When the Cold response is returned to the SSCP by the LU, all FSMs in the reset hierarchy of the secondary SSCP-LU session are reset; when the Cold response is received, all FSMs in the reset hierarchy of the primary SSCP-LU session are reset.

When an ACTLU (Cold or ERP) is addressed to an LU that is associated with a PU_T1|2 node and a positive Cold response is returned, CSC manager in the node providing boundary function support performs the necessary session outage notification as a result of resetting the SSCP-LU session subtree.

ACTLU(ERP) is used to activate or to resynchronize the SSCP-LU session without affecting other sessions. An ERP response indicates that the SSCP-LU data traffic FSMs and LU services are reset; other FSMs are unaffected.

If the LU returns a positive ERP response, the boundary function establishes the session parameters retained from the ACTLU being responded to without resetting the subtree.

DACTLU is sent to deactivate the session between the SSCP and the LU. The type of deactivation is indicated in the request as follows:

- Normal deactivation: The response to this type of DACTLU is accompanied by the resetting of the FSMs in the reset hierarchy of the SSCP-LU half-sessions.

- Session outage notification: The cause specified for session outage notification is contained in DACTLU. Only the subject SSCP-LU session is reset; the hierarchy reset is not performed.)

Cleanup can be indicated in the DACTLU request, as an SDN cause (see Appendix E), when the SSCP is resetting its SSCP-LU half-session before receiving the LU's response to DACTLU. In this case, CSC_MGR.SEND, in the node containing the SSCP, generates the response to the DACTLU and sends the response to CSC_MGR.RCV located in the same SSCP's node; CSC_MGR.RCV forwards the response to the SSCP.SVC_MGR resetting the SSCP-LU primary half-session. CSC_MGR.SEND also forwards the DACTLU to path control (Chapter 3).

When DACTLU(Cleanup) is addressed to a PU_T1|2 node supported by a boundary function, CSC_MGR.RCV in that node generates a response to DACTLU, and sends the response to itself. The receipt of the response resets the boundary function supporting the secondary half-session. CSC_MGR.RCV also forwards the DACTLU to path control (Chapter 3).

When DACTLU is addressed to an LU that is associated with a PU_T1|2 node, CSC manager in the node providing boundary function support also performs session outage notification depending upon the type of the DACTLU.

BIND SESSION (BIND)
UNBIND SESSION (UNBIND)

Flow: From primary LU to secondary LU (Expedited)

Principal FSMs: FSM_SESS_LU_LU_PRI (Page 13-94)
FSM_SESS_LU_LU_SEC (Page 13-94)
FSM_SESS_BF_LU_LU (Page 13-98)

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. Control information in either LU is updated only on a positive response. A successful BIND causes reset of the reset hierarchy of the PLU, SLU, and BF.LU.

Two types of BIND are defined: nonnegotiable and negotiable.

BIND does not have ERP types as do other session activation requests (e.g., ACTPU). The distinction between simple activation and resynchronizing reactivation following a failure is made after the session has been activated. In some cases (e.g., when the sync point protocol is used), STSN is used; in others, end user protocols are invoked.

For the nonnegotiable BIND, the secondary LU receiver of BIND checks the session parameters, which are specified by the FM, TS, and PS Profile and Usage fields (discussed below). If they are unacceptable, it returns a negative response with the sense code, Invalid Parameter (0821, 0832, 0833, or 0835). If the information carried on the BIND is otherwise acceptable (e.g., session limit not exceeded), a positive response is returned and the session parameters specified by the BIND are used for this activation of the session.

For the negotiable BIND, the receiver does not reject the BIND because of any incompatibility (if it supports negotiable BIND) with the proposed session parameters (except secondary send maximum RU size and secondary receive pacing count). Rather, if the BIND is otherwise acceptable (e.g., session limit not exceeded), a positive response is returned that carries a complete set of session parameters; these parameters can either match the primary LU's session parameters, or can differ, where the secondary chooses different options. The secondary may freely modify the session parameters, except for pacing parameters and maximum RU sizes (see SNA LU-LU Session Types). The maximum RU sizes may be reduced and the secondary CPMGR receive pacing count may be reduced, but not to zero; if the secondary CPMGR receive pacing count is reduced and the staging indicator is for one stage, the primary CPMGR send pacing

count is set equal to the secondary CPMGR receive pacing count (refer to Figure 13-8, BIND Image and BIND RU Modification Table). The primary LU receiver of the response checks the parameters as received, and sends UNBIND if they are not acceptable. If they are acceptable, then these parameters are used for the activated session.

When a BIND or UNBIND is sent to an LU supported by a boundary function, the boundary function also processes the request, and the response, to update its control information for the session. As part of the BF support processing, the BF.LU.SVC_MGR notifies the SSCP.SVC_MGR via SESSEND when the LU-LU session that it is supporting goes reset, via UNBIND, after having achieved ACTIVE state (i.e., positive response to BIND had passed through the BF.LU.SVC_MGR). If the BIND request is nonnegotiable, the BF may reduce the secondary CPMGR send pacing count if the staging indicator is set for two-stage pacing. If the BIND request is negotiable, the BF may reduce the maximum RU sizes, alter the TS profile, and reduce the primary CPMGR send pacing count and the secondary CPMGR send pacing count if the staging indicator is set for two-stage pacing (refer to Figure 13-8 for additional rules on BIND parameter modifications).

If a BIND is sent to a peripheral LU and a positive response is returned, the boundary function resets all boundary function FSMs in its reset hierarchy and establishes the session parameters retained from the BIND, for a nonnegotiable BIND, or carried on the positive response, for a negotiable BIND. If the negotiable BIND response is not acceptable to the BF.LU.SVC_MGR, or the secondary LU responds with a nonnegotiable response to a negotiable request and the boundary function had changed the maximum RU sizes, TS profile, or primary CPMGR send pacing count, the +RSP(BIND) is turned into a -RSP(BIND, sense code: 084D Invalid Session Parameters--BF) by the BF.LU.SVC_MGR and is sent to the PLU; the PLU generates and sends an UNBIND to the SLU upon receipt of an 084D sense code. If the parameters in the +RSP(BIND) are not acceptable to CSC manager in the primary half-session, CSC manager turns the +RSP(BIND) into a -RSP(BIND, sense code: 084E Invalid Session Parameters--PRI) and sends the response to the PLU. The PLU generates and sends an UNBIND to the SLU upon receipt of an 084E sense code.

A general description of the BIND RU fields follows (see Appendix E for details):

Format: This specifies the format of the BIND RU. One format is defined: Format 0; others are reserved.

Type: Two types of BIND are defined: nonnegotiable and negotiable. If the secondary does not support the type specified, it may return a negative response with either the Function Not Supported (1003) or the Invalid Session Parameter (0821, 0832, 0833, 0835) sense code, or, if the secondary does not support negotiable BIND, it may process the request as a nonnegotiable BIND. (This means that the response is positive or negative in accordance with the acceptability of the BIND parameters to the secondary; the returned response is nonnegotiable.) For the negotiable BIND, the returned positive response has the same general format as the BIND request. For the nonnegotiable BIND, the returned positive response RU can be the one-byte request code or if session-level cryptography is specified in the BIND request, the extended response is returned, consisting of at least 36 bytes. For additional details, see Appendix E.

FM Profile: This field contains a binary key that specifies some of the data flow control and function management protocols to be used by the LUs in this session. The FM Profile field contains an assigned profile number that specifies a particular set of mandatory and optional protocol rules. For those profiles with rules having options, the FM Usage field (see below) specifies which options have been selected. For additional details, see Appendix F.

TS Profile: The TS profile specifies which transmission control facilities will be used for the duration that the session remains active. The information specified by the TS profile may be supplemented by that in the TS Usage field. Certain TS profiles do not require the use of the TS Usage field. For additional details, see Appendix F.

FM Usage: This field supplements the information specified by the FM Profile. It is divided into three subfields: a common field, a secondary LU field, and a primary LU field. The common field contains those protocol rules that the primary and secondary LUs must jointly enforce (e.g., whether the normal requests will flow one direction at a time (HDX), or may flow in both directions concurrently (FDX)). The secondary LU field specifies the rules that the secondary will follow (e.g., whether the secondary may end a bracket). The secondary LU may refrain from using all the freedom the rules allow (e.g., single-RU chains may be sent even though chains with multiple RUs are allowed). Similarly, the primary LU field specifies the rules that the primary will follow for the session.

TS Usage: This field supplements the information specified by the TS profile. It is used to specify pacing parameters and maximum RU sizes on the normal flow.

PS Profile: This field contains a format indicator and an LU-LU session type designation, which together determine the format and meaning of the following PS Usage field. If LU-LU session type 0 is designated, the format and meaning of the PS Usage field are implementation defined. If a nonzero LU-LU session type,j(j>0), is designated, then the format and meaning of the PS Usage field are architecturally defined according to the format indicator and the value (j) specified for the LU-LU session type.

LU Session Type: a nonzero LU-LU session type, unlike LU-LU session type 0, architecturally determines the following for the session:

- The mandatory and optional values allowed in the FM Profile field, TS Profile field, FM Usage field, and TS Usage field of BIND
- The usage of SNA character string (SCS) controls, FM headers, RU parameters (e.g., status codes allowed in LUSTAT), and sense codes
- Presentation services protocols, such as those associated with FM header usage (see SNA LU-LU Session Types for details)

PS Usage: This field supplements the information specified by the TS and FM Profile/Usage fields by identifying additional function management options that will be used by the primary and secondary half-sessions. There are three subfields: a common field, a secondary LU field, and a primary LU field. These subfields are used in the same manner as the subfields of the same name in the FM Usage field.

Cryptography Options: This field specifies whether cryptography is used, and, if so, specifies the cryptography options and parameters to be used for the session. This field includes a count specifying the length, in bytes, of the following variable-length subfield that gives the cryptography options and parameters for the session If the count specifies 0, then cryptography is not used for the session. The PLU sets the count field to 0 and omits the following variable-length subfield if session-level cryptography has been specified (only these options require that the session cryptography keys be distributed). The PLU sets the cryptography option flags in byte 26 according to the highest order cryptography requirement, as determined by the indicators received in the BIND image of the preceding CINIT and the implementation- and installation-determined requirements for the PLU. The order is: session-level mandatory, session-level selective, none. The SLU saves the session cryptography key received in BIND and checks that the cryptography option specified is not of lower order than

it requires. For session-level cryptography, the SLU returns an enciphered session-seed value in +RSP(BIND) that is used both as a test value in CRV (see the section on that request) and as a seed value in the enciphering and deciphering processes (see "Sessions with Cryptography" in Chapter 4 for additional discussion and references).

Primary LU Name: An uninterpreted name, as carried in the INIT request from the SLU, or otherwise a network name, preceded by a one-byte binary length indication. The length value does not include the Count field itself. The PLU name in the BIND RU identifies the PLU to the SLU. The PLU name may also be carried on the INIT and CINIT requests; these requests flow to the SSCP from the initiating LU and from the SSCP to the PLU, respectively, and precede the sending of BIND (see Chapter 8 and Appendix E). The SSCP assigns the proper PLU network address using the PLU name.

User Data: Contains data defined by the LU services managers of the session or by their end users. It is not used by the CSC_MGR(s), nor is it used by the SSCP(s) that built the BIND image in CINIT. It is preceded by a one-byte binary Length field. The length value does not include the Length field itself. Two formats are defined, based upon the value of the first byte:

- -X'00': the entire User Data field is unstructured and can be used for implementation defined purposes.

- X'00': the User Data field contains one or more architected structured subfields. Each subfield is preceded by 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.

The following subfields are defined:

Unstructured Data: Subfield number X'00' contains unstructured data. It can be used for implementation defined purposes.

Session Qualifier: Subfield number X'01' contains data that associates the session with resources within the two LUs. It can be used to associate the session with resynchronization data so that a session that failed can be reactivated without loss of data. See the discussion of the STSN request, Chapter 4. It also can be used to associate the session with a statically defined resource such as a queue of output messages. Two subfields are used:

- Primary Resource Qualifier: This field, changeable on the (negotiable) response, defines the associated resources within the primary LU. It consists of a one-byte binary Length field followed by the qualifier data. The length does not include the Length field itself.

- Secondary Resource Qualifier: This field, changeable on the (negotiable) response, defines the associated resources within the secondary LU. It consists of a one-byte binary Length field followed by the qualifier data. The length does not include the length field itself.

The session resource qualifiers supply a unique name for each session even when parallel sessions are in use:

- Session Qualifier Pair = (primary resource qualifier, secondary resource qualifier)

- Session Name (SN) = (PLUname, SLUname). session_qualifier_pair

This allows half-sessions to be named independently of their network addresses (the naming implies that sessions are reactivated restart with the primary/secondary polarity of the original session).

The User Data field in BIND is constructed from both User Data fields (inside and outside the BIND image) in CINIT.

User Request Correlation Field: Contains the user request correlation (URC) value, as extracted from the CINIT RU by the primary LU services manager. This allows the SLU to relate the INIT request and the BIND. It is preceded by a one-byte binary Length field. The length value does not include the Length field itself. The Length field is nonzero only if the SLU generated the URC, originally, in an INIT that resulted in the sending of the BIND.

Secondary LU Name: Contains the secondary LU network name (an EBCDIC symbolic name), as extracted from the CINIT RU by the primary LU services manager. It is preceded by a one-byte binary Length field. The length value does not include the Length field itself. The Length field is nonzero if and only if BIND is negotiable.

UNBIND is sent to deactivate an active session between the two LUs. The positive response to UNBIND is accompanied by the deletion of the SCBs used by the session.

If UNBIND is addressed to an LU associated with a peripheral node, CSC manager in the node providing boundary function support resets all boundary function FSMs in its reset hierarchy when UNBIND or its response (whether positive or negative) is processed. The type of deactivation (e.g., normal end of session, virtual route inoperative (see Appendix E)) is specified in the UNBIND request.

Cleanup can be indicated in the UNBIND request, as an SON cause (see Appendix E), when an LU is resetting its LU-LU half-session before receiving the partner LU's response to UNBIND. In this case, CSC_MGR.SEND in the sending node generates the response to the UNBIND and sends the response to CSC_MGR.RCV in that node; CSC_MGR.RCV forwards the response to the LU.SVC_MGR, resetting its LU-LU half-session. CSC_MGR.SEND also forwards the UNBIND to path control (Chapter 3).

When UNBIND(Cleanup) is addressed to a peripheral node supported by a boundary function, CSC_MGR.RCV generates a response to UNBIND and sends the response to itself. The receipt of the response resets the boundary function supporting the LU-LU half-session. CSC_MGR.RCV also forwards the UNBIND to path control (Chapter 3).

	CINIT for same-domain session SSCP(PLU) or CDCINIT SSCP(SLU) (Note 1)	CINIT for cross-domain session SSCP(PLU) (Note 2)	BIND (nonnegotiable) PLU (Note 3)	BF (Note 4)	SLU (Note 5)	BIND (negotiable) PLU (Note 3)	BF (Note 4)	SLU (Note 5)
Type	I	A	A	-	-	A	-	A
Format	I	-	-	-	-	-	-	-
FM profile	I	A	A	-	-	A	-	A
TS profile	I	A	A	-	-	A	A	A
FM usage	I	A	A	-	-	A	-	A
TS usage (Note 6)								
staging indicator for secondary-to-primary pacing	I	-	-	-	-	-	-	-
secondary CPMGR send pacing count	I	D3	D3	D2	-	D3	D2	-
secondary CPMGR receive pacing count	I	-	-	-	-	-	-	D
secondary-to-primary maximum RU size	I	D	D	-	-	D	D	D
primary-to-secondary maximum RU size	I	D	D	-	-	D	D	D
primary CPMGR send pacing count	I	-	-	-	-	-	D2	D1
staging indicator for primary-to-secondary pacing	I	-	-	-	-	-	-	-
primary CPMGR receive pacing count	I	D	D	-	-	D	-	-
PS profile	I	A	A	-	-	A	-	A
PS usage	I	A	A	-	-	A	-	A
Cryptography	I	C	C	-	C	C	-	C
Primary LU name	O	O	-	-	EO	-	-	EO
User data	I	A	A	-	EO	A	-	A
User request correlation	O1	P1	P2	-	EO	P2	-	EO
Secondary LU name	I1	I1	-	-	/	-	-	EO

/ not included in nonnegotiable BIND
- not allowed to change
A allowed to change
C change only as allowed for cryptography
D allowed to decrease
D1 sets equal to (if greater than) the value of secondary CPMGR receive pacing count for one-stage pacing
D2 allowed to decrease for two-stage pacing
D3 sets equal to (if not already equal) the value of primary CPMGR receive pacing count for one-stage pacing
I values initially assigned based on optional implementation and installation parameters for the specific LU
I1 is the network name; valid for negotiable BIND only
O uninterpreted name used if SLU issued INIT; network name otherwise; uninterpreted name obtained from INIT RU for same domain, CDINIT RU for cross-dcmain
O1 obtained from INIT RU
P1 not allowed to change if SLU issued INIT; included if PLU or SLU issued INIT; otherwise, not present
P2 included if SLU issued INIT; otherwise, not included
EO may be echoed or omitted by setting the length to zero

NOTES:

1. SSCP(SLU) assigns initial values in "BIND image" (see CDCINIT, Chapter 8) SSCP(PLU) assigns initial values in "BIND image" (see CINIT, Chapter 8)

2. SSCP(PLU) can change the value on CDCINIT to CINIT conversion (see CDCINIT, Chapter 8)

3. PLU can change the value on CINIT to BIND conversion (see CINIT, Chapter 8)

4. BF can change the value on BIND request (see BIND, this chapter)

5. SLU can change the value on BIND response (see BIND, this chapter)

6. Changing from an unspecified value to a specified value is considered to be a decrease.

Figure 13-8. BIND Image and BIND RU Modification Table

```
CSC_MGR.SEND: PROCEDURE;

                                                                        /*
 ┌──────────────────────────────────────────────────────────────────────┐
 │     FUNCTION:  THIS  PROCEDURE RECEIVES  THE REQUESTS  FOR  SESSION ACTIVATION  AND  │
 │                DEACTIVATION FROM  THE NAU.SVC_MGR,  FROM CSC_MGR.SON,  OR FROM  THE  │
 │                VR_MGR TO  BE SENT TO PATH  CONTROL (CHAPTER 3).  THE  RESPONSES FOR  │
 │                SESSION   ACTIVATION  AND   DEACTIVATION  ARE   RECEIVED  FROM   THE  │
 │                NAU.SVC_MGR  OR FROM  THE VR_MGR.   THIS PROCEDURE  OBTAINS THE  SCB  │
 │                POINTER (EITHER PRESENT OR NULL), SETS  THE  CB_TYPE TO HALF_SESS OR  │
 │                BF_SESS, AND SETS THE POINTER TO THE NODE RESOURCE ENTRY.             │
 │                                                                                      │
 │     INPUT:     ACTIVATION OR DEACTIVATION REQUESTS  OR RESPONSES; MUCB.DIRECTION IS  │
 │                SET                                                                   │
 │                                                                                      │
 │     OUTPUT:    THE REQUEST  OR RESPONSE IS  ROUTED TO THE  APPROPRIATE CSC_MGR.SEND  │
 │                ROUTINE; NRCB_PTR, CB_TYPE, AND SCB_PTR ARE SET.                      │
 └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF DISPATCHED_BY(BF.LU.SVC_MGR) | DISPATCHED_BY(BF.PU.SVC_MGR) THEN
    CB_TYPE = BF_SESS;                                                  /* PAGE 13-99   */
  ELSE
    CB_TYPE = HALF_SESS;                                                /* PAGE 13-99   */
  SELECT ANYORDER(NCB.PU_TYPE);
  . WHEN(T1)
  .   DO;
  .   . NRCB_PTR = LOCATE_NODE_RESOURCE(B'0000000000'||LSID(2:7));  /* APPENDIX B   */
  .   . FIND SCB IN SCB_LIST
  .   .       WHERE(SCB.LOCAL_SESSION_ID = LSID);
  .   . CALL CSC_MGR.T1_OR_T2_SEND;                                    /* PAGE 13-37   */
  .   END;
  . WHEN(T2)
  .   DO;
  .   . NRCB_PTR = LOCATE_NODE_RESOURCE(X'00'||OAFPRIME);             /* APPENDIX B   */
  .   . FIND SCB IN SCB_LIST
  .   .       WHERE(SCB.PARTNER_ID = DAFPRIME &
  .   .       SCB.THIS_ID = OAFPRIME);
  .   . CALL CSC_MGR.T1_OR_T2_SEND;                                    /* PAGE 13-37   */
  .   END;
  . WHEN(T4 | T5)
  .   DO;
  .   . NRCB_PTR = LOCATE_NODE_RESOURCE(OEF);                          /* APPENDIX B   */
  .   . FIND SCB IN SCB_LIST
  .   .       WHERE(SCB.PARTNER_SA = DSAF &
  .   .       SCB.PARTNER_EA = DEF &
  .   .       SCB.THIS_SA = OSAF &
  .   .       SCB.THIS_EA = OEF);
  .   . CALL CSC_MGR.T4_OR_T5_SEND;                                    /* PAGE 13-38   */
  .   END;
  END;
  RETURN;
END CSC_MGR.SEND;
```

This page intentionally left blank

```
CSC_MGR.T1_OR_T2_SEND: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS  PROCEDURE RECEIVES  THE  REQUESTS  AND RESPONSES  FOR  SESSION │
  │                 DEACTIVATIONAND RESPONSES  FOR SESSION ACTIVATION  FROM CSC_MGR.SEND │
  │                 (PAGE 13-35).  THE  REQUEST OR RESPONSE IS  VERIFIED, E.G., CHECKING │
  │                 THAT  THE REQUEST  OR RESPONSE  CAN  FLOW ON  THE SESSION,  CHECKING │
  │                 PARAMETERS WITHIN THE  RU, (SUCH AS, FM AND TS  PROFILES), AND DOING │
  │                 THE  STATE  SEND CHECKS.   IF  THE  REQUEST  OR RESPONSE  IS  VALID, │
  │                 PARAMETERS ARE RETAINED FOR THE ACTIVATION OF THE SESSION AND THE RU │
  │                 IS SENT TO PATH  CONTROL; OTHERWISE, A REJECT SIGNAL IS  SENT TO THE │
  │                 SENDING  PROCEDURE.    FOR  DEACTIVATION   RESPONSES,  THE   SCB  IS │
  │                 DISCARDED.                                                   │
  │                                                                              │
  │      INPUT:     ACTIVATION  RESPONSES   OR  DEACTIVATION   REQUESTS  OR   RESPONSES; │
  │                 NRCB_PTR, CB_TYPE, AND SCB_PTR ARE SET.                      │
  │                                                                              │
  │      OUTPUT:    IF ALL SEND  CHECKS ARE PASSED, THE  REQUEST OR RESPONSE IS  SENT TO │
  │                 PATH CONTROL  (CHAPTER 3).   IF THE  SEND CHECKS  ARE NOT  PASSED, A │
  │                 SEND_CHECK IS SENT TO THE SENDING NAU.SVC_MGR.               │
  │                                                                              │
  │      NOTE:      FOR ACTIVATION REQUESTS, THE SCB WAS CREATED BY CSC_MGR.T1_OR_T2_RCV │
  │                 (PAGE 13-42).  PU_T1|2'S CANNOT SEND SESSION ACTIVATION REQUESTS.  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF RRI = RQ THEN
    DO;
    . IF RQ_CHECKS = RQ_OK THEN            /* PAGE 13-48, SEE NOTE                 */
    .   DO;
    .   . CALL #FSM_SESS;                  /* PAGES 13-91 THROUGH 13-98            */
    .   . IF MU_PTR ¬= NULL THEN           /* MU COULD HAVE BEEN DISCARDED BY FSM  */
    .   .   IF NCB.PU_TYPE = T1 THEN
    .   .     SEND MU TO PC_T1.SEND;                          /* CHAPTER 3        */
    .   .   ELSE
    .   .     SEND MU TO PC_T2.SEND;                          /* CHAPTER 3        */
    .   END;
    . ELSE
    .   SEND SEND_CHECK TO SENDING_PROCEDURE;
    END;
  IF RRI = RSP THEN
    DO;
    . IF RSP_CHECKS = RSP_OK THEN                              /* PAGE 13-49       */
    .   DO;
    .   . CALL #FSM_SESS;                  /* PAGES 13-91 THROUGH 13-98            */
    .   . IF MU_PTR ¬= NULL THEN           /* MU COULD HAVE BEEN DISCARDED BY FSM  */
    .   .   IF NCB.PU_TYPE = T1 THEN
    .   .     SEND MU TO PC_T1.SEND;                          /* CHAPTER 3        */
    .   .   ELSE
    .   .     SEND MU TO PC_T2.SEND;                          /* CHAPTER 3        */
    .   END;
    . ELSE
    .   SEND SEND_CHECK TO SENDING_PROCEDURE;
    END;
  RETURN;
END CSC_MGR.T1_OR_T2_SEND;
```

```
CSC_MGR.T4_OR_T5_SEND: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS  PROCEDURE RECEIVES  ACTIVATION AND  DEACTIVATION REQUESTS  AND │
│                 RESPONSES FROM  THE VR_MGR (CHAPTER  12) AND FROM  CSC_MGR.SEND THAT │
│                 ARE SENT  BY A  PU_T4|5 NODE  (THIS EXCLUDES  THE BOUNDARY  FUNCTION │
│                 PORTION OF THE PU_T4|5 NODE).  THE  REQUEST OR RESPONSE IS VERIFIED, │
│                 E.G., CHECKING THAT THE REQUEST OR RESPONSE CAN FLOW ON THE SESSION, │
│                 CHECKING PARAMETERS WITHIN THE RU, SUCH  AS, FM AND TS PROFILES, AND │
│                 DOING STATE SEND CHECKS.                                 │
│                                                                          │
│      INPUT:     ACTIVATION OR  DEACTIVATION REQUESTS  OR RESPONSES.  ACTIVATION AND │
│                 DEACTIVATION REQUESTS CAN COME--VIA CSC_MGR.SEND--FROM A NAU.SVC_MGR │
│                 OR  FROM THE  VR_MGR.  ALL OTHER  DEACTIVATION  REQUESTS COME  FROM │
│                 NAU.SVC_MGR  OR CSC_MGR.SON  AND  ARE TO  BE  SENT  TO PATH  CONTROL │
│                 (CHAPTER 3).                                             │
│                                                                          │
│      OUTPUT:    IF ALL SEND  CHECKS ARE PASSED, THE  REQUEST OR RESPONSE IS  SENT TO │
│                 PATH  CONTROL (CHAPTER  3); IF  THE SEND  CHECKS ARE  NOT PASSED,  A │
│                 REJECT SIGNAL  IS RETURNED TO THE  SENDING NAU.SVC_MGR.  IF A  VR IS │
│                 NEEDED FOR THE SESSION, THE REQUEST IS SENT TO THE VR_MGR.       │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL SAVE_MU_PTR PTR;
SELECT ANYORDER;
                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      ACTIVATION REQUEST RECEIVED  FROM NAU.SVC_MGR AND THE SESSION  CONTROL BLOCK IS │
│      NOT PRESENT.                                                        │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
. WHEN(RRI = RQ & SCB_PTR = NULL &  ¬DISPATCHED_BY(PU.SVC_MGR.PC_ROUTE_MGR.RCV))
.   DO;
.   . IF RQ_CHECKS = RQ_OK THEN                                 /* PAGE 13-48      */
.   .   DO;
.   .   . CALL SCB_CREATE;                                      /* PAGE 13-87      */
.   .   . IF SCB_PTR = NULL THEN                                /* SCB NOT CREATED */
.   .   .   DO;
.   .   .   . SEND SEND_CHECK(X'0812') TO SENDING PROCEDURE;    /* INSUFFICIENT RESOURCE */
.   .   .   . RETURN;
.   .   .   END;
.   .   . CALL #FSM_SESS;                 /* PAGES 13-91 THROUGH 13-94 */
.   .   . SEND MU TO PU.SVC_MGR.PC_ROUTE_MGR.RCV;               /* VR_MGR IN CHAPTER 12 */
.   .   END;
.   . ELSE
.   .   SEND SEND_CHECK TO SENDING_PROCEDURE;
.   END;
                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      ACTCDRM IS RETURNED BY  THE VR_MGR AND THE SESSION CONTROL  BLOCK WAS DISCARDED │
│      WHILE THE ACTCDRM WAS IN THE VR_MGR. THIS  CAN HAPPEN WHEN THE OTHER SSCP SENDS │
│      THE ACTCDRM AND THIS SSCP SENDS A -RSP(ACTCDRM).                    │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
. WHEN(RRI = RQ & SCB_PTR = NULL &
.      DISPATCHED_BY(PU.SVC_MGR.PC_ROUTE_MGR.RCV))                  /* CHAPTER 12      */
.   SEND MU TO CSC_MGR.SEND;                                        /* PAGE 13-35      */
                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      ACTIVATION  REQUEST AND  VRCB  POINTER IS  RETURNED BY  THE  VR_MGR; A  SESSION │
│      CONTROL BLOCK IS PRESENT.  IF THE VRCB  POINTER IN THE SESSION CONTROL BLOCK IS │
│      NOT NULL  (I.E., A  VIRTUAL ROUTE  HAS BEEN  CONNECTED TO  THE SESSION  CONTROL │
│      BLOCK), THE ACTIVATION REQUEST IS RETURNED TO THE NAU.SVC_MGR WITH A -RSP.  │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
. WHEN(RRI = RQ & SCB_PTR ¬= NULL &
.      DISPATCHED_BY(PU.SVC_MGR.PC_ROUTE_MGR.RCV))                  /* CHAPTER 12      */
.   DO;
.   . IF SCB.VRCBPTR = NULL THEN          /* SESSION DOES NOT HAVE VR */
.   .   DO;
.   .   . SCB.VRCBPTR = VRCB_PTR;                                  /* CONNECT VRCB TO SCB */
.   .   . VRCB.SESS_COUNT = VRCB.SESS_COUNT + 1; /* ADD THIS SESSION TO THE NUMBER */
.   .   .                             /* OF SESSIONS USING THIS VR */
.   .   . SEND MU TO PC.VRC.SEND;                                  /* CHAPTER 3       */
.   .   END;
.   . ELSE                            /* OTHER HALF-SESSION OBTAINED A VR */
.   .   DO;
.   .   . IF VRCB_PTR->VRCB.SESS_COUNT <= 0 THEN
.   .   .   SEND 'SESS_COUNT=0' TO PU.SVC_MGR.PC_ROUTE_MGR.RCV; /* CHAPTER 12      */
.   .   . SEND_CHECK_SENSE = X'080D';                              /* NAU CONTENTION  */
.   .   . SEND SEND_CHECK TO SSCP.SVC_MGR.CS.RCV;                  /* CHAPTER 7       */
.   .   END;
.   END;
```

```
                                                                        /*
┌──────────────────────────────────────────────────────────────────────┐
│     THE ONLY VALID  REQUESTS ARE THE DEACTIVATION REQUESTS FROM  THE NAU.SVC_MGR OR
│     CSC_MGR.SON.  IF  THERE IS  NO VR FOR  THE SESSION  (SCB.VRCBPTR IS  NULL), THE
│     NAU.SVC_MGR  IS "CHASING"  AN ACTIVATION  REQUEST WITH  A DEACTIVATION  REQUEST
│     PRIOR TO THE ACTIVATION OF THE VR.
└──────────────────────────────────────────────────────────────────────┘
                                                                        */
. WHEN(RRI = RQ & SCB_PTR ¬= NULL &
.      ¬DISPATCHED_BY(PU.SVC_MGR.PC_ROUTE_MGR.RCV))          /* CHAPTER 12          */
.   DO;
.   . IF RQ_CHECKS = RQ_OK THEN                             /* PAGE 13-48          */
.   .   DO;
.   .   . CALL #FSM_SESS;              /* PAGE 13-91, 13-93, 13-92, OR 13-94       */
.   .   . IF SON_TYPE = CLEANUP THEN                        /* PAGE 13-90          */
.   .   .   DO;
.   .   .   . SAVE_MU_PTR = MU_PTR;
.   .   .   . MU_PTR = CREATE_DEACTIVATION_RSP;             /* PAGE 13-89          */
.   .   .   . SEND MU TO CSC_MGR.RCV;                       /* PAGE 13-41          */
.   .   .   . MU_PTR = SAVE_MU_PTR;
.   .   .   END;
.   .   . IF SCB.VRCBPTR = NULL THEN    /* CHASING ACTIVATION RQ WITH DEACTIVATION  */
.   .   .   SEND MU TO PU.SVC_MGR.PC_ROUTE_MGR.RCV;         /* CHAPTER 12          */
.   .   . ELSE                          /* NORMAL DEACTIVATION SEND TO PATH CONTROL */
.   .   .   DO;
.   .   .   . IF MU_PTR ¬= NULL THEN    /* MU COULD HAVE BEEN DISCARDED BY FSM      */
.   .   .   .   DO;
.   .   .   .   . VRCB_PTR = SCB.VRCBPTR;
.   .   .   .   . SEND MU TO PC.VRC.SEND;                  /* CHAPTER 3           */
.   .   .   .   END;
.   .   .   END;
.   .   END;
.   . ELSE
.   .   SEND SEND_CHECK TO SENDING_PROCEDURE;
.   END;
                                                                        /*
┌──────────────────────────────────────────────────────────────────────┐
│     RESPONSES FROM THE VR_MGR. (SEE CHAPTER 12)
│        • -RSP(ACTIVATION): VR_MGR COULD NOT OBTAIN A VIRTUAL ROUTE FOR A SESSION.
│        • +RSP(DEACTIVATION):  VR_MGR  FOUND AND REMOVED THE  ACTIVATION REQUEST IN
│          THE VR RESERVATION LIST; THIS IS A "CHASED" ACTIVATON REQUEST.
│        • -RSP(DEACTIVATION): VR_MGR  COULD NOT FIND THE ACTIVATION  REQUEST IN THE
│          VR RESERVATION LIST.  CHECK TO SEE IF THE ACTIVATION REQUEST WAS RETURNED
│          TO CSC_MGR WITH A VR.  IF SO, SEND THE CORRESPONDING DEACTIVATION REQUEST
│          TO SESSION PARTNER.
└──────────────────────────────────────────────────────────────────────┘
                                                                        */
. WHEN(RRI = RSP & DISPATCHED_BY(PU.SVC_MGR.PC_ROUTE_MGR.RCV)) /* CHAPTER 12       */
.   DO;
.   . IF RTI = NEG & RQ_CODE = (DACTPU | DACTLU | UNBIND | DACTCDRM) &
.   .      SCB.VRCBPTR ¬= NULL THEN
.   .   DO;
.   .   . CALL CREAT_DEACT_RQ(SWITCHED,RQ_CODE,CLEANUP,SEND);  /* PAGE 13-65       */
.   .   . SEND MU TO CSC_MGR.SEND;                          /* PAGE 13-35          */
.   .   END;
.   . ELSE
.   .   IF SCB_PTR ¬= NULL THEN
.   .     DO;
.   .     . IF RQ_CODE ¬= (ACTCDRM | DACTCDRM) THEN
.   .     .   REMOVE SCB FROM SCB_LIST DISCARD;
.   .     . IF RQ_CODE = (ACTCDRM | DACTCDRM) & SCB.VRCBPTR = NULL THEN
.   .     .   REMOVE SCB FROM SCB_LIST DISCARD;
.   .     . IF RQ_CODE = (BIND | UNBIND) THEN /* DETERMINE THE SVC_MGR TO RECEIVE RSP */
.   .     .   SEND MU TO LU.SVC_MGR.SS.RCV;                 /* CHAPTER 8           */
.   .     . ELSE
.   .     .   SEND MU TO SSCP.SVC_MGR.CS.RCV;               /* CHAPTER 7           */
.   .     END;
.   .   ELSE
.   .     DISCARD MU;
.   END;
                                                                        /*
┌──────────────────────────────────────────────────────────────────────┐
│     NAU.SVC_MGR IS RETURNING A RESPONSE. FOR SON CONDITIONS, THE MU IS DISCARDED BY
│     THE FSM.
└──────────────────────────────────────────────────────────────────────┘
                                                                        */
. WHEN(RRI = RSP &  ¬DISPATCHED_BY(PU.SVC_MGR.PC_ROUTE_MGR))
.   DO;
.   . IF RSP_CHECKS = RSP_OK THEN                           /* PAGE 13-49          */
.   .   DO;
.   .   . CALL #FSM_SESS;              /* PAGE 13-91, 13-93, 13-92, OR 13-94       */
.   .   . IF MU_PTR ¬= NULL THEN        /* MU COULD HAVE BEEN DISCARDED BY FSM      */
.   .   .   DO;
.   .   .   . VRCB_PTR = SCB.VRCBPTR;
.   .   .   . SEND MU TO PC.VRC.SEND;                      /* CHAPTER 3           */
.   .   .   END;
.   .   END;
.   . ELSE
.   .   SEND SEND_CHECK TO SENDING_PROCEDURE;
.   END;
. OTHERWISE
.   DO;
.   . SEND_CHECK_SENSE = X'8005';                           /* NO SESSION          */
.   . SEND SEND_CHECK TO SENDING_PROCEDURE;
.   END;
  END;
  RETURN;
END CSC_MGR.T4_OR_T5_SEND;
```

```
CSC_MGR.BF_SEND: PROCEDURE;

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS  PROCEDURE RECEIVES  THE  REQUESTS  AND RESPONSES  FOR  SESSION │
 │                 ACTIVATION AND  DEACTIVATION FROM THE  BF.(PU | LU).SVC_MGR  OR FROM │
 │                 CSC_MGR.SON THIS PROCEDURE  OBTAINS THE SCB POINTER  (EITHER PRESENT │
 │                 OR NULL), SETS THE  SCB_TYPE TO BF_SESS, AND SETS THE POINTER TO THE │
 │                 NODE  RESOURCE  ENTRY FOR  THE  BOUNDARY  FUNCTION RESOURCE  OF  THE │
 │                 ACTIVATION.                                                  │
 │                                                                              │
 │      INPUT:     ACTIVATION OR DEACTIVATION REQUESTS OR RESPONSES; MUCB.DIRECTION AND │
 │                 NRCB_PTR IS SET.                                             │
 │                                                                              │
 │      OUTPUT:    THE REQUEST  OR RESPONSE IS  ROUTED TO THE  APPROPRIATE CSC_MGR.SEND │
 │                 ROUTINE.                                                     │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  CB_TYPE = BF_SESS;
  NRCB_PTR = LOCATE_NODE_RESOURCE(DEF);                          /* APPENDIX B   */
  IF NRCB.RESOURCE_TYPE = PU_T1 THEN
    FIND SCB IN SCB_LIST
         WHERE(SCB.LOCAL_SESSION_ID = LSID);
  ELSE
    FIND SCB IN SCB_LIST
         WHERE(SCB.PARTNER_ID = OAFPRIME &
         SCB.THIS_ID = DAFPRIME);
  IF RRI = RQ THEN
    DO;
    . IF RQ_CHECKS = RQ_OK THEN                                  /* PAGE 13-48   */
    .   DO;
    .   . CALL #FSM_SESS;                 /* PAGES 13-91 THROUGH 13-98          */
    .   . IF MU_PTR ¬= NULL THEN          /* MU COULD HAVE BEEN DISCARDED BY FSM */
    .   .   IF SCB_PTR ¬= NULL THEN
    .   .     LSCB_PTR = SCB.ALS_FOR_BF;                         /* USED IN BF.PC */
    .   . SEND MU TO BF.PC;                                      /* CHAPTER 3    */
    .   END;
    . ELSE
    .   SEND SEND_CHECK TO SENDING_PROCEDURE;
    END;
  IF RRI = RSP THEN
    DO;
    . IF RSP_CHECKS = RSP_OK THEN                                /* PAGE 13-49   */
    .   DO;
    .   . CALL #FSM_SESS;                 /* PAGES 13-91 THROUGH 13-98          */
    .   . IF MU_PTR ¬= NULL THEN          /* MU COULD HAVE BEEN DISCARDED BY FSM */
    .   .   IF SCB_PTR ¬= NULL THEN
    .   .     LSCB_PTR = SCB.ALS_FOR_BF;                         /* USED IN BF.PC */
    .   . SEND MU TO BF.PC;                                      /* CHAPTER 3    */
    .   END;
    . ELSE
    .   SEND SEND_CHECK TO SENDING_PROCEDURE;
    END;
  RETURN;
END CSC_MGR.BF_SEND;
```

```
CSC_MGR.RCV: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS  PROCEDURE RECEIVES  THE  REQUESTS  AND RESPONSES  FOR  SESSION  │
   │                 ACTIVATION  AND DEACTIVATION  FROM PATH  CONTROL  (CHAPTER 3).   THE  │
   │                 REQUEST  OR  RESPONSE  IS  ROUTED  TO  THE  APPROPRIATE  CSC_MGR.RCV  │
   │                 ROUTINE.                                                 │
   │                                                                          │
   │      INPUT:     SESSION ACTIVATION OR DEACTIVATION REQUESTS OR RESPONSES  │
   │                                                                          │
   │      OUTPUT:    RU IS ROUTED TO THE APPROPRIATE CSC_MGR.RCV ROUTINE      │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF NCB.PU_TYPE = (T1 | T2) THEN
    CALL CSC.T1_OR_T2_RCV;                                  /* PAGE 13-42    */
  ELSE
    IF NCB.PU_TYPE = (T4 | T5) THEN
      CALL CSC.T4_OR_T5_RCV;                                /* PAGE 13-45    */
END CSC_MGR.RCV;
```

```
CSC_MGR.T1_OR_T2_RCV: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS  PROCEDURE RECEIVES  THE  REQUESTS  AND RESPONSES  FOR  SESSION  │
│                 ACTIVATION  AND DEACTIVATION  FROM  CSC_MGR.RCV  (PAGE 13-41).  THE  │
│                 REQUEST OR RESPONSE IS VERIFIED (E.G., CHECKING THAT THE REQUEST OR  │
│                 RESPONSE CAN FLOW ON THE SESSION, CHECKING PARAMETERS WITHIN THE RU, │
│                 (E.G., FM AND  TS PROFILES), AND MAKING THE  STATE  RECEIVE CHECKS).  │
│                 THE SESSION CONTROL  BLOCK IS CREATED, IF  NECESSARY, AND PARAMETERS  │
│                 ARE RETAINED FOR THE ACTIVATION OF THE SESSION.  IF THE VERIFICATION  │
│                 OF THE RU IS  SUCCESSFUL, THE RU IS SENT TO  THE NAU.SVC_MGR; IF THE  │
│                 VERIFICATION IS NOT SUCCESSFUL, THE REQUEST IS CHANGED TO A -RSP AND  │
│                 SENT BACK TO PATH CONTROL (CHAPTER 3)                                 │
│                                                                                       │
│      INPUT:     ACTIVATION OR DEACTIVATION REQUESTS OR RESPONSES                       │
│                                                                                       │
│      OUTPUT:    THE   REQUEST    OR   RESPONSE   IS    SENT   TO    NAU.SVC_MGR;   A  │
│                 CSC_MGR.RCV-GENERATED -RSP IS SENT TO PATH CONTROL (CHAPTER 3).       │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL SAVE_MU_PTR PTR;
                                                                        /*
                  ┌──────────────────────────────────────────────┐
                  │ INITIALIZE PARAMETERS; MUCB.DIRECTION ALREADY │
                  │ SET                                           │
                  └──────────────────────────────────────────────┘
                                                                        */
  CB_TYPE = HALF - SESS;
  IF NCB.PU_TYPE = PU_T1 THEN
    DO;
    . NRCB_PTR = LOCATE_NODE_RESOURCE(B'00000000000'||LSID(2:7)); /* APPENDIX B          */
    . IF NRCB_PTR ¬= NULL THEN
    .   FIND SCB IN SCB_LIST
    .        WHERE(SCB.LOCAL_SESSION_ID = LSID);
    END;
  ELSE
    DO;
    . NRCB_PTR = LOCATE_NODE_RESOURCE(X'00'||DAFPRIME);          /* APPENDIX B          */
    . IF NRCB_PTR ¬= NULL THEN
    .   FIND SCB IN SCB_LIST
    .        WHERE(SCB.PARTNER_ID = OAFPRIME &
    .        SCB.THIS_ID = DAFPRIME);
    END;
  IF NRCB_PTR = NULL |
       NRCB.RESOURCE_CATEGORY ¬= (PU | LU) THEN
    DO;
    . IF RRI = RSP THEN
    .   DISCARD MU;
    . ELSE                                                       /* REQUEST             */
    .   DO;
    .   . RECEIVE_CHECK_SENSE = X'8004';   /* UNRECOGNIZED DESTINATION ADDRESS          */
    .   . CALL CHANGE_MU_TO_NEG_RSP(RECEIVE_CHECK_SENSE);        /* APPENDIX B          */
    .   . IF NCB.PU_TYPE = T1 THEN
    .   .   SEND MU TO PC_T1.SEND;                               /* CHAPTER 3           */
    .   . ELSE
    .   .   SEND MU TO PC_T2.SEND;                               /* CHAPTER 3           */
    .   END;
    . RETURN;
    END;
                                                                        /*
                  ┌──────────────────────────────────────────────┐
                  │ CB_TYPE,   SCB_PTR,   AND   NRCB_PTR   NOW   │
                  │ INITIALIZED                                   │
                  └──────────────────────────────────────────────┘
                                                                        */
  SELECT ANYORDER;
  . WHEN(RRI = RQ)
  .   SELECT ANYORDER(RQ_CHECKS);                                /* PAGE 13-48          */
  .   . WHEN(RQ_OK)
  .   .   DO;
  .   .   . IF SCB_PTR = NULL THEN
  .   .   .   DO;
  .   .   .   . CALL SCB_CREATE;                                 /* PAGE 13-87          */
  .   .   .   . IF SCB_PTR = NULL THEN                          /* SCB NOT CREATED     */
  .   .   .   .   DO;
  .   .   .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'0812'); /* APPENDIX B, INSUFFICIENT RESOURCE */
  .   .   .   .   . IF NCB.PU_TYPE = T1 THEN
  .   .   .   .   .   SEND MU TO PC_T1.SEND;                     /* CHAPTER 3           */
  .   .   .   .   . ELSE
  .   .   .   .   .   SEND MU TO PC_T2.SEND;                     /* CHAPTER 3           */
  .   .   .   .   END;
  .   .   .   END;
  .   .   . CALL #FSM_SESS;              /* PAGES 13-91 THROUGH 13-98                   */
  .   .   . SEND MU TO #SVC_MGR;
  .   .   END;
```

```
.    . WHEN(RQ_NG)
.      DO;
.      . CALL CHANGE_MU_TO_NEG_RSP(RECEIVE_CHECK_SENSE);      /* APPENDIX B          */
.      . IF NCB.PU_TYPE = T1 THEN
.      .    SEND MU TO PC_T1.SEND                            /* CHAPTER 3           */
.      .         ELSE
.      .         SEND MU TO PC_T2.SEND;                      /* CHAPTER 3           */
.      . END;
.    END;
.  WHEN(RRI = RSP)
.                                        /* RSP(UNBIND) IS THE ONLY VALID RESPONSE  */
.    IF RSP_CHECKS = RSP_OK THEN                             /* PAGE 13-49          */
       DO;
.      . CALL #FSM_SESS;                 /* PAGES 13-91 THROUGH 13-98               */
.      . SEND MU TO #SVC_MGR;
.      . CALL SCB_DISCARD;               /*  SCB WAS NOT DISCARDED BY FSM IN THIS CASE */
.      END;
.    ELSE
.      DISCARD MU;
  END;
  RETURN;
END CSC_MGR.T1_OR_T2_RCV;
```

```
CSC_MGR.T4_OR_T5_RCV: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS  PROCEDURE RECEIVES  THE  REQUESTS  AND RESPONSES  FOR  SESSION  │
│                 ACTIVATION  AND DEACTIVATION  FROM  CSC_MGR.RCV  (PAGE 13-35).   THE  │
│                 REQUEST OR RESPONSE IS VERIFIED (E.G.,  CHECKING THAT THE REQUEST OR  │
│                 RESPONSE CAN FLOW ON THE SESSION, CHECKING PARAMETERS WITHIN THE RU,   │
│                 (E.G., FM AND  TS PROFILES), AND MAKING THE  STATE  RECEIVE CHECKS).   │
│                 THE SESSION CONTROL  BLOCK IS CREATED, IF  NECESSARY, AND PARAMETERS   │
│                 ARE RETAINED FOR THE ACTIVATION OF THE SESSION.  IF THE VERIFICATION   │
│                 OF THE RU IS  SUCCESSFUL, THE RU IS SENT TO  THE NAU.SVC_MGR; IF THE   │
│                 VERIFICATION IS NOT SUCCESSFUL, THEN:                                  │
│                                                                                        │
│                    • THE REQUEST IS CHANGED TO A -RSP  AND SENT BACK TO PATH CONTROL   │
│                      (CHAPTER 3)                                                       │
│                    • IN THE SPECIAL CASE OF A DEACTIVATION RU COMING OVER A VR OTHER   │
│                      THAN THE VR ASSIGNED FOR THAT SESSION, THE RU IS DISCARDED.       │
│                    • THE RESPONSE IS DISCARDED                                         │
│                    • IN   THE  SPECIAL   CASE   OF   +RSP  (084E--(INVALID   SESSION   │
│                      PARAMETERS--PRI)),  THE  RESPONSE  IS  CHANGED  TO  A  NEGATIVE   │
│                      RESPONSE AND SENT TO THE SVC_MGR.                                 │
│                                                                                        │
│      INPUT:     ACTIVATION OR DEACTIVATION REQUESTS OR RESPONSES                        │
│                                                                                        │
│      OUTPUT:    THE   REQUEST  OR   RESPONSE   IS  SENT   TO   THE  NAU.SVC_MGR;   A   │
│                 CSC_MGR.RCV-GENERATED -RSP IS SENT TO PATH CONTROL (CHAPTER 3).        │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL SAVE_MU_PTR PTR;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │ INITIALIZE   PARAMETERS;  MUCB.DIRECTION   IS  │
                  │ ALREADY SET.                                   │
                  └──────────────────────────────────────────┘
                                                                        */
  NRCB_PTR = LOCATE_NODE_RESOURCE(DEF);                 /* APPENDIX B      */
  FIND SCB IN SCB_LIST
       WHERE(SCB.PARTNER_SA = OSAF &
       SCB.PARTNER_EA = OEF &
       SCB.THIS_SA = DSAF &
       SCB.THIS_EA = DEF);
  IF NRCB_PTR = NULL |
       NRCB.RESOURCE_CATEGORY ¬= (PU | LU | SSCP | BF.PU | BF.LU) THEN
    DO;
    . IF RRI = RSP THEN
    .   DISCARD MU;
    . ELSE                                              /* REQUEST         */
    .   DO;
    .   . RECEIVE_CHECK_SENSE = X'8004';   /* UNRECOGNIZED DESTINATION ADDRESS */
    .   . CALL CHANGE_MU_TO_NEG_RSP(RECEIVE_CHECK_SENSE);  /* APPENDIX B     */
    .   . SEND MU TO PC.VRC.SEND;                       /* CHAPTER 3       */
    .   END;
    . RETURN;
    END;
  ELSE
    IF NRCB.RESOURCE_CATEGORY = BF.PU |
         NRCB.RESOURCE_CATEGORY = BF.LU THEN
      CB_TYPE = BF_SESS;
    ELSE
      CB_TYPE = HALF_SESS;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │ CB_TYPE,    SCB_PTR,   AND    NRCB_PTR    NOW  │
                  │ INITIALIZED                                    │
                  └──────────────────────────────────────────┘
                                                                        */
  SELECT ANYORDER;
  . WHEN(RRI = RQ)
  .   SELECT ANYORDER(RQ_CHECKS);                       /* PAGE 13-48      */
  .   . WHEN(RQ_OK)
  .   .   DO;
  .   .   . IF SCB_PTR = NULL THEN
  .   .   .   DO;
  .   .   .   . CALL SCB_CREATE;                       /* PAGE 13-87      */
  .   .   .   . IF SCB_PTR = NULL THEN                 /* SCB NOT CREATED */
  .   .   .   .   DO;
  .   .   .   .   . CALL CHANGE_MU_TO_NEG_RSP(X'0812'); /* APPENDIX B, INSUFFICIENT RESOURCE */
  .   .   .   .   . SEND MU TO PC.VRC.SEND;           /* CHAPTER 3       */
  .   .   .   .   END;
  .   .   .   . ELSE
  .   .   .   .   IF NCB.PU_TYPE = (T4 | T5) THEN
  .   .   .   .     DO;
  .   .   .   .     . SCB.VRCBPTR = VRCB_PTR;          /* CONNECT VRCB TO SCB */
  .   .   .   .     . VRCB.SESS_COUNT = VRCB.SESS_COUNT + 1;
  .   .   .   .     END;
  .   .   .   END;
  .   .   . ELSE                                      /* SCB NOT NULL    */
  .   .   .   IF RQ_CODE = ACTCDRM THEN               /* CONTENTION WINNER */
  .   .   .     DO;                 /* REPLACE VRCBPTR IN SCB WITH NEW VRCB_PTR */
  .   .   .     . IF SCB.VRCBPTR ¬= NULL THEN
  .   .   .     .   DO;
  .   .   .     .   . SCB.VRCBPTR->SESS_COUNT = SCB.VRCBPTR->SESS_COUNT - 1;
  .   .   .     .   . IF SCB.VRCBPTR->SESS_COUNT <= 0 THEN
  .   .   .     .   .   SEND 'SESS_COUNT = 0'
  .   .   .     .   .        TO PU.SVC_MGR.PC_ROUTE_MGR.RCV
  .   .   .     .   .        USING(VRCB_PTR = SCB.VRCBPTR);
```

```
.   .   .     .   END;
.   .   .     . SCB.VRCBPTR = VRCB_PTR;                          /* CONNECT VRCB TO SCB      */
.   .   .     . VRCB.SESS_COUNT = VRCB.SESS_COUNT + 1;
.   .   .     END;
.   .   . CALL #FSM_SESS;                  /* PAGES 13-91 THROUGH 13-98                      */
.   .   . SEND MU TO #SVC_MGR;
.   .   END;
.   . WHEN(RQ_NG)
.   .   DO;
.   .   . CALL CHANGE_MU_TO_NEG_RSP(RECEIVE_CHECK_SENSE);      /* APPENDIX B               */
.   .   . SEND MU TO PC.VRC.SEND;                              /* CHAPTER 3                */
.   .   END;
.   . WHEN(RQ_WRONG_VR)
.   .   DO;
.   .   . CALL UPM_LOG('WRONG_VR');                            /* APPENDIX B               */
.   .   . DISCARD MU;
.   .   END;
.   END;
. WHEN(RRI = RSP)
.   IF RSP_CHECKS = RSP_OK THEN                                /* PAGE 13-49               */
.     DO;
.     . IF MUCB.DIRECTION = RECEIVE & RECEIVE_CHECK_SENSE = X'084E' THEN
.     .                                /* INVALID SESSION PARAMETERS--PRI                  */
.     .   DO;
.     .   . RTI = NEG;                 /* TURN INTO -RSP, WILL CAUSE UNBIND TO FLOW        */
.     .   . SNC = RECEIVE_CHECK_SENSE;
.     .   END;
.     . CALL #FSM_SESS;                /* PAGES 13-91 THROUGH 13-98                        */
.     . SEND MU TO #SVC_MGR;           /* SCB WASN'T DISCARDED BY FSM                      */
.     . IF (RTI = NEG) | (RQ_CODE = UNBIND | DACTLU | DACTPU | DACTCDRM)
.     .       & (SNC ¬= X'080D' | X'080E') THEN
.     .   CALL SCB_DISCARD;                                     /* PAGE 13-88               */
.     END;
.   ELSE
.     DISCARD MU;
. END;
. RETURN;
END CSC_MGR.T4_OR_T5_RCV;
```

```
CSC_MGR.BF_RCV: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE RECEIVES THE REQUESTS AND RESPONSES FOR SESSION │
│                 ACTIVATION AND DEACTIVATION FROM PATH CONTROL (CHAPTER 3) TO BE SENT │
│                 TO THE (NAU|BF.LU|BF.PU).SVC_MGR.  THE REQUEST OR RESPONSE IS │
│                 VERIFIED (E.G., CHECKING THAT THE REQUEST OR RESPONSE CAN FLOW ON │
│                 THE SESSION, CHECKING PARAMETERS SUCH AS FM AND TS PROFILES WITHIN │
│                 THE RU, AND MAKING THE STATE RECEIVE CHECKS).  THE SESSION CONTROL │
│                 BLOCK IS CREATED, IF NECESSARY, AND PARAMETERS ARE RETAINED FOR THE │
│                 ACTIVATION OF THE SESSION.  IF THE VERIFICATION OF THE RU IS │
│                 SUCCESSFUL, THE RU IS SENT TO THE (BF.LU|BF.PU).SVC_MGR; IF THE │
│                 VERIFICATION IS NOT SUCCESSFUL, THE REQUEST IS CHANGED TO A -RSP AND │
│                 SENT BACK TO PATH CONTROL (CHAPTER 3)                    │
│                                                                          │
│      INPUT:     ACTIVATION OR DEACTIVATION REQUESTS OR RESPONSES          │
│                                                                          │
│      OUTPUT:    THE REQUEST OR RESPONSE IS SENT TO (BF.LU|BF.PU).SVC_MGR; A │
│                 CSC_MGR.RCV-GENERATED -RSP IS SENT TO BOUNDARY FUNCTION PATH CONTROL │
│                 (CHAPTER 3).                                             │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL FOUND BIT(1);
                                                                        /*
                  ┌──────────────────────────────────────────────┐
                  │ INITIALIZE PARAMETERS; MUCB.DIRECTION IS     │
                  │ ALREADY SET.                                 │
                  └──────────────────────────────────────────────┘
                                                                        */
  CB_TYPE = BF_SESS;
  FOUND = NO;
  SCAN NRCB_LIST PTR(NRCB_PTR) WHILE(¬FOUND);
  . IF NRCB.RESOURCE_CATEGORY = (BF.PU | BF.LU) &
  .       ((NRCB.RESOURCE_TYPE = PU_T1 &
  .       NRCB.BF_LOCAL_ID = B'00000000000'||LSID(2:7)) |
  .       (NRCB.RESOURCE_TYPE = PU_T2 &
  .       NRCB.BF_LOCAL_ID = X'00'||OAFFPRIME)) THEN
  .    DO;
  .    . P = FIND_ALS_FOR_RESOURCE(NRCB.ELEMENT_ADDRESS);          /* APPENDIX B      */
  .    . IF P->NRCB.ELEMENT_ADDRESS = LSCB_EA THEN
  .    .    FOUND = YES;
  .    END;
  SCANEND;
  IF FOUND = NO | NRCB.RESOURCE_CATEGORY ¬= (PU | LU) THEN
    DO;
    . IF RRI = RSP THEN
    .    DISCARD MU;
    . ELSE
    .    DO;
    .    . CALL CHANGE_MU_TO_NEG_RSP(X'8004');
    .    .       /* APPENDIX B, UNRECOGNIZED DESTINATION ADDRESS                */
    .    . IF SCB_PTR ¬= NULL THEN
    .    .    LSCB_PTR = SCB.ALS_FOR_BF;                             /* USED IN BF.PC   */
    .    . SEND MU TO BF.PC;                                         /* CHAPTER 3       */
    .    END;
    . RETURN;
    END;
  ELSE
    IF NRCB.RESOURCE_TYPE = PU_T1 THEN
      FIND SCB IN SCB_LIST                 /* SCB_PTR IS NULL IF AN SCB          */
           WHERE(SCB.LOCAL_SESSION_ID = LSID);                       /* DOES NOT YET EXIST */
    ELSE
      FIND SCB IN SCB_LIST                 /* SCB_PTR IS NULL IF AN SCB          */
           WHERE(SCB.PARTNER_ID = DAFFPRIME &                        /* DOES NOT YET EXIST */
           SCB.THIS_ID = OAFFPRIME);
                                                                        /*
                  ┌──────────────────────────────────────────────┐
                  │ CB_TYPE, SCB_PTR, AND NRCB_PTR NOW           │
                  │ INITIALIZED                                  │
                  └──────────────────────────────────────────────┘
                                                                        */
  SELECT ANYORDER(RRI);
  . WHEN(RQ)
  .    IF RQ_CHECKS = RQ_OK THEN                                      /* PAGE 13-48      */
  .      DO;
  .      . CALL #FSM_SESS;                 /* PAGES 13-91 THROUGH 13-98          */
  .      . SEND MU TO #SVC_MGR;
  .      END;
  .    ELSE
  .      DO;
  .      . CALL CHANGE_MU_TO_NEG_RSP(RECEIVE_CHECK_SENSE);           /* APPENDIX B      */
  .      . IF SCB_PTR ¬= NULL THEN
  .      .    LSCB_PTR = SCB.ALS_FOR_BF;                             /* USED IN BF.PC   */
  .      . SEND MU TO BF.PC;                                         /* CHAPTER 3       */
  .      END;
  . WHEN(RSP)
  .    IF RSP_CHECKS = RSP_OK THEN                                    /* PAGE 13-49      */
  .      DO;
  .      . CALL #FSM_SESS;                 /* PAGES 13-91 THROUGH 13-98          */
  .      . SEND MU TO #SVC_MGR;
  .      END;
  .    ELSE
  .      DISCARD MU;
  END;
  RETURN;
END CSC_MGR.BF_RCV;
```

```
CSC_MGR.SON: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  PERFORMS A  ROUTING FUNCTION FOR  SESSION OUTAGE  NOTIFICATION (SON)
  │                 PROCESSING BY CALLING THE APPROPRIATE  SON PROCEDURES BASED UPON THE
  │                 CAUSE OF THE SESSION OUTAGE.
  │
  │      INPUT:     THE SIGNAL "SON-CAUSE" WITH PARAMETERS AS DEFINED BELOW:
  │                   • "VRINOP" OR "DACTVR_FORCED"; PARAMETER: VRCB_PTR FROM VR_MGR
  │                   • "HIERARCHICAL_RESET"; PARAMETER: SCB_PTR  FROM CSC_MGR.SEND (VIA
  │                     FSM'S)
  │                   • "SSCP_GONE"; PARAMETER: SCB_PTR FROM CSC_MGR.SEND (VIA FSM'S)
  │                   • "REX_INOP"; PARAMETER: ELEMENT ADDRESS  OF ADJACENT LINK STATION
  │                     FROM CHAPTER 11.
  │
  │      OUTPUT:    INPUT IS ROUTED TO APPROPRIATE PROCEDURES
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL CSC_MGR_SON_SCB_PTR PTR;

  CSC_MGR_SON_SCB_PTR = SCB_PTR;                                /* SAVE SCB POINTER        */

  SELECT ANYORDER;
  .
  . WHEN(INPUT('VRINOP') | INPUT('DACTVR_FORCED'))
  .   CALL SON_VR;                                              /* PAGE 13-60              */
  .
  . WHEN(INPUT('HIERARCHICAL_RESET'))
  .   CALL SON_RESET;                                           /* PAGE 13-60              */
  . WHEN(INPUT('SSCP_GONE'))
  .   DO;
  .   . CALL SON_RESET;                                         /* PAGE 13-60              */
  .   . SCB_PTR = CSC_MGR_SON_SCB_PTR;     /* RESTORE THE SCB POINTER                     */
  .   . CALL SCB_DISCARD;                                       /* PAGE 13-88              */
  .   END;
  .
  . WHEN(INPUT('REX_INOP'))
  .   CALL SON_REX_INOP;                                        /* PAGE 13-64              */
  END;
  RETURN;
END CSC_MGR.SON;
```

```
RQ_CHECKS: PROCEDURE RETURNS(BIT(2));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE VERIFIES THAT THE  ACTIVATION OR DEACTIVATION REQUEST  │
   │                 IS  VALID, AND  THAT  A SESSION  CONTROL BLOCK  CAN  BE CREATED,  IF  │
   │                 NECESSARY.                                                   │
   │                                                                              │
   │      INPUT:     ACTIVATION  AND DEACTIVATION  REQUESTS  FROM T1_T2_OR_BF_SEND  (PAGE  │
   │                 13-37), OR T4_OR_T5_SEND (PAGE 13-38), OR CSC_MGR.RCV (PAGE 13-41).  │
   │                                                                              │
   │      OUTPUT:    A RETURN CODE OF RQ_OK, RQ_NG, OR RQ_WRONG_VR                │
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(2);
  RC = RQ_NG;

  SELECT ANYORDER;
  . WHEN(SCB_PTR ¬= NULL)
  .   DO;
  .   . IF TYPE_SESSION = OK &                                  /* PAGE 13-51    */
  .   .      RQ_PARAMETERS = OK THEN                            /* PAGE 13-55    */
  .   .   DO;
  .   .   . IF  ¬SEND_OR_RECEIVE_CHECK(@FSM_SESS) THEN /*PAGES 13-91 THROUGH 13-98  */
  .   .   .   RC = RQ_OK;
  .   .   . ELSE
  .   .   .   IF MUCB.DIRECTION = RECEIVE THEN
  .   .   .     SNC = X'0809';                                  /* MODE INCONSISTENCY */
  .   .   .   ELSE
  .   .   .     RECEIVE_CHECK_SENSE = X'0809';                  /* MODE INCONSISTENCY */
  .   .   . IF NCB.PU_TYPE = (T4 | T5) &
  .   .   .      MUCB.DIRECTION = RECEIVE &
  .   .   .      RQ_CODE = (DACTCDRM | DACTLU | DACTPU | UNBIND) &
  .   .   .      VRCB_PTR ¬= SCB.VRCBPTR THEN
  .   .   .   RC = RQ_WRONG_VR;          /* PIU CAME OVER DIFFERENT VR          */
  .   .   END;
  .   END;
  . WHEN(SCB_PTR = NULL)
  .   IF FUNCTION_SUPPORTED = OK &                              /* PAGE 13-53    */
  .      RQ_PARAMETERS = OK THEN                                /* PAGE 13-55    */
  .     RC = RQ_OK;
  END;
  RETURN(RC);
END RQ_CHECKS;
```

```
RSP_CHECKS: PROCEDURE RETURNS(BIT(2));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE VERIFIES THAT THE ACTIVATION OR DEACTIVATION RESPONSE │
  │                 IS VALID.                                                    │
  │                                                                              │
  │      INPUT:     ACTIVATION AND  DEACTIVATION RESPONSES  FROM T1_T2_OR_BF_SEND  (PAGE │
  │                 13-37), T4_OR_T5_SEND (PAGE 13-38), OR CSC_MGR.RCV (PAGE 13-41). │
  │                                                                              │
  │      OUTPUT:    A RETURN CODE OF RSP_OK OR RSP_NG                            │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(2);
  RC = RSP_NG;
  SELECT ANYORDER;
  .
  . WHEN(SCB_PTR ¬= NULL)
  .   DO;
  .   . IF TYPE_SESSION = OK &                                  /* PAGE 13-51    */
  .   .      RSP_PARAMETERS = OK THEN                           /* PAGE 13-56    */
  .   .   IF  ¬SEND_OR_RECEIVE_CHECK(#FSM_SESS) THEN /* PAGES 13-91 THROUGH 13-98 */
  .   .     RC = RSP_OK;
  .   .   ELSE
  .   .     IF MUCB.DIRECTION = RECEIVE THEN
  .   .       RECEIVE_CHECK_SENSE = X'0809';                   /* MODE INCONSISTENCY */
  .   .     ELSE
  .   .       SEND_CHECK_SENSE = X'0809';                      /* MODE INCONSISTENCY */
  .   END;
  .
  . WHEN(SCB_PTR = NULL)
  .   IF MUCB.DIRECTION = RECEIVE THEN
  .     RECEIVE_CHECK_SENSE = X'8005';                         /* NO SESSION    */
  .   ELSE
  .     SEND_CHECK_SENSE = X'8005';                            /* NO SESSION    */
  .
  END;
  RETURN(RC);
END RSP_CHECKS;
```

```
TYPE_SESSION: PROCEDURE RETURNS(BIT(1));

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE  VERIFIES THAT  THE REQUEST  OR RESPONSE  IS FOR  THE │
 │                 CORRECT  TYPE OF  SESSION.  THIS  PROCEDURE  IS CALLED  ONLY WHEN  A │
 │                 SESSION CONTROL  BLOCK IS PRESENT.  THIS PROCEDURE ALSO  LIMITS THE │
 │                 SENDING OF ACTIVATION AND DEACTIVATION REQUESTS.  FOR EXAMPLE, ACTLU │
 │                 AND  DACTLU CANNOT  BE  SENT  BY THE  LU;  HOWEVER,  DACTLU CAN  BE │
 │                 RECEIVED  BY  THE  SSCP;  IT CAN  BE  GENERATED  BY  SESSION  OUTAGE │
 │                 NOTIFICATION.                                                │
 │                                                                              │
 │      INPUT:     CURRENT RU, WHICH IS AN ACTIVATION OR DEACTIVATION RU.       │
 │                                                                              │
 │      OUTPUT:    A RETURN CODE OF OK OR NG                                    │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);
  RC = NG;
  SELECT ANYORDER;
  .
  . WHEN(SCB.TYPE_OF_SESSION = SSCP_SSCP & RQ_CODE = (ACTCDRM | DACTCDRM))
  .   RC = OK;
  .
  . WHEN(SCB.TYPE_OF_SESSION = SSCP_LU & SCB.SCB_TYPE = HALF_SESS)
  .   DO;
  .   . IF SCB.HALF_SESSION = PRI THEN     /* THIS HALF-SESSION IS IN AN SSCP          */
  .   .   DO;
  .   .   . IF MUCB.DIRECTION = SEND &
  .   .   .       ((RRI = RQ & RQ_CODE = (ACTLU | DACTLU)) |
  .   .   .       (RRI = RSP & RQ_CODE = DACTLU)) THEN
  .   .   .   RC = OK;
  .   .   . IF MUCB.DIRECTION = RECEIVE &
  .   .   .       ((RRI = RQ & RQ_CODE = DACTLU) |
  .   .   .       (RRI = RSP & RQ_CODE = (ACTLU | DACTLU))) THEN
  .   .   .   RC = OK;
  .   .   END;
  .   . IF SCB.HALF_SESSION = SEC THEN     /* THIS HALF-SESSION IS IN AN LU            */
  .   .   DO;
  .   .   . IF MUCB.DIRECTION = SEND &
  .   .   .       RRI = RSP & (RQ_CODE = (ACTLU | DACTLU)) THEN
  .   .   .   RC = OK;
  .   .   . IF MUCB.DIRECTION = RECEIVE &
  .   .   .       RRI = RQ & (RQ_CODE = (ACTLU | DACTLU)) THEN
  .   .   .   RC = OK;
  .   .   END;
  .   END;
  .
  . WHEN(SCB.TYPE_OF_SESSION = SSCP_LU & SCB.SCB_TYPE = BF_SESS)
  .   DO;
  .   .           /* THIS CHECK DETERMINES THE DIRECTION OF THE RU.                */
  .   . IF NRCB.ELEMENT_ADDRESS = DEF THEN
  .   .                                   /* RU RECEIVED FROM PC OR CSC_MGR.SON:  */
  .   .                                                   /* SENDING TO BF.PC  */
  .   .  IF RRI = RQ & RQ_CODE = (ACTLU | DACTLU) THEN
  .   .    RC = OK;
  .   .   ELSE;
  .   . ELSE                              /* RECEIVED FROM BF.PC OR CSC_MGR.SON:  */
  .   .                                                   /* SENDING TO PC     */
  .   .   IF RRI = RSP & RQ_CODE = (ACTLU | DACTLU) THEN
  .   .     RC = OK;
  .   END;
  .
  . WHEN(SCB.TYPE_OF_SESSION = SSCP_PU & SCB.SCB_TYPE = HALF_SESS)
  .   DO;
  .   . IF SCB.HALF_SESSION = PRI THEN     /* THIS HALF-SESSION IS IN AN SSCP          */
  .   .   DO;
  .   .   . IF MUCB.DIRECTION = SEND &
  .   .   .       ((RRI = RQ & RQ_CODE = (ACTPU | DACTPU)) |
  .   .   .       (RRI = RSP & RQ_CODE = DACTPU)) THEN
  .   .   .   RC = OK;
  .   .   . IF MUCB.DIRECTION = RECEIVE &
  .   .   .       ((RRI = RQ & RQ_CODE = DACTPU) |
  .   .   .       (RRI = RSP & RQ_CODE = (ACTPU | DACTPU))) THEN
  .   .   .   RC = OK;
  .   .   END;
  .   . IF SCB.HALF_SESSION = SEC THEN     /* THIS HALF-SESSION IS IN A PU             */
  .   .   DO;
  .   .   . IF MUCB.DIRECTION = SEND &
  .   .   .       RRI = RSP & (RQ_CODE = (ACTPU | DACTPU)) THEN
  .   .   .   RC = OK;
  .   .   . IF MUCB.DIRECTION = RECEIVE &
  .   .   .       RRI = RQ & (RQ_CODE = (ACTPU | DACTPU)) THEN
  .   .   .   RC = OK;
  .   .   END;
  .   END;
  .
```

```
. WHEN(SCB.TYPE_OF_SESSION = SSCP_PU & SCB.SCB_TYPE = BF_SESS)
.   DO;
.   .                                          /* THIS CHECK DETERMINES THE DIRECTION OF THE RU. */
.   . IF NRCB.ELEMENT_ADDRESS = DEF THEN
.   .                                          /* RU RECEIVED FROM PC OR CSC_MGR.SON:           */
.   .                                                                  /* SENDING TO BF.PC      */
.   .   IF RRI = RQ & RQ_CODE = (ACTPU | DACTPU) THEN
.   .     RC = OK;
.   .   ELSE;
.   . ELSE                                     /* RECEIVED FROM BF.PC OR CSC_MGR.SON:           */
.   .                                                                  /* SENDING TO PC         */
.   .   IF RRI = RSP & RQ_CODE = (ACTPU | DACTPU) THEN
.   .     RC = OK;
.   END;
.
. WHEN(SCB.TYPE_OF_SESSION = LU_LU & SCB.SCB_TYPE = HALF_SESS)
.   DO;
.   . IF SCB.HALF_SESSION = PRI THEN
.   .   DO;
.   .   . IF MUCB.DIRECTION = SEND &
.   .   .       ((RRI = RQ & RQ_CODE = (BIND | UNBIND)) |
.   .   .       (RRI = RSP & RQ_CODE = UNBIND)) THEN
.   .   .   RC = OK;
.   .   . IF MUCB.DIRECTION = RECEIVE &
.   .   .       ((RRI = RQ & RQ_CODE = UNBIND) |
.   .   .       (RRI = RSP & RQ_CODE = (BIND | UNBIND))) THEN
.   .   .   RC = OK;
.   .   END;
.   . IF SCB.HALF_SESSION = SEC THEN
.   .   DO;
.   .   . IF MUCB.DIRECTION = SEND &
.   .   .       ((RRI = RQ & RQ_CODE = UNBIND) |
.   .   .       (RRI = RSP & RQ_CODE = (BIND | UNBIND))) THEN
.   .   .   RC = OK;
.   .   . IF MUCB.DIRECTION = RECEIVE &
.   .   .       ((RRI = RQ & RQ_CODE = (BIND | UNBIND)) |
.   .   .       (RRI = RSP & RQ_CODE = UNBIND)) THEN
.   .   .   RC = OK;
.   .   END;
.   END;
.
. WHEN(SCB.TYPE_OF_SESSION = LU_LU & SCB.SCB_TYPE = BF_SESS)
.   DO;
.   .                                          /* THIS CHECK DETERMINES THE DIRECTION OF THE RU. */
.   . IF NRCB.ELEMENT_ADDRESS = DEF THEN
.   .                                          /* RU RECEIVED FROM PC OR CSC_MGR.SON:           */
.   .                                                                  /* SENDING TO BF.PC      */
.   .   IF (RRI = RQ & RQ_CODE = (BIND | UNBIND)) |
.   .       (RRI = RSP & RQ_CODE = UNBIND) THEN
.   .     RC = OK;
.   .   ELSE;
.   . ELSE                                     /* RECEIVED FROM BF.PC OR CSC_MGR.SON:           */
.   .                                                                  /*SENDING TO PC          */
.   .   IF (RRI = RQ & RQ_CODE = UNBIND) |
.   .       (RRI = RSP & RQ_CODE = (BIND | UNBIND)) THEN
.   .     RC = OK;
.   END;
  END;
  IF RC = NG THEN
    IF MUCB.DIRECTION = RECEIVE THEN
      RECEIVE_CHECK_SENSE = X'0809';                                /* MODE INCONSISTENCY    */
    ELSE
      SEND_CHECK_SENSE = X'0809';                                   /* MODE INCONSISTENCY    */
  RETURN(RC);
END TYPE_SESSION;
```

This page intentionally left blank

```
FUNCTION_SUPPORTED: PROCEDURE RETURNS(BIT(1));

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE  VERIFIES THAT  THE REQUEST IS  FOR THE  CORRECT TYPE │
 │                 NAU|BF.LU|BF.PU.  THIS  PROCEDURE IS  CALLED ONLY  WHEN THE  SESSION │
 │                 CONTROL BLOCK IS NOT PRESENT.                                │
 │                                                                              │
 │      INPUT:     CURRENT RU, WHICH IS AN ACTIVATION OR DEACTIVATION REQUEST.   │
 │                                                                              │
 │      OUTPUT:    A RETURN CODE OF OK OR NG                                    │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);
  DCL SENSE BIT(16);
  SENSE = 0;

  RC = NG;
  IF CB_TYPE = BF_SESS THEN
    DO;
    . IF NRCB.ELEMENT_ADDRESS ¬= DEF THEN
    .   SENSE = X'8005';                                    /* NO SESSION       */
    . ELSE
    .   IF RQ_CODE = (ACTPU | ACTLU | BIND) THEN
    .     RC = OK;
    .   ELSE
    .     SENSE = X'8005';                                  /* NO SESSION       */
    END;
  IF MUCB.DIRECTION = SEND & CB_TYPE = HALF_SESS THEN
    SELECT ANYORDER(NRCB.RESOURCE_CATEGORY);
    .
    . WHEN(SSCP)
    .   IF RQ_CODE = (ACTCDRM | ACTLU | ACTPU) THEN
    .     RC = OK;
    .   ELSE
    .     SENSE = X'8005';                                  /* NO SESSION       */
    .
    . WHEN(LU)
    .   IF RQ_CODE = BIND THEN
    .     IF NCB.PU_TYPE = (T1 | T2) THEN
    .       SENSE = X'0809';                                /* MODE INCONSISTENCY */
    .     ELSE
    .       RC = OK;
    .   ELSE
    .     SENSE = X'8005';                                  /* NO SESSION       */
    .
    . WHEN(PU)
    .   SENSE = X'8005';                                    /* NO SESSION       */
    END;
  IF MUCB.DIRECTION = RECEIVE & CB_TYPE = HALF_SESS THEN
    SELECT ANYORDER(NRCB.RESOURCE_CATEGORY);
    . WHEN(SSCP)
    .   IF RQ_CODE = ACTCDRM THEN
    .     RC = OK;
    .   ELSE
    .     IF RQ_CODE = DACTCDRM THEN
    .       SENSE = X'0809';                                /* MODE INCONSISTENCY */
    .     ELSE
    .       SENSE = X'8005';                                /* NO SESSION       */
    . WHEN(LU)
    .   IF RQ_CODE = (ACTLU | BIND) THEN
    .     RC = OK;
    .   ELSE
    .     SENSE = X'8005';                                  /* NO SESSION       */
    . WHEN(PU)
    .   IF RQ_CODE = ACTPU THEN
    .     RC = OK;
    .   ELSE
    .     IF RQ_CODE = DACTPU THEN
    .       SENSE = X'0809';                                /* MODE INCONSISTENCY */
    .     ELSE
    .       SENSE = X'8005';                                /* NO SESSION       */
    END;
  IF SENSE ¬= 0 THEN
    IF MUCB.DIRECTION = SEND THEN
      SEND_CHECK_SENSE = SENSE;
    ELSE
      RECEIVE_CHECK_SENSE = SENSE;
  RETURN(RC);
END FUNCTION_SUPPORTED;
```

```
RQ_PARAMETERS: PROCEDURE RETURNS(BIT(1));

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE  VERIFIES THAT THE  PARAMETERS THAT ARE  CONTAINED IN │
│                 THE RQ  ARE VALID.  IN A  NODE PROVIDING BOUNDARY  FUNCTION SUPPORT, │
│                 THE BF.(PU | LU).SVC_MGR CHECKS THAT THE PARAMETERS ARE VALID.       │
│                                                                          │
│      INPUT:     THE CURRENT RQ, WHICH IS AN ACTIVATION OR DEACTIVATION REQUEST.      │
│                                                                          │
│      OUTPUT:    A   RETURN   CODE    OF   OK   OR   NG--THE    SEND_CHECK_SENSE   OR │
│                 RECEIVE_CHECK_SENSE IS SET IF THE RETURN CODE IS NG.     │
│                                                                          │
│      NOTES:     1. THE CHECK OF OEF = 0 IS A CHECK  TO SEE IF THE ORGINATOR IS A PU, │
│                    THE ELEMENT  ADDRESS OF  THE PU  IS DEFINED  IN CHAPTER  2.  THIS │
│                    CHECK IS OPTIONAL.                                    │
│                                                                          │
│                 2. WHERE THE SENSE CODE OF 0835 IS GENERATED, 0821 AND 0833 ARE ALSO │
│                    ALLOWED.                                              │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL SAVE_MU_PTR PTR;
DCL SAVE_VRCB_PTR PTR;
DCL RC BIT(1);
DCL SENSE BIT(32);
SENSE = 0;

RC = NG;
SELECT ANYORDER(RQ_CODE);
. WHEN(ACTCDRM)
.   SELECT INORDER;
.   . WHEN(OEF = 0)                                          /* SEE NOTES          */
.   .   SENSE = X'8005';                                     /* NO SESSION         */
.   . WHEN(ACTCDRM_RQ.FORMAT ¬= 0 | ACTCDRM_RQ.TYPE_ACTIVATION ¬= (COLD | ERP))
.   .   SENSE = X'08350001';                                 /* INVALID PARAMETERS */
.   . WHEN(ACTCDRM_RQ.FM_PROFILE ¬= 17)
.   .   SENSE = X'08350002';                                 /* INVALID PARAMETERS */
.   . WHEN(ACTCDRM_RQ.TS_PROFILE ¬= 17)
.   .   SENSE = X'08350003';                                 /* INVALID PARAMETERS */
.   . WHEN(MUCB.DIRECTION = RECEIVE & SCB_PTR ¬= NULL &
.   .      SCB.ACT_RQ_RSP_SEQ_ID >= UPM_GET_SEQ_ID)          /* PAGE 13-88         */
.   .   SENSE = X'0852';                 /* SESSION EXISTS FROM LATER ACTIVATION */
.   . OTHERWISE
.   .                                                        /* VALID PARAMETERS   */
.   .   IF SCB_PTR = NULL THEN
.   .     RC = OK;
.   .   ELSE                                                 /* SCB EXISTS         */
.   .     IF SCB.VRCBPTR ¬= NULL & SCB.VRCBPTR ¬= VRCB_PTR &
.   .         FSM_SESS_SSCP_SSCP_PRI_OR_SEC ¬= ACTIVE THEN
.   .       DO; /* DON'T CHECK CONTENTION CASE IF SESSION IS ACTIVE, SIMPLY OVERRIDE */
.   .       . IF MUCB.DIRECTION = RECEIVE &
.   .       .      ACTCDRM_RQ.SSCP_ID <= SCB.THIS_HALF_SESSION_SSCP_ID THEN
.   .       .   SENSE = X'080D'; /* NAU CONTENTION, THE RECEIVED ACTCDRM IS THE LOSER */
.   .       . ELSE                          /* RECEIVED ACTCDRM IS THE WINNER */
.   .       .   IF SCB.VRCBPTR ¬= VRCB_PTR THEN
.   .       .     DO;                       /* FOLLOW LOSING ACTCDRM WITH DACTCDRM */
.   .       .     . SAVE_VRCB_PTR = VRCB_PTR;
.   .       .     . SAVE_MU_PTR = MU_PTR;
.   .       .     . CALL CREATE_DEACT_RQ(¬SWITCHED,
.   .       .     .      DACTCDRM,SSCP_CONTENTION,SEND);  /* PAGE 13-65         */
.   .       .     . VRCB_PTR = SCB.VRCBPTR;
.   .       .     . SEND MU TO PC.VRC.SEND;               /* CHAPTER 3          */
.   .       .     . MU_PTR = SAVE_MU_PTR;
.   .       .     . VRCB_PTR = SAVE_VRCB_PTR;
.   .       .     END;
.   .       END;
.   END;
. WHEN(ACTLU)
.   DO;
.   . SENSE = PU_ACTIVE_AND_VR_CHECK;                        /* PAGE 13-59         */
.   . IF SENSE = 0 THEN
.   .   SELECT INORDER;
.   .   . WHEN(NCB.PU_TYPE = (T4 | T5) & OEF = 0)           /* SEE NOTES          */
.   .   .   SENSE = X'8005';                                 /* NO SESSION         */
.   .   . WHEN(CB_TYPE = HALF_SESS)
.   .   .   SELECT INORDER;
.   .   .   . WHEN(MUCB.DIRECTION = SEND)
.   .   .   .   IF ACTLU_RQ.TYPE_ACTIVATION ¬= (COLD | ERP) THEN
.   .   .   .     SENSE = X'08350001';                       /* INVALID PARAMETERS */
.   .   .   . WHEN(ACTLU_RQ.FM_PROFILE ¬= (0 | 6) | ACTLU_RQ.TS_PROFILE ¬= 1)
.   .   .   .   SENSE = X'08350002';                         /* INVALID PARAMETERS */
.   .   .   . WHEN(MUCB.DIRECTION = RECEIVE)
.   .   .   .   IF ACTLU_RQ.TYPE_ACTIVATION ¬= (COLD | ERP) THEN
.   .   .   .     DO;
.   .   .   .     . ACTLU_RQ.TYPE_ACTIVATION = COLD;
.   .   .   .     . RC = OK;
.   .   .   .     END;
.   .   .   . OTHERWISE
.   .   .   .   RC = OK;
.   .   .   END;
.   .   . WHEN(CB_TYPE = BF_SESS)
.   .   .   RC = OK;                    /* PARAMETERS ARE CHECKED BY BF.LU.SVC_MGR */
.   .   END;
.   END;
. WHEN(ACTPU)
.   DO;
.   . IF CB_TYPE = BF_SESS & MUCB.DIRECTION = RECEIVE THEN
```

```
.   .   SENSE = PU_ACTIVE_AND_VR_CHECK;                          /* PAGE 13-59          */
.   . IF SENSE = 0 THEN
.   .   SELECT INORDER;
.   .   . WHEN(NUCB.DIRECTION = SEND & CB_TYPE = BF_SESS)
.   .   .   RC = OK;                          /*  PARAMETERS ARE CHECKED BY BF.PU.SVC_MGR  */
.   .   . WHEN(NUCB.DIRECTION = SEND & CB_TYPE = HALF_SESS)
.   .   .   IF ACTPU_RQ.TYPE_ACTIVATION ¬= (COLD | ERP) THEN
.   .   .     SENSE = X'08350001';                                /* INVALID PARAMETERS  */
.   .   . WHEN(CB_TYPE = HALF_SESS & (ACTPU_RQ.FM_PROFILE ¬= (0 | 5) |
.   .   .       ACTPU_RQ.TS_PROFILE ¬= (1 | 5)))
.   .   .   SENSE = X'08350002';                                  /* INVALID PARAMETERS  */
.   .   . WHEN(NUCB.DIRECTION = RECEIVE)
.   .   .   DO;
.   .   .   . IF SCB_PTR ¬= NULL &
.   .   .   .       SCB.ACT_RQ_RSP_SEQ_ID >= UPM_GET_SEQ_ID THEN /* PAGE 13-88          */
.   .   .   .   SENSE = X'0852';          /* SESSION EXISTS FROM LATER ACTIVATION        */
.   .   .   . IF ACTPU_RQ.TYPE_ACTIVATION ¬= (COLD | ERP) THEN
.   .   .   .   DO;
.   .   .   .   . RC = OK;
.   .   .   .   . ACTPU_RQ.TYPE_ACTIVATION = COLD;
.   .   .   .   END;
.   .   .   END;
.   .   . OTHERWISE
.   .   .   RC = OK;
.   .   END;
.   END;
. WHEN(BIND)
.   SELECT INORDER;
.   . WHEN(NCB.PU_TYPE = (T4 | T5) & OEF = 0)                    /* SEE NOTES           */
.   .   SENSE = X'8005';                                         /* NO SESSION          */
.   . WHEN(CB_TYPE = BF_SESS)
.   .   RC = OK;                          /* PARAMETERS ARE CHECKED BY BF.LU.SVC_MGR     */
.   . WHEN(BIND_RQ.FORMAT ¬= 0 | BIND_RQ.TYPE ¬= (NEGOTIABLE | NONNEGOTIABLE))
.   .   SENSE = X'08350001';                                     /* INVALID PARAMETERS  */
.   . WHEN(BIND_RQ.FM_PROFILE ¬= (2 | 3 | 4 | 7 | 18))
.   .   SENSE = X'08350002';                                     /* INVALID PARAMETERS  */
.   . WHEN(BIND_RQ.TS_PROFILE ¬= (2 | 3 | 4 | 7))
.   .   SENSE = X'08350003';                                     /* INVALID PARAMETERS  */
.   . OTHERWISE
.   .   RC = OK;
.   END;
. WHEN(DACTCDRM)
.   RC = OK;
. WHEN(DACTPU)
.   RC = OK;
. WHEN(DACTLU)
.   RC = OK;
. WHEN(UNBIND) /* IF UNBIND_RQ.TYPE NOT KNOWN BY LU.SVC_MGR THEN TYPE IS HANDLED          */
.   RC = OK;                                                     /* AS A NORMAL_END     */
END;
IF SENSE ¬= 0 THEN
  IF NUCB.DIRECTION = SEND THEN
    SEND_CHECK_SENSE = SENSE;
  ELSE
    RECEIVE_CHECK_SENSE = SENSE;
RETURN(RC);
END RQ_PARAMETERS;
```

```
RSP_PARAMETERS: PROCEDURE RETURNS(BIT(1));

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  THIS PROCEDURE VERIFIES THAT THE PARAMETERS CONTAINED IN THE RSP ARE │
 │                 VALID.  IN NODES  PROVIDING BOUNDARY FUNCTION SUPPORT,  THE BF.(PU | │
 │                 LU).SVC_MGR VERIFIES  THAT THE PARAMETERS  CONTAINED IN THE  RSP ARE │
 │                 VALID.                                                       │
 │                                                                              │
 │      INPUT:     THE CURRENT MU, WHICH IS AN ACTIVATION OR DEACTIVATION RESPONSE.  │
 │                                                                              │
 │      OUTPUT:    A  RETURN  CODE  OF  OK  OR   NG;  AND  THE  RECEIVE_CHECK_SENSE  OR │
 │                 SEND_CHECK_SENSE IS SET IF THE RETURN CODE IS NG.            │
 │                                                                              │
 │      NOTE:      WHERE THE SENSE  CODE OF 0835 IS  GENERATED, 0821 AND 0833  ARE ALSO │
 │                 ALLOWED.                                                     │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);
  DCL SENSE BIT(32);
  SENSE = 0;
  RC = NG;
  IF RTI = NEG THEN
    RC = OK;
  ELSE                                                      /* POSITIVE RESPONSE      */
    DO;
    . SELECT ANYORDER(RQ_CODE);
    . . WHEN(ACTCDRM)
    . .   SELECT INORDER;
    . .   . WHEN(ACTCDRM_RSP.FORMAT ¬= 0 | ACTCDRM_RSP.TYPE_ACTIVATION ¬= (COLD | ERP))
    . .   .   SENSE = X'08350001';                          /* INVALID PARAMETERS     */
    . .   . WHEN(ACTCDRM_RSP.FM_PROFILE ¬= 17)
    . .   .   SENSE = X'08350002';                          /* INVALID PARAMETERS     */
    . .   . WHEN(ACTCDRM_RSP.TS_PROFILE ¬= 17)
    . .   .   SENSE = X'08350003';                          /* INVALID PARAMETERS     */
    . .   . WHEN(SCB_PTR ¬= NULL &
    . .   .      SCB.ACT_RQ_RSP_SEQ_ID >= UPM_GET_SEQ_ID)   /* PAGE 13-88             */
    . .   .   SENSE = X'0852';                              /* SESSION EXISTS         */
    . .   . OTHERWISE                                       /* VALID PARAMETERS       */
    . .   .   RC = OK;
    . .   END;
    . . WHEN(ACTLU)
    . .   SELECT ANYORDER;
    . .   . WHEN(CB_TYPE = HALF_SESS)
    . .   .   SELECT INORDER;
    . .   .   . WHEN(ACTLU_RSP.TYPE_ACTIVATION ¬= (COLD | ERP))
    . .   .   .   SENSE = X'08350001';                      /* INVALID PARAMETERS     */
    . .   .   . WHEN(ACTLU_RSP.FM_PROFILE ¬= (0 | 6) | ACTLU_RSP.TS_PROFILE ¬= 1)
    . .   .   .   SENSE = X'08350002';                      /* INVALID PARAMETERS     */
    . .   .   . OTHERWISE                                   /* VALID PARAMETERS       */
    . .   .   .   RC = OK;
    . .   .   END;
    . .   . WHEN(CB_TYPE = BF_SESS)        /* BF.LU.SVC_MGR CHECKS PARAMETERS        */
    . .   .   RC = OK;
    . .   END;
    . . WHEN(ACTPU)
    . .   SELECT INORDER;
    . .   . WHEN(CB_TYPE = HALF_SESS)
    . .   .   IF ACTPU_RSP.TYPE_ACTIVATION ¬= (COLD | ERP) THEN
    . .   .     SENSE = X'08350001';                       /* INVALID PARAMETERS     */
    . .   . WHEN(MUCB.DIRECTION = SEND & CB_TYPE = BF_SESS)
    . .   .   RC = OK;
    . .   . OTHERWISE
    . .   .   RC = OK;
    . .   END;
```

```
    . . WHEN(BIND)
    . .   SELECT INORDER;
    . .   . WHEN(CB_TYPE = BF_SESS)        /* BF.LU.SVC_MGR CHECKS PARAMETERS                */
    . .   .   RC = OK;
    . .   . WHEN(DCF = RSP_OF_LENGTH_ONE)  /* NONEXTENDED NONNEGOTIABLE                      */
    . .   .   RC = OK;
    . .   . WHEN(BIND_RSP.FORMAT ¬= 0)
    . .   .   SENSE = X'08350001';                                  /* INVALID PARAMETERS    */
    . .   . WHEN(BIND_RSP.TYPE ¬= (NEGOTIABLE | NONNEGOTIABLE))
    . .   .   SENSE = X'08350001';                                  /* INVALID PARAMETERS    */
    . .   . WHEN(BIND_RSP.TYPE = NEGOTIABLE)
    . .   .   SELECT INORDER;
    . .   .   . WHEN(BIND_RSP.PS_PROFILE ¬= (2 | 3 | 4 | 7 | 18))
    . .   .   .   SENSE = X'08350002';                              /* INVALID PARAMETERS    */
    . .   .   . WHEN(BIND_RSP.TS_PROFILE ¬= (2 | 3 | 4 | 7))
    . .   .   .   SENSE = X'08350003';                              /* INVALID PARAMETERS    */
    . .   .   . WHEN(BIND_CRYPTOGRAPHY_CK ¬= OK)                    /* PAGE 13-58            */
    . .   .   .   SENSE = X'08350026';                              /* INVALID PARAMETERS    */
    . .   .   . OTHERWISE
    . .   .   .   RC = OK;
    . .   .   END;
    . .   END;
    . . WHEN(DACTCDRM)
    . .   RC = OK;
    . . WHEN(DACTPU)
    . .   RC = OK;
    . . WHEN(DACTLU)
    . .   RC = OK;
    . . WHEN(UNBIND)
    . .   RC = OK;
    . END;
    END;
  IF SENSE ¬= 0 THEN
    IF MUCB.DIRECTION = RECEIVE THEN
      DO;
      . RC = OK;
      . RECEIVE_CHECK_SENSE = X'084E';     /* INVALID SESSION PARAMETERS--PRI                */
      END;
    ELSE
      DO;
      . RC = NG;
      . SEND_CHECK_SENSE = SENSE;
      END;
  RETURN(RC);
END RSP_PARAMETERS;
```

```
BIND_CRYPTOGRAPHY_CK: PROCEDURE RETURNS(BIT(1));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  VERIFIES THAT THE CRYPTOGRAPHY OPTIONS  SELECTED ON THE BIND REQUEST │
   │                 ARE NOT DECREASED BY THE SECONDARY LU.                   │
   │                                                                          │
   │      INPUT:     NONE                                                     │
   │                                                                          │
   │      OUTPUT:    A RETURN CODE OF EITHER OK OR NG                         │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);

  RC = OK;
  IF DCF = RSP_OF_LENGTH_ONE THEN
    IF SCB.CRYPTOGRAPHY_SESSION_LEVEL ¬= 0 THEN
      RC = NG;                              /* NO SESSION SEED RETURNED          */
    ELSE
      DO;
      . IF SCB.CRYPTOGRAPHY_SESSION_LEVEL ¬=
      .       BIND_RSP.CRYPTOGRAPHY_SESSION_LEVEL THEN
      .    DO;
      .    . IF SCB.CRYPTOGRAPHY_SESSION_LEVEL = MANDATORY THEN
      .    .    RC = NG;                                    /* BIND HAD MANDATORY  */
      .    . IF BIND_RSP.CRYPTOGRAPHY_SESSION_LEVEL = NONE THEN
      .    .    RC = NG;                 /* RSP HAD NO SESSION-LEVEL CRYPTO       */
      .    .                                               /* ELSE, PROMOTION IS OK */
      .    END;
      END;
  RETURN(RC);
END BIND_CRYPTOGRAPHY_CK;
```

```
PU_ACTIVE_AND_VR_CHECK: PROCEDURE RETURNS(BIT(32));

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │     FUNCTION:  FOR ACTLU TO A SUBAREA NODE, THIS  PROCEDURE RETURNS A SENSE CODE OF │
   │                8008 IF THE SSCP IDENTIFIED BY THE  ORIGIN ADDRESS IN THE ACTLU DOES │
   │                NOT HAVE AN  ACTIVE SSCP-PU SESSION WITH THE PU  ASSOCIATED WITH THE │
   │                LU.  IF  THE SSCP-PU  SESSION IS  ACTIVE, THIS  PROCEDURE RETURNS  A │
   │                SENSE CODE OF 8012 IF THE VR USED  BY THE SSCP-PU SESSION IS NOT THE │
   │                SAME AS THE VR TRAVERSED BY THE ACTLU.                │
   │                                                                      │
   │                FOR ACTLU TO  THE BOUNDARY FUNCTION, THIS PROCEDURE  RETURNS A SENSE │
   │                CODE OF  8008 IF THE  SSCP IDENTIFIED BY  THE ORIGIN ADDRESS  IN THE │
   │                ACTLU DOES NOT HAVE AN ACTIVE SSCP-PU SESSION WITH THE PU ASSOCIATED │
   │                WITH THE LU.  IF THE SSCP-PU SESSION IS ACTIVE, A SENSE CODE OF 8012 │
   │                IS RETURNED IF THE VR USED BY THE SSCP-PU SESSION IS NOT THE SAME AS │
   │                THE VR TRAVERSED BY THE ACTLU.                        │
   │                                                                      │
   │                FOR ACTPU TO  THE BOUNDARY FUNCTION, THIS PROCEDURE  RETURNS A SENSE │
   │                CODE OF  8012 IF THE  SSCP IDENTIFIED BY  THE ORIGIN ADDRESS  IN THE │
   │                ACTPU DOES  NOT HAVE AN  ACTIVE SSCP-PU SESSION  WITH THE PU  IN THE │
   │                NODE PROVIDING THE BF SUPPORT.  THIS  PROCEDURE ALSO RETURNS 8012 IF │
   │                THE  VR USED  BY THE  SSCP-PU  SESSION IS  NOT  THE SAME  AS THE  VR │
   │                TRAVERSED BY THE CURRENT ACTPU.                       │
   │                                                                      │
   │     INPUT:     CURRENT MU--ACTPU OR ACTLU                            │
   │                                                                      │
   │     OUTPUT:    APPROPRIATE SENSE CODE--8008, 8012, OR 0 (0 INDICATES NO ERRORS) │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL P PTR;
  DCL SENSE BIT(32);

  SENSE = 0;

  SELECT ANYORDER;
  .
  . WHEN(RQ_CODE = ACTLU & CB_TYPE = HALF_SESS)
  .   DO;
  .   .       /* FIND THE SCB FOR SSCP-PU SESSION FOR PU IN THIS NODE                  */
  .   . FIND P->SCB IN SCB_LIST WHERE(P->SCB.PARTNER_NA = OSAF||OEF &
  .   .       P->SCB.THIS_EA = X'0000' & P->SCB.#FSM_SESS = ACTIVE);
  .   . IF P = NULL THEN                                    /* NO SSCP-PU SESSION   */
  .   .   SENSE = X'8008';                                  /* PU NOT ACTIVE        */
  .   . ELSE
  .   .   IF P->SCB.VRCBPTR ¬= VRCB_PTR THEN
  .   .     SENSE = X'8012';                                /* INVALID VIRTUAL ROUTE */
  .   END;
  .
  . WHEN(RQ_CODE = ACTLU & CB_TYPE = BF_SESS)
  .   DO;
  .   . NRCB_PTR = LOCATE_NODE_RESOURCE(DEF);  /* FIND PU IN PERIPHERAL NODE           */
  .   . IF NRCB_PTR = NULL THEN            /* NO SSCP-PERIPHERAL PU SESSION           */
  .   .   SENSE = X'8008';                                  /* PU NOT ACTIVE        */
  .   . ELSE
  .   .    /* FIND SCB FOR SSCP-PU SESSION FOR PU IN THE NODE PROVIDING BF SUPPORT    */
  .   .   FIND P->SCB IN SCB_LIST WHERE(NRCB.ASSOCIATED_RESOURCE = SCB.THIS_EA &
  .   .        OSAF||OEF = SCB.PARTNER_NA);
  .   . IF P = NULL | P->SCB.VRCBPTR ¬= VRCB_PTR THEN
  .   .   SENSE = X'8012';                 /* SUBAREA PU NOT ACTIVE OR INVALID VR      */
  .   END;
  .
  . WHEN(RQ_CODE = ACTPU & CB_TYPE = BF_SESS)
  .   DO;
  .   .  /* FIND SCB FOR SSCP-PU SESSION FOR PU IN THE NODE PROVIDING BF SUPPORT      */
  .   . FIND P->SCB IN SCB_LIST WHERE(P->SCB.PARTNER_NA = OSAF||OEF &
  .   .       P->SCB.THIS_EA = X'0000' & P->SCB.#FSM_SESS = ACTIVE);
  .   . IF P = NULL | P->SCB.VRCBPTR ¬= VRCB_PTR THEN
  .   .   SENSE = X'8012';                 /* SUBAREA PU NOT ACTIVE OR INVALID VR      */
  .   END;
  .
  END;
  RETURN(SENSE);
END PU_ACTIVE_AND_VR_CHECK;
```

```
SON_VR: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  PERFORMS SESSION OUTAGE NOTIFICATION (SON) BY SENDING DACTCDRM,  │
   │                 DACTLU, DACTPU, OR UNBIND REQUESTS TO NAU'S IN THE SUBAREA OF THIS │
   │                 NODE.  THE SON IS CAUSED BY A VIRTUAL ROUTE (VR) BECOMING │
   │                 INOPERATIVE OR BEING DEACTIVATED (DACTVR_FORCED).  │
   │                 THIS PROCEDURE RECEIVES THE SON SIGNAL WITH THE ADDRESS OF THE │
   │                 AFFECTED VIRTUAL ROUTE CONTROL BLOCK (VRCB) IN THE VRCB_PTR, AND │
   │                 SCANS THE SCB LIST FOR SESSIONS ASSOCIATED WITH THE GIVEN VRCB.  FOR │
   │                 EACH SUCH SESSION THAT IS FOUND, A DEACTIVATION REQUEST (DACTCDRM, │
   │                 DACTLU, DACTPU, OR UNBIND) IS CREATED AND IS SENT TO CSC_MGR.RCV.  │
   │                                                                      │
   │      INPUT:     THE SIGNAL "VRINOP" OR "DACTVR_FORCED" WITH THE VRCB_PTR │
   │                                                                      │
   │      OUTPUT:    DACTCDRM, DACTLU, DACTPU, OR UNBIND (WITH SON_CODE) TO CSC_MGR.RCV │
   │                 (PAGE 13-41)                                         │
   │                                                                      │
   │      NOTE:      IN A PU_T4|5 NODE PROVIDING BOUNDARY FUNCTION SUPPORT, THE │
   │                 DACTLU|DACTPU IS NOT SENT TO THE PU_T1|2 NODE, SO AS NOT TO AFFECT │
   │                 ANY UNDERLYING (LU,LU) SESSIONS.  THE DACTLU|DACTPU WILL BE │
   │                 DISCARDED IN THE BF FSM'S.  THIS IS NECESSARY BECAUSE UNLIKE PU_T4|5 │
   │                 NODES, ALL PU_T1|2 NODES DO NOT SUPPORT THE SON TYPE OF │
   │                 DEACTIVATION.                                        │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL R_CODE BIT(8);
  SCAN SCB_LIST PTR(SCB_PTR);
  . IF SCB.VRCBPTR = VRCB_PTR THEN
  .   DO;
  .   . SELECT ANYORDER(SCB.TYPE_OF_SESSION);
  .   . .
  .   . . WHEN(SSCP_SSCP)
  .   . .   R_CODE = DACTCDRM;
  .   . .
  .   . . WHEN(SSCP_LU)
  .   . .   R_CODE = DACTLU;
  .   . .
  .   . . WHEN(SSCP_PU)
  .   . .   R_CODE = DACTPU;
  .   . .
  .   . . WHEN(LU_LU)
  .   . .   R_CODE = UNBIND;
  .   . .
  .   . END;
  .   . IF INPUT('VRINOP') THEN
  .   .   CALL CREATE_DEACT_RQ(SWITCHED,R_CODE,
  .   .        VRINOP,RECEIVE);                          /* PAGE 13-65      */
  .   . ELSE
  .   .   CALL CREATE_DEACT_RQ(SWITCHED,R_CODE,
  .   .        DACTVR_FORCED,RECEIVE);                   /* PAGE 13-65      */
  .   . SEND MU TO CSC_MGR.RCV;                          /* PAGE 13-41      */
  .   END;
  SCANEND;
  RETURN;
END SON_VR;



SON_RESET: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  CALLS PU_T1_OR_T2_RESET OR PU_T4_OR_T5_RESET UPON RECEIPT OF THE │
   │                 SIGNAL "SSCP_GONE" OR "HIERARCHICAL_RESET".  SEE FIGURE 13-5 FOR A │
   │                 SUMMARY OF SON ACTIVITY.                             │
   │                                                                      │
   │      INPUT:     THE SIGNAL "HIERARCHICAL_RESET" OR "SSCP_GONE" WITH THE SCB_PTR │
   │                 POINTING TO THE SCB REPRESENTING THE SESSION ON WHICH THE │
   │                 DEACTIVATION REQUEST OR RSP(COLD) WAS FLOWING.  THE SIGNAL IS │
   │                 GENERATED BY THE SESS FSM'S.  NRCB_PTR IS SET.       │
   │                                                                      │
   │      OUTPUT:    CALL TO APPROPRIATE PROCEDURE                        │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF NCB.PU_TYPE = (T1 | T2) THEN
    CALL PU_T1_OR_T2_RESET;                             /* PAGE 13-61      */
  ELSE
    CALL PU_T4_OR_T5_RESET;                             /* PAGE 13-62      */
END SON_RESET;
```

```
PU_T1_OR_T2_RESET: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  PERFORMS SESSION OUTAGE NOTIFICATION (SON) BY SENDING DACTPU,│
  │                 DACTLU, OR UNBIND REQUESTS. THE SON IS CAUSED BY THE RESETTING OF
  │                 THE SSCP_PU OR SSCP_LU HIERARCHY RESULTING FROM RSP(DACTLU),
  │                 RSP(DACTPU), RSP(ACTPU,COLD), OR RSP(ACTLU,COLD). SEE FIGURE 13-5
  │                 FOR A SUMMARY OF SON ACTIVITY.                               │
  │                                                                              │
  │      INPUT:     THE SIGNAL "HIERARCHICAL_RESET" OR "SSCP_GONE" WITH THE SCB_PTR
  │                 POINTING TO THE SCB REPRESENTING THE SESSION ON WHICH THE    │
  │                 DEACTIVATION REQUEST OR RSP(COLD) WAS FLOWING. THE SIGNAL IS
  │                 GENERATED BY THE SESS FSM'S.                                 │
  │                                                                              │
  │      OUTPUT:    DACTPU, DACTLU, OR UNBIND, WITH THE SON CODE                  │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL SAVE_SCB_PTR PTR;
  DCL SAVE_BF_PU_ADDRESS BIT(16);
  DCL 1 SAVE_SSCP_ADDRESS,
        2 SUBAREA BIT(32),
        2 ELEMENT BIT(16);
  DCL SAVE_LU_ADDRESS BIT(16);
  DCL SON_SIGNAL CHAR(16);

  SELECT ANYORDER;
                                                                                /*
                  ┌──────────────────────────────────────────────┐
                  │ RSP(ACTPU,COLD) SENT OR DACTPU RECEIVED BY   │
                  │ PU_T2. ACTPU AND DACTPU DON'T FLOW TO A      │
                  │ PU_T1. RESET LU-LU AND SSCP-LU SESSIONS      │
                  │ ONLY.                                        │
                  └──────────────────────────────────────────────┘
                                                                                */
  . WHEN(NRCB.RESOURCE_CATEGORY = PU)
  .   SCAN SCB_LIST PTR(SCB_PTR);
  .   . IF SCB.TYPE_OF_SESSION = LU_LU THEN
  .   .   DO;
  .   .   . CALL CREATE_DEACT_RQ(SWITCHED,UNBIND,
  .   .   .      HIERARCHICAL_RESET,RECEIVE);                  /* PAGE 13-65   */
  .   .   . SEND MU TO CSC_MGR.RCV;                            /* PAGE 13-41   */
  .   .   END;
  .   . ELSE
  .   .   IF SCB.TYPE_OF_SESSION = SSCP_LU THEN
  .   .     DO;
  .   .     . CALL CREATE_DEACT_RQ(SWITCHED,DACTLU,
  .   .     .      HIERARCHICAL_RESET,RECEIVE);                /* PAGE 13-65   */
  .   .     . SEND MU TO CSC_MGR.RCV;                           /* PAGE 13-41   */
  .   .     END;
  .   SCANEND;
                                                                                /*
                  ┌──────────────────────────────────────────────┐
                  │ RSP(ACTLU,COLD) SENT OR DACTLU RECEIVED BY   │
                  │ PERIPHERAL PU                                │
                  └──────────────────────────────────────────────┘
                                                                                */
  . WHEN(NRCB.RESOURCE_CATEGORY = LU)
  .   SCAN SCB_LIST PTR(SCB_PTR);
  .   . IF SCB.TYPE_OF_SESSION = LU_LU &
  .   .      SCB.THIS_EA = NRCB.ELEMENT_ADDRESS THEN
  .   .   DO;
  .   .   . CALL CREATE_DEACT_RQ(SWITCHED,UNBIND,HIERARCHICAL_RESET,RECEIVE);  /* PAGE 13-65 */
  .   .   . SEND MU TO CSC_MGR.RCV;                            /* PAGE 13-41   */
  .   .   END;
  .   SCANEND;
  END;
  RETURN;
END PU_T1_OR_T2_RESET;
```

```
PU_T4_OR_T5_RESET: PROCEDURE;

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  PERFORMS SESSION OUTAGE NOTIFICATION (SON) FOR PU_T4 AND PU_T5 NODES │
│                 BY SENDING DACTPU, DACTLU, OR UNBIND REQUESTS.  THE SON IS CAUSED BY │
│                 THE RESETTING  OF THE  SSCP_PU OR  SSCP_LU HIERARCHY  RESULTING FROM │
│                 RSP(DACTLU), RSP(DACTPU), RSP(ACTPU,COLD),  OR RSP(ACTLU,COLD).  SEE │
│                 FIGURE 13-5 FOR A SUMMARY OF SON ACTIVITY.                │
│                                                                           │
│      INPUT:     THE  SIGNAL "HIERARCHICAL_RESET"  OR  "SSCP_GONE"  WITH THE  SCB_PTR │
│                 POINTING  TO  THE  SCB  REPRESENTING  THE  SESSION  ON  WHICH  THE │
│                 DEACTIVATION  REQUEST  OR  RSP(COLD) WAS  FLOWING.  THE  SIGNAL  IS │
│                 GENERATED BY THE SESS FSM'S.                              │
│                                                                           │
│      OUTPUT:    DACTPU, DACTLU, OR UNBIND, WITH THE SON CODE               │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
DCL SAVE_SCB_PTR PTR;
DCL SAVE_BF_PU_ADDRESS BIT(16);
DCL 1 SAVE_SSCP_ADDRESS,
      2 SUBAREA BIT(32),
      2 ELEMENT BIT(16);
DCL SAVE_LU_ADDRESS BIT(16);
DCL SON_SIGNAL CHAR(16);

SELECT ANYORDER(NRCB.RESOURCE_CATEGORY);
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │ • RSP(ACTPU,COLD)  RECEIVED  BY  BF  FROM  A │
                  │   PU_T2, OR                              │
                  │ • RSP(ACTPU,COLD) SENT  TO PRIMARY  ON BEHALF │
                  │   OF A PU_T1, OR                         │
                  │ • DACTPU ROUTED TO A PU_T2, OR           │
                  │ • RSP(DACTPU) SENT TO PRIMARY  ON BEHALF OF A │
                  │   PU_T1.                                 │
                  │ SEND   UNBIND   TO   ALL   LU-LU   PRIMARY │
                  │ HALF-SESSIONS,  RESET  BF  FOR  SSCP-LU  BY │
                  │ SENDING DACTLU TO SECONDARY.  (DACTLU IS NOT │
                  │ ACTUALLY SENT  BUT DISCARDED IN THE  BF FSM.) │
                  │ FOR BF SUPPORT OF PU_T1'S, SEND UNBIND TO THE │
                  │ SECONDARY HALF-SESSIONS.                 │
                  └──────────────────────────────────────────┘
                                                                        */
. WHEN(BF.PU)
.   DO;
.   . SAVE_BF_PU_ADDRESS = NRCB.ELEMENT_ADDRESS;
.   . SCAN NRCB_LIST PTR(NRCB_PTR);
.   . . IF NRCB.RESOURCE_CATEGORY = BF.LU &  /* FIND LU'S ASSOCIATED WITH THE BF.PU      */
.   . .      NRCB.ASSOCIATED_RESOURCE = SAVE_BF_PU_ADDRESS THEN
.   . .   SCAN SCB_LIST PTR(SCB_PTR);
.   . .   . IF SCB.SCB_TYPE = BF_SESS &
.   . .   .      SCB.THIS_EA = NRCB.ELEMENT_ADDRESS THEN
.   . .   .   IF SCB.TYPE_OF_SESSION = LU_LU THEN
.   . .   .                          /* FIND LU-LU SESSIONS ASSOCIATED WITH LU      */
.   . .   .     DO;
.   . .   .     . CALL CREATE_DEACT_RQ(¬SWITCHED,UNBIND,
.   . .   .     .      HIERARCHICAL_RESET,RECEIVE);          /* PAGE 13-65          */
.   . .   .     . SEND MU TO CSC_MGR.RCV;                    /* PAGE 13-41          */
.   . .   .     . IF CB_TYPE = BF_SESS & NRCB.RESOURCE_TYPE = PU_T1 THEN
.   . .   .     .   DO;
.   . .   .     .   . CALL CREATE_DEACT_RQ(SWITCHED,UNBIND,
.   . .   .     .   .      HIERARCHICAL_RESET,SEND);         /* PAGE 13-65          */
.   . .   .     .   . SEND MU TO CSC_MGR.BF_SEND;            /* PAGE 13-40          */
.   . .   .     .   END;
.   . .   .     END;
.   . .   .   ELSE
.   . .   .     DO;                  /* RESET BF FOR SSCP-LU SESSION                */
.   . .   .     . CALL CREATE_DEACT_RQ(SWITCHED,DACTLU,
.   . .   .     .      HIERARCHICAL_RESET,RECEIVE);          /* PAGE 13-65          */
.   . .   .     . SEND MU TO CSC_MGR.RCV;                    /* PAGE 13-41          */
.   . .   .     END;
.   . .   SCANEND;
.   . SCANEND;
.   END;
                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │ RSP(ACTLU,COLD)  RECEIVED  BY  BF  OR  DACTLU │
                  │ ROUTED TO PU_T1|2 BY BF.  SEND UNBIND TO ALL │
                  │ THE PRIMARY LU-LU HALF-SESSIONS.         │
                  └──────────────────────────────────────────┘
                                                                        */
. WHEN(NRCB.RESOURCE_CATEGORY = BF.LU)
.   SCAN SCB_LIST PTR(SCB_PTR);
.   . IF SCB.TYPE_OF_SESSION = LU_LU &
.   .      SCB.THIS_EA = NRCB.ELEMENT_ADDRESS THEN
.   .   DO;
.   .   . CALL CREATE_DEACT_RQ(¬SWITCHED,UNBIND,
.   .   .      HIERARCHICAL_RESET,RECEIVE);                 /* PAGE 13-65          */
.   .   . SEND MU TO CSC_MGR.RCV;                           /* PAGE 13-41          */
.   .   END;
.   SCANEND;
```

```
                                                                                /*
                    ┌──────────────────────────────────────────────┐
                    │ RSP(ACTPU,COLD) OR DACTPU RECEIVED  BY THE PU │
                    │ IN A SUBAREA NODE                            │
                    └──────────────────────────────────────────────┘
                                                                                */
  . WHEN(NRCB.RESOURCE_CATEGORY = PU)
  .   DO;
  .   . SAVE_SSCP_ADDRESS = SCB.PARTNER_NA;
  .   . SCAN SCB_LIST PTR(SCB_PTR);
  .   . . SELECT ANYORDER;
  .   . . . WHEN(SCB.TYPE_OF_SESSION = SSCP_LU &
  .   . . .      SCB.PARTNER_NA = SAVE_SSCP_ADDRESS &
  .   . . .      INPUT('HIERARCHICAL_RESET'))
  .   . . .   DO;
  .   . . .   . SAVE_LU_ADDRESS = SCB.THIS_EA;
  .   . . .   . SAVE_SCB_PTR = SCB_PTR;    /* SAVE THE OUTER SCAN POINTER          */
  .   . . .   . SCAN SCB_LIST PTR(SCB_PTR);
  .   . . .   . . IF SCB.TYPE_OF_SESSION = LU_LU &
  .   . . .   . .      SCB.THIS_EA = SAVE_LU_ADDRESS THEN
  .   . . .   . .   DO;
  .   . . .   . .   . CALL CREATE_DEACT_RQ(¬SWITCHED,UNBIND,
  .   . . .   . .   .      HIERARCHICAL_RESET,SEND);          /* PAGE 13-65      */
  .   . . .   . .   . SEND MU TO CSC_MGR.SEND;                /* PAGE 13-35      */
  .   . . .   . .   . CALL CREATE_DEACT_RQ(SWITCHED,UNBIND,
  .   . . .   . .   .      HIERARCHICAL_RESET,RECEIVE);       /* PAGE 13-65      */
  .   . . .   . .   . SEND MU TO CSC_MGR.RCV;                 /* PAGE 13-41      */
  .   . . .   . .   END;
  .   . . .   . SCANEND;
  .   . . .   . SCB_PTR = SAVE_SCB_PTR;                      /* RESTORE POINTER  */
  .   . . .   .                                              /* SEND DACTLU TO LU*/
  .   . . .   . CALL CREATE_DEACT_RQ(SWITCHED,DACTLU,
  .   . . .   .      HIERARCHICAL_RESET,RECEIVE);            /* PAGE 13-65      */
  .   . . .   . SEND MU TO CSC_MGR.RCV;                      /* PAGE 13-41      */
  .   . . .   END;
  .   . . . WHEN(SCB.SCB_TYPE = BF_SESS & SCB.TYPE_OF_SESSION = SSCP_PU &
  .   . . .      SCB.PARTNER_NA = SAVE_SSCP_ADDRESS &
  .   . . .      INPUT('HIERARCHICAL_RESET'))
  .   . . .   DO;
  .   . . .   . CALL CREATE_DEACT_RQ(SWITCHED,DACTPU,
  .   . . .   .      HIERARCHICAL_RESET,RECEIVE);            /* PAGE 13-65      */
  .   . . .   . SEND MU TO CSC_MGR.RCV;                      /* PAGE 13-41      */
  .   . . .   END;
  .   . . . WHEN(SCB.TYPE_OF_SESSION = SSCP_LU &
  .   . . .      SCB.PARTNER_NA = SAVE_SSCP_ADDRESS &
  .   . . .      INPUT('SSCP_GONE'))
  .   . . .   CALL SCB_DISCARD; /* PAGE 13-88, RESETS THE SSCP-LU OR SSCP-BF.LU SESSION */
  .   . . . WHEN(SCB.TYPE_OF_SESSION = SSCP_PU &
  .   . . .      SCB.SCB_TYPE = BF_SESS &
  .   . . .      SCB.PARTNER_NA = SAVE_SSCP_ADDRESS &
  .   . . .      INPUT('SSCP_GONE'))
  .   . . .   CALL SCB_DISCARD;            /* PAGE 13-88, RESETS THE SSCP-BF.PU SESSION */
  .   . . END;
  .   . SCANEND;
  .   END;
                                                                                /*
                    ┌──────────────────────────────────────────────┐
                    │ RSP(ACTLU,COLD) OR  DACTLU RECEIVED BY  AN LU │
                    │ IN A SUBAREA NODE                            │
                    └──────────────────────────────────────────────┘
                                                                                */
  . WHEN(NRCB.RESOURCE_CATEGORY = LU)
  .   SCAN SCB_LIST PTR(SCB_PTR);
  .   . IF SCB.TYPE_OF_SESSION = LU_LU &
  .   .      SCB.THIS_EA = NRCB.ELEMENT_ADDRESS THEN
  .   .   DO;
  .   .   . CALL CREATE_DEACT_RQ(¬SWITCHED,UNBIND,
  .   .   .      HIERARCHICAL_RESET,SEND);                   /* PAGE 13-65      */
  .   .   . SEND MU TO CSC_MGR.SEND;                         /* PAGE 13-35      */
  .   .   . CALL CREATE_DEACT_RQ(SWITCHED,UNBIND,
  .   .   .      HIERARCHICAL_RESET,RECEIVE);                /* PAGE 13-65      */
  .   .   . SEND MU TO CSC_MGR.RCV;                          /* PAGE 13-41      */
  .   .   END;
  .   SCANEND;
  END;
  RETURN;
END PU_T4_OR_T5_RESET;
```

```
SON_REX_INOP: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  PERFORMS  SESSION  OUTAGE  NOTIFICATION  (SON)  BY  SENDING  UNBIND  │
   │                 REQUESTS TO LU'S.  THE SON IS  CAUSED BY AN INOPERATIVE CONDITION ON  │
   │                 THE    ROUTE    EXTENSION.     THE    SON    SIGNAL   ORGINATES    AT  │
   │                 CNTL.ALS_SEC_SUBTREE_RESET OF  THE PU.SVC_MGR.NS  (CHAPTER 11)  WITH  │
   │                 LSCB.EA CONTAINING THE  ELEMENT ADDRESS OF THE  INOPERATIVE ADJACENT  │
   │                 LINK STATION.  WHEN IN A PU_T4|5  NODE, FOR EACH SCB_LIST ENTRY WITH  │
   │                 A MATCHING ALS_EA, THE PROCEDURE PERFORMS THE FOLLOWING:              │
   │                                                                                       │
   │                    • IF  THE SCB REPRESENTS AN  SSCP-BASED SESSION, IT  DISCARDS THE  │
   │                      BFSCB FOR THAT SESSION.                                          │
   │                    • IF  THE SCB REPRESENTS AN  (LU,LU) SESSION, IT SENDS  AN UNBIND  │
   │                      REQUEST TO  THE LU  IDENTIFIED BY  SCB.PARTNER_NA IN  THE GIVEN  │
   │                      SCB.  THE TYPE CODE IN UNBIND IS SET TO REX_INOP (X'08').        │
   │                                                                                       │
   │                 WHEN IN  A PU_T1|2  NODE, FOR  EACH SCB_LIST  ENTRY WITH  A MATCHING  │
   │                 ALS_EA,  THE  PROCEDURE  SENDS  AN UNBIND,  DACTPU,  OR  DACTLU  (AS  │
   │                 APPROPRIATE) TO EACH  HALF-SESSION REPRESENTED BY A  SESSION CONTROL  │
   │                 BLOCK.                                                                │
   │                                                                                       │
   │      INPUT:     ADDRESS OF THE ALS IN LSCB.EA                                         │
   │                                                                                       │
   │      OUTPUT:    UNBIND  (WITH  SON CODE)  TO  CSC_MGR.RCV  (PAGE 13-41),  WHERE  THE  │
   │                 "DESTINATION NETWORK ADDRESS"  OF THE UNBIND IS  THE PARTNER_NA FROM  │
   │                 THE SCB.                                                              │
   │                                                                                       │
   │      NOTE:      INOP PROCESSING BY  THE SSCP RESETS THE PRIMARY  HALF-SESSION OF THE  │
   │                 SSCP-PU AND SSCP-LU SESSIONS.                                         │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RQ_CD BIT(8);
  SELECT ANYORDER;
  . WHEN(NCB.PU_TYPE = (T4 | T5))
  .   SCAN SCB_LIST PTR(SCB_PTR);
  .   . NRCB_PTR = FIND_ALS_FOR_RESOURCE(SCB.THIS_EA);          /* APPENDIX B          */
  .   . IF LSCB.EA = NRCB.ELEMENT_ADDRESS THEN
  .   .   DO;
  .   .   . IF SCB.TYPE_OF_SESSION = LU_LU THEN
  .   .   .   DO;
  .   .   .   . CALL CREATE_DEACT_RQ(¬SWITCHED,UNBIND,REX_INOP,RECEIVE);  /* PAGE 13-65  */
  .   .   .   . SEND MU TO CSC_MGR.BF_RCV;                     /* PAGE 13-46          */
  .   .   .   END;
  .   .   . ELSE
  .   .   .   CALL SCB_DISCARD;          /* RESETS THE BF REPRESENTATION OF THE SSCP-PU */
  .   .   .                              /* AND SSCP-LU SESSIONS, PAGE 13-88            */
  .   .   END;
  .   SCANEND;
  . WHEN(NCB.PU_TYPE = (T1 | T2))
  .   DO;
  .   . SCAN SCB_LIST PTR(SCB_PTR);
  .   . . SELECT ANYORDER(SCB.TYPE_OF_SESSION);
  .   . . . WHEN(SSCP_PU)
  .   . . .   RQ_CD = DACTPU;
  .   . . . WHEN(SSCP_LU)
  .   . . .   RQ_CD = DACTLU;
  .   . . . WHEN(LU_LU)
  .   . . .   RQ_CD = UNBIND;
  .   . . END;
  .   . . CALL CREATE_DEACT_RQ(SWITCHED,RQ_CD,REX_INOP,RECEIVE);  /* PAGE 13-65       */
  .   . . SEND MU TO CSC_MGR.RCV;                             /* PAGE 13-41          */
  .   . SCANEND;
  .   END;
  END;
  RETURN;
END SON_REX_INOP;
```

```
CREATE_DEACT_RQ: PROCEDURE(ADDR_SWITCH,REQUEST_CODE,SON_CODE,DIRECTION);

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  CREATES A DACTCDRM, DACTLU, DACTPU, OR UNBIND        │
  │                                                                      │
  │      INPUT:     ADDR_SWITCH (¬SWITCHED INDICATES THE MU WILL  BE BUILT TO LOOK AS IF
  │                 IT ORIGINATED IN THIS  NODE--OR IN THE SECONDARY IF THIS  IS THE BF;
  │                 SWITCHED INDICATES THE MU WILL BE BUILT  TO LOOK AS IF IT ORIGINATED
  │                 IN THE PARTNER NODE), REQUEST CODE, AND SON_CODE      │
  │                                                                      │
  │      OUTPUT:    THE RQ(DACTCDRM | DACTLU | DACTPU | UNBIND) WITH THE SON_CAUSE CODE
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL ADDR_SWITCH BIT(1);
  DCL REQUEST_CODE BIT(8);
  DCL SON_CODE BIT(8);
  DCL DIRECTION BIT(1);

  CREATE MU;
  RRI = REQUEST;
  RQ_CODE = REQUEST_CODE;

  SELECT ANYORDER(NCB.PU_TYPE);
  . WHEN(T1)
  .   DO;
  .   . FID = FID3
  .   .      LSID = SCB.LOCAL_SESSION_ID;
  .   END;
  . WHEN(T2)
  .   DO;
  .   . FID = FID2;
  .   . IF ADDR_SWITCH =  ¬SWITCHED THEN
  .   .   DO;
  .   .   . OAFPRIME = SCB.THIS_ID;
  .   .   . DAFPRIME = SCB.PARTNER_ID;
  .   .   END;
  .   . ELSE
  .   .   DO;
  .   .   . OAFPRIME = SCB.PARTNER_ID;
  .   .   . DAFPRIME = SCB.THIS_ID;
  .   .   END;
  .   END;
  . WHEN(T4 | T5)
  .   DO;
  .   . FID = FID4;
  .   . IF ADDR_SWITCH =  ¬SWITCHED THEN
  .   .   DO;
  .   .   . OSAF = SCB.THIS_SA;
  .   .   . OEF = SCB.THIS_EA;
  .   .   . DSAF = SCB.PARTNER_SA;
  .   .   . DEF = SCB.PARTNER_EA;
  .   .   END;
  .   . ELSE
  .   .   DO;
  .   .   . OSAF = SCB.PARTNER_SA;
  .   .   . OEF = SCB.PARTNER_EA;
  .   .   . DSAF = SCB.THIS_SA;
  .   .   . DEF = SCB.THIS_EA;
  .   .   END;
  .   END;
  END;
  MUCB.DIRECTION = DIRECTION;

  SELECT ANYORDER(RQ_CODE);
  .
  . WHEN(UNBIND)
  .   UNBIND_RQ.SON_CAUSE = SON_CODE;
  .
  . WHEN(DACTLU)
  .   DACTLU_RQ.SON_CAUSE = SON_CODE;
  .
  . WHEN(DACTPU)
  .   DACTPU_RQ.SON_CAUSE = SON_CODE;
  .
  . WHEN(DACTCDRM)
  .   DACTCDRM_RQ.SON_CAUSE = SON_CODE;
  .
  END;
  RETURN;
END CREATE_DEACT_RQ;
```

```
SESSACT.REQUEST: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  SAVES THE PARAMETERS CONTAINED IN THE ACTIVATION REQUEST IN THE SCB.
  │
  │      INPUT:     THE CURRENT ACTIVATION REQUEST
  │
  │      OUTPUT:    NONE
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF CB_TYPE = HALF_SESS THEN
    DO;
    . SCB.SCB_TYPE = HALF_SESS;
    . IF MUCB.DIRECTION = SEND THEN
    .   SCB.HALF_SESSION = PRI;
    . ELSE
    .   SCB.HALF_SESSION = SEC;
    END;
  ELSE
    SCB.SCB_TYPE = BF_SESS;
  SELECT ANYORDER(RQ_CODE);
  .
  . WHEN(ACTCDRM)
  .   DO;
  .   . SCB.OPTIONS = 0;
  .   . SCB.TYPE_OF_SESSION = SSCP_SSCP;
  .   . SCB.FM_PROFILE = ACTCDRM_RQ.FM_PROFILE;
  .   . SCB.TS_PROFILE = ACTCDRM_RQ.TS_PROFILE;
  .   . SCB.PRI_RCV_PACING_CNT = ACTCDRM_RQ.PRI_RCV_PAC_CNT;
  .   . SCB.SEC_SEND_PACING_CNT = ACTCDRM_RQ.PRI_RCV_PAC_CNT;
  .   . IF MUCB.DIRECTION = SEND THEN
  .   .   SCB.THIS_HALF_SESSION_SSCP_ID = ACTCDRM_RQ.SSCP_ID;
  .   . ELSE
  .   .   DO;
  .   .   . SCB.PARTNER_HALF_SESSION_SSCP_ID = ACTCDRM_RQ.SSCP_ID;
  .   .   . SCB.ACT_RQ_RSP_SEQ_ID = UPM_GET_SEQ_ID;          /* PAGE 13-88      */
  .   .   END;
  .   . #SVC_MGR = SSCP.SVC_MGR.CS.RCV;                     /* CHAPTER 7       */
  .   END;
  .
  . WHEN(ACTLU)
  .   DO;
  .   . SCB.TYPE_OF_SESSION = SSCP_LU;
  .   . SCB.TS_PROFILE = PAD_4_BITS||ACTLU_RQ.TS_PROFILE;
  .   .                          /* MAKE EIGHT BITS FOR ASSIGNMENT STATEMENT  */
  .   . IF CB_TYPE = HALF_SESS THEN
  .   .   DO;
  .   .   . SCB.FM_PROFILE = PAD_4_BITS||ACTLU_RQ.FM_PROFILE;
  .   .   . IF MUCB.DIRECTION = SEND THEN
  .   .   .   #SVC_MGR = SSCP.SVC_MGR.CS.RCV;              /* CHAPTER 7       */
  .   .   . ELSE
  .   .   .   #SVC_MGR = LU.SVC_MGR.SS.RCV;                /* CHAPTER 8       */
  .   .   END;
  .   . ELSE
  .   .   #SVC_MGR = BF.LU.SVC_MGR;
  .   END;
```

```
.
. WHEN(ACTPU)
.   DO;
.   . IF NCB.PU_TYPE = (T1 | T2) & (DISPATCHED_BY(PUCP.SVC_MGR) |
.   .       MUCB.PU_BASED_SESSION = B'01') THEN
.   .   SCB.TYPE_OF_SESSION = PUCP_PU;
.   . ELSE
.   .   SCB.TYPE_OF_SESSION = SSCP_PU;
.   . SCB.TS_PROFILE = PAD_4_BITS||ACTPU_RQ.TS_PROFILE;
.   .                                   /* MAKE EIGHT BITS FOR ASSIGNMENT STATEMENT      */
.   . IF CB_TYPE = HALF_SESS THEN
.   .   DO;
.   .   . SCB.FM_PROFILE = PAD_4_BITS||ACTPU_RQ.FM_PROFILE;
.   .   . IF MUCB.DIRECTION = SEND THEN
.   .   .   DO;
.   .   .   . IF DISPATCHED_BY(PUCP.SVC_MGR) THEN
.   .   .   .   #SVC_MGR = PUCP.SVC_MGR;
.   .   .   . ELSE
.   .   .   .   #SVC_MGR = SSCP.SVC_MGR.CS.RCV;          /* CHAPTER 7                     */
.   .   .   END;
.   .   . ELSE
.   .   .   DO;
.   .   .   . #SVC_MGR = PU.SVC_MGR.NS.RCV;              /* CHAPTER 11                    */
.   .   .   . IF NCB.PU_TYPE = (T4 | T5) THEN /* SESSION OVERRIDE ONLY FOR PU_T4|5.      */
.   .   .   .   SCB.ACT_RQ_RSP_SEQ_ID = UPM_GET_SEQ_ID;  /* PAGE 13-88                    */
.   .   .   END;
.   .   END;
.   . ELSE
.   .   #SVC_MGR = BF.PU.SVC_MGR;
.   END;
.
. WHEN(BIND)
.   DO;
.   . SCB.TYPE_OF_SESSION = LU_LU;
.   . SCB.TS_PROFILE = BIND_RQ.TS_PROFILE;
.   . CALL TS_PROFILE_PROC;                             /* PAGE 13-70                    */
.   . IF CB_TYPE = HALF_SESS THEN
.   .   DO;
.   .   . SCB.FM_PROFILE = BIND_RQ.FM_PROFILE;
.   .   . SCB.PS_PROFILE = BIND_RQ.PS_PROFILE;
.   .   . SCB.PS_USAGE = BIND_RQ.PS_USAGE;
.   .   . CALL FM_PROFILE_PROC;                         /* PAGE 13-70                    */
.   .   . CALL UPM_PS_PROFILE;          /* SAVE PS PROFILE AND PS USAGE INFORMATION      */
.   .   . #SVC_MGR = LU.SVC_MGR.SS.RCV;                 /* CHAPTER 8                     */
.   .   . IF MUCB.DIRECTION = RECEIVE &
.   .   .       SCB.SC_CRV = ALLOWED THEN
.   .   .   SCB.SESS_CRYPTOGRAPHY_KEY = BIND_RQ.SESS_CRYPTOGRAPHY_KEY;
.   .   END;
.   . ELSE
.   .   #SVC_MGR = BF.LU.SVC_MGR;
.   END;
  END;
  RETURN;
END SESSACT.REQUEST;
```

```
SESSACT.RESPONSE: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE INITIALIZES  THE SESSION CONTROL BLOCK  BY SAVING THE      │
  │                 PARAMETERS THAT  ARE CONTAINED  IN THE  ACTIVATION RESPONSE,  IF THE     │
  │                 ACTIVATION RESPONSE  CARRIES PARAMETERS.  THIS PROCEDURE  ALSO CALLS     │
  │                 INITIALIZATION ROUTINES  IN TC  (CHAPTER 4) AND  DFC (CHAPTER  5) TO     │
  │                 COMPLETE THE INITIALIZATION OF THE  SESSION PARAMETERS (E.G., PACING      │
  │                 COUNTS)  IN  THE  SESSION  CONTROL  BLOCK.  NO  INITIALIZATION  IS       │
  │                 NECESSARY FOR SNS  SINCE THE FSM'S ASSOCIATED  WITH THE HALF-SESSION     │
  │                 ARE  DEFINED TO  BE  IN THE  RESET STATE  WHEN  THE HALF-SESSION  IS     │
  │                 ACTIVATED.                                                               │
  │                                                                                          │
  │      INPUT:     THE CURRENT ACTIVATION RESPONSE                                          │
  │                                                                                          │
  │      OUTPUT:    NONE                                                                     │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL PROFILE_UPDATE BIT(1);
  PROFILE_UPDATE = ON;
  SELECT ANYORDER(RQ_CODE);
  .
  . WHEN(ACTCDRM)
  .   DO;
  .   . SCB.FM_PROFILE = ACTCDRM_RSP.FM_PROFILE;
  .   . SCB.TS_PROFILE = ACTCDRM_RSP.TS_PROFILE;
  .   . SCB.SEC_RCV_PACING_CNT = ACTCDRM_RSP.SEC_RCV_PAC_CNT;
  .   . SCB.PRI_SEND_PACING_CNT = ACTCDRM_RSP.SEC_RCV_PAC_CNT;
  .   . IF MUCB.DIRECTION = SEND THEN
  .   .   SCB.THIS_HALF_SESSION_SSCP_ID = ACTCDRM_RSP.SSCP_ID;
  .   . ELSE
  .   .   SCB.PARTNER_HALF_SESSION_SSCP_ID = ACTCDRM_RSP.SSCP_ID;
  .   . SCB.ACT_RQ_RSP_SEQ_ID = UPM_GET_SEQ_ID;                  /* PAGE 13-88      */
  .   END;
  .
  . WHEN(ACTLU)
  .   DO;
  .   . IF DCF = RSP_OF_LENGTH_TWO THEN    /*NO MAX RU SIZE SPECIFIED                 */
  .   .   DO;
  .   .   . SCB.PRI_SEND_MAX_RU_SIZE = X'85'; /* 256 BYTES ENCODED = X'85'          */
  .   .   . SCB.SEC_SEND_MAX_RU_SIZE = X'85';                 /* APPENDIX E         */
  .   .   END;
  .   . ELSE
  .   .   DO;
  .   .   . SCB.PRI_SEND_MAX_RU_SIZE = ACTLU_RSP.MAX_RU_SIZE;
  .   .   . SCB.SEC_SEND_MAX_RU_SIZE = ACTLU_RSP.MAX_RU_SIZE;
  .   .   . SCB.TS_PROFILE = PAD_4_BITS||ACTLU_RSP.TS_PROFILE;
  .   .   .                          /* MAKE EIGHT  BITS FOR ASSIGNMENT STATEMENT  */
  .   .   . IF CB_TYPE = HALF_SESS THEN
  .   .   .   SCB.FM_PROFILE = PAD_4_BITS||ACTLU_RSP.FM_PROFILE;
  .   .   END;
  .   END;
  .
  . WHEN(ACTPU)
  .   DO;
  .   . IF CB_TYPE = HALF_SESS THEN
  .   .   DO;
  .   .   . IF SCB.HALF_SESSION = PRI THEN
  .   .   .   SCB.REQUEST_RECEIVE = NOT_ALLOWED;
  .   .   . SELECT ANYORDER(ACTPU_RSP.FORMAT);
  .   .   . . WHEN(0)
  .   .   . .   IF NCB.PU_TYPE = (T4 | T5) &
  .   .   . .        OEF = 0 THEN          /* PU_T4|5 ALLOWS FMD REQUESTS          */
  .   .   . .     SCB.REQUEST_RECEIVE = ALLOWED;
  .   .   . .
  .   .   . . WHEN(1)
  .   .   . .   SCB.REQUEST_RECEIVE = FORMAT_1_ACTPU_RSP.REQUEST_RCV;
  .   .   . .
  .   .   . . WHEN(2)
  .   .   . .   SCB.REQUEST_RECEIVE = FORMAT_2_ACTPU_RSP.REQUEST_RCV;
  .   .   . .
  .   .   . . WHEN(3)
  .   .   . .   SCB.REQUEST_RECEIVE = ALLOWED;
  .   .   . END;
  .   .   END;
  .   END;
```

```
.
. WHEN(BIND)
.   DO;
.   . PROFILE_UPDATE = OFF;
.   . IF DCF ¬= RSP_OF_LENGTH_ONE & BIND_RSP.TYPE = NEGOTIABLE THEN
.   .   DO;
.   .   . PROFILE_UPDATE = ON;
.   .   . SCB.OPTIONS = 0;
.   .   . SCB.TS_PROFILE = BIND_RSP.TS_PROFILE;
.   .   . IF CB_TYPE = HALF_SESS THEN
.   .   .   DO;
.   .   .   . SCB.FM_PROFILE = BIND_RSP.FM_PROFILE;
.   .   .   . SCB.PS_PROFILE = BIND_RSP.PS_PROFILE;
.   .   .   . SCB.PS_USAGE = BIND_RSP.PS_USAGE;
.   .   .   . SCB.CRYPTOGRAPHY_SESSION_LEVEL = BIND_RSP.CRYPTOGRAPHY_SESSION_LEVEL;
.   .   .   END;
.   .   END;
.   END;
END;
IF PROFILE_UPDATE = ON THEN              /* OFF FOR NONNEGOTIABLE BIND                  */
  CALL TS_PROFILE_PROC;                                    /* PAGE 13-70              */
IF CB_TYPE = HALF_SESS THEN
  DO;
  . IF PROFILE_UPDATE = ON THEN          /* OFF FOR NONNEGOTIABLE BIND                  */
  .   CALL FM_PROFILE_PROC;                                /* PAGE 13-70              */
  . CALL SESSACT.DFC_INITIALIZE;                           /* CHAPTER 5               */
  . CALL SESSACT.DFC_RESET;                                /* CHAPTER 5               */
  . CALL SESSACT.TC_INITIALIZE;                            /* CHAPTER 4               */
  . CALL SESSACT.TC_RESET;                                 /* CHAPTER 4               */
  END;
ELSE                                                       /* CB_TYPE = BF_SESS       */
  DO;
  . CALL BF.SESSACT.TC.INITIALIZE;                         /* CHAPTER 4               */
  . CALL BF.TC.RESET;                                      /* CHAPTER 4               */
  END;
RETURN;
END SESSACT.RESPONSE;
```

```
FM_PROFILE_PROC: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE ROUTES TO THE PROPER FM PROFILE ROUTINES.     │
  │                                                                              │
  │      INPUT:     THE CURRENT MU                                               │
  │                                                                              │
  │      OUTPUT:    NONE                                                         │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SELECT ANYORDER(SCB.FM_PROFILE);
  .
  . WHEN(PROFILE_0)                     /* MAY BE USED FOR SSCP-PU OR SSCP-LU SESSIONS   */
  .   CALL FM_PROFILE_0;                                    /* PAGE 13-71              */
  .
  . WHEN(PROFILE_2)                     /* MAY BE USED FOR LU-LU SESSIONS               */
  .   CALL FM_PROFILE_2;                                    /* PAGE 13-72              */
  .
  . WHEN(PROFILE_3)                     /* MAY BE USED FOR LU-LU SESSIONS               */
  .   CALL FM_PROFILE_3;                                    /* PAGE 13-73              */
  .
  . WHEN(PROFILE_4)                     /* MAY BE USED FOR LU-LU SESSIONS               */
  .   CALL FM_PROFILE_4;                                    /* PAGE 13-74              */
  .
  . WHEN(PROFILE_5)                     /* MAY BE USED FOR SSCP-PU SESSIONS             */
  .   CALL FM_PROFILE_5;                                    /* PAGE 13-75              */
  .
  . WHEN(PROFILE_6)                     /* MAY BE USED FOR SSCP-LU SESSIONS             */
  .   CALL FM_PROFILE_6;                                    /* PAGE 13-75              */
  .
  . WHEN(PROFILE_7)                     /* MAY BE USED FOR LU-LU SESSIONS               */
  .   CALL FM_PROFILE_7;                                    /* PAGE 13-76              */
  .
  . WHEN(PROFILE_17)                    /* MAY BE USED FOR SSCP-SSCP SESSIONS           */
  .   CALL FM_PROFILE_17;                                   /* PAGE 13-77              */
  .
  . WHEN(PROFILE_18)                    /* MAY BE USED FOR LU-LU SESSIONS               */
  .   CALL FM_PROFILE_18;                                   /* PAGE 13-78              */
  END;
  RETURN;
END FM_PROFILE_PROC;



TS_PROFILE_PROC: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS  PROCEDURE ROUTES  TO THE  PROPER  TS PROFILE  ROUTINES, OR  BF  │
  │                 PARAMETER ROUTINE THAT SAVES THE  PARAMETERS NEEDED FOR TRANSMISSION  │
  │                 CONTROL.                                                     │
  │                                                                              │
  │      INPUT:     THE CURRENT MU                                               │
  │                                                                              │
  │      OUTPUT:    NONE                                                         │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF CB_TYPE = BF_SESS THEN                                 /* USED FOR BF SUPPORT  */
    CALL BF_TS_PARAMETERS;                                  /*  PAGE 13-85          */
  ELSE

    SELECT ANYORDER(SCB.TS_PROFILE);
    .
    . WHEN(PROFILE_1)                   /* MAY BE USED FOR SSCP-PU OR SSCP-LU SESSIONS   */
    .   CALL TS_PROFILE_1;                                  /*  PAGE 13-80             */
    .
    . WHEN(PROFILE_2)                   /* MAY BE USED FOR LU-LU SESSIONS               */
    .   CALL TS_PROFILE_2;                                  /*  PAGE 13-72             */
    .
    . WHEN(PROFILE_3)                   /* MAY BE USED FOR LU-LU SESSIONS               */
    .   CALL TS_PROFILE_3;                                  /*  PAGE 13-81             */
    .
    . WHEN(PROFILE_4)                   /* MAY BE USED FOR LU-LU SESSIONS               */
    .   CALL TS_PROFILE_4;                                  /*  PAGE 13-82             */
    .
    . WHEN(PROFILE_5)                   /* MAY BE USED FOR SSCP-PU SESSIONS             */
    .   CALL TS_PROFILE_5;                                  /*  PAGE 13-82             */
    .
    . WHEN(PROFILE_7)                   /* MAY BE USED FOR LU-LU SESSIONS               */
    .   CALL TS_PROFILE_7;                                  /*  PAGE 13-83             */
    .
    . WHEN(PROFILE_17)                  /* MAY BE USED FOR SSCP-SSCP SESSIONS           */
    .   CALL TS_PROFILE_17;                                 /*  PAGE 13-83             */
    END;
  RETURN;
END TS_PROFILE_PROC;
```

```
FM_PROFILE_0: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  FILLS IN PARAMETERS IN THE SCB FOR FM PROFILE 0.             │
  │                                                                              │
  │      INPUT:     NONE                                                         │
  │                                                                              │
  │      OUTPUT:    NONE                                                         │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SCB.PRI_CHAIN_USE         = SINGLE;                    /* PRIMARY USAGE      */
  SCB.PRI_RQ_MODE           = IMMEDIATE;
  SCB.PRI_NO_RSP_CHAIN      = NOT_ALLOWED;
  SCB.PRI_EXCP_RSP_CHAIN    = NOT_ALLOWED;
  SCB.PRI_DEF_RSP_CHAIN     = ALLOWED;
  SCB.PRI_COMPR_IND         = NO_COMPRESSION;
  SCB.PRI_EB_IND            = MAY_NOT_SEND;
  SCB.SEC_CHAIN_USE         = SINGLE;                    /* SECONDARY USAGE    */
  SCB.SEC_RQ_MODE           = IMMEDIATE;
  SCB.SEC_NO_RSP_CHAIN      = NOT_ALLOWED;
  SCB.SEC_EXCP_RSP_CHAIN    = NOT_ALLOWED;
  SCB.SEC_DEF_RSP_CHAIN     = ALLOWED;
  SCB.SEC_COMPR_IND         = NO_COMPRESSION;
  SCB.SEC_EB_IND            = MAY_NOT_SEND;
  SCB.FM_HDR_USAGE          = NO_FM_HEADERS;             /* COMMON USAGE       */
  SCB.BRACKETS_RESET_STATE  = BRACKETS_NOT_USED;
  SCB.ALT_CODE              = NOT_USED;
  SCB.SEND_RCV_MODE         = HDX_CONTENTION;
  SCB.RECOVERY_RESP         = PRI;
  SCB.CONT_WIN              = SEC;
  SCB.PRI_RSP_MODE          = IMMEDIATE;
  SCB.SEC_RSP_MODE          = IMMEDIATE;
  RETURN;
END FM_PROFILE_0;
```

```
FM_PROFILE_2: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  FILLS IN PARAMETERS IN THE SCB FOR FM PROFILE 2.         │
   │                                                                          │
   │      INPUT:     NONE                                                     │
   │                                                                          │
   │      OUTPUT:    NONE                                                     │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      THE FOLLOWING PARAMETERS ARE FILLED IN FROM THE FM USAGE FIELD IN THE BIND RU.  │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF RRI = RQ THEN
    DO;
    . SCB.PRI_RQ_MODE               = BIND_RQ.PRI_RQ_MODE;
    . SCB.BRACKETS_RESET_STATE      = BIND_RQ.BRACKETS_USAGE;
    . SCB.ALT_CODE                  = BIND_RQ.ALT_CODE;
    . SELECT ANYORDER(BIND_RQ.PRI_CHAIN_RSP);
    . . WHEN(EXCP_RESPONSE)
    . .   DO;
    . .   . SCB.PRI_NO_RSP_CHAIN    = NOT_ALLOWED;
    . .   . SCB.PRI_EXCP_RSP_CHAIN  = ALLOWED;
    . .   . SCB.PRI_DEF_RSP_CHAIN   = NOT_ALLOWED;
    . .   END;
    . . WHEN(DEF_RESPONSE)
    . .   DO;
    . .   . SCB.PRI_NO_RSP_CHAIN    = NOT_ALLOWED;
    . .   . SCB.PRI_EXCP_RSP_CHAIN  = NOT_ALLOWED;
    . .   . SCB.PRI_DEF_RSP_CHAIN   = ALLOWED;
    . .   END;
    . . WHEN(DEF_OR_EXCP_RESPONSE)
    . .   DO;
    . .   . SCB.PRI_NO_RSP_CHAIN    = NOT_ALLOWED;
    . .   . SCB.PRI_EXCP_RSP_CHAIN  = ALLOWED;
    . .   . SCB.PRI_DEF_RSP_CHAIN   = ALLOWED;
    . .   END;
    . .                                     /* NO RSP MAY NOT BE SPECIFIED BY FM PROFILE 2    */
    . END;
    END;

  ELSE
    DO;
    . SCB.PRI_RQ_MODE               = BIND_RSP.PRI_RQ_MODE;
    . SCB.BRACKETS_RESET_STATE      = BIND_RSP.BRACKETS_USAGE;
    . SCB.ALT_CODE                  = BIND_RSP.ALT_CODE;
    END;
                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      THE FOLLOWING SESSION RULES ARE SPECIFIED IN THE DEFINITION OF FM PROFILE 2.  │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SCB.SEC_EB_IND                    = MAY_NOT_SEND;
  SCB.PRI_COMPR_IND                 = NO_COMPRESSION;
  IF SCB.BRACKETS_RESET_STATE = BETB THEN
    SCB.PRI_EB_IND                  = MAY_SEND;
  ELSE
    SCB.PRI_EB_IND                  = MAY_NOT_SEND;
  SCB.SEC_CHAIN_USE                 = SINGLE;
  SCB.SEC_RQ_MODE                   = DELAYED;
  SCB.SEC_NO_RSP_CHAIN              = ALLOWED;
  SCB.SEC_EXCP_RSP_CHAIN            = NOT_ALLOWED;
  SCB.SEC_DEF_RSP_CHAIN             = NOT_ALLOWED;
  SCB.SEC_COMPR_IND                 = NO_COMPRESSION;
  SCB.SEC_EB_IND                    = MAY_NOT_SEND;
  SCB.FM_HDR_USAGE                  = NO_FM_HEADERS;
  SCB.BRKT_TERM_RULE                = UNCONDITIONAL;
  SCB.SEND_RCV_MODE                 = FULL_DUPLEX;
  SCB.RECOVERY_RESP                 = PRI;
  SCB.CONT_WIN                      = SEC;
  SCB.SEC_RSP_MODE                  = IMMEDIATE;
  RETURN;
END FM_PROFILE_2;
```

```
FM_PROFILE_3: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  FILLS IN PARAMETERS IN THE SCB FOR FM PROFILE 3.     │
   │                                                                      │
   │      INPUT:     NONE                                                 │
   │                                                                      │
   │      OUTPUT:    NONE                                                 │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      THE FOLLOWING PARAMETERS ARE FILLED IN FROM THE FM USAGE FIELD IN THE BIND RU.  │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF RRI = RQ THEN
    DO;
    . SCB.PRI_CHAIN_USE         = BIND_RQ.PRI_CHAIN_USE;
    . SCB.PRI_RQ_MODE           = BIND_RQ.PRI_RQ_MODE;
    . SCB.PRI_COMPR_IND         = BIND_RQ.PRI_COMPR_IND;
    . SCB.PRI_EB_IND            = BIND_RQ.PRI_EB_IND;
    . SCB.SEC_CHAIN_USE         = BIND_RQ.SEC_CHAIN_USE;
    . SCB.SEC_RQ_MODE           = BIND_RQ.SEC_RQ_MODE;
    . SCB.SEC_COMPR_IND         = BIND_RQ.SEC_COMPR_IND;
    . SCB.SEC_EB_IND            = BIND_RQ.SEC_EB_IND;
    . SCB.FM_HDR_USAGE          = BIND_RQ.FM_HDR_USAGE;
    . SCB.BRACKETS_RESET_STATE  = BIND_RQ.BRACKETS_USAGE;
    . SCB.BRKT_TERM_RULE        = BIND_RQ.BRKT_TERM_RULE;
    . SCB.ALT_CODE              = BIND_RQ.ALT_CODE;
    . SCB.SEND_RCV_MODE         = BIND_RQ.SEND_RCV_MODE;
    . SCB.RECOVERY_RESP         = BIND_RQ.RECOVERY_RESP;
    . SCB.CONT_WIN              = BIND_RQ.CONT_WIN_LOSE;
    . SCB.HDX_FF_RESET_STATE    = BIND_RQ.HDX_FF_RESET_STATE;
    END;
  ELSE
    DO;
    . SCB.PRI_CHAIN_USE         = BIND_RSP.PRI_CHAIN_USE;
    . SCB.PRI_RQ_MODE           = BIND_RSP.PRI_RQ_MODE;
    . SCB.PRI_COMPR_IND         = BIND_RSP.PRI_COMPR_IND;
    . SCB.PRI_EB_IND            = BIND_RSP.PRI_EB_IND;
    . SCB.SEC_CHAIN_USE         = BIND_RSP.SEC_CHAIN_USE;
    . SCB.SEC_RQ_MODE           = BIND_RSP.SEC_RQ_MODE;
    . SCB.SEC_COMPR_IND         = BIND_RSP.SEC_COMPR_IND;
    . SCB.SEC_EB_IND            = BIND_RSP.SEC_EB_IND;
    . SCB.FM_HDR_USAGE          = BIND_RSP.FM_HDR_USAGE;
    . SCB.BRACKETS_RESET_STATE  = BIND_RSP.BRACKETS_USAGE;
    . SCB.BRKT_TERM_RULE        = BIND_RSP.BRKT_TERM_RULE;
    . SCB.ALT_CODE              = BIND_RSP.ALT_CODE;
    . SCB.SEND_RCV_MODE         = BIND_RSP.SEND_RCV_MODE;
    . SCB.RECOVERY_RESP         = BIND_RSP.RECOVERY_RESP;
    . SCB.CONT_WIN              = BIND_RSP.CONT_WIN_LOSE;
    . SCB.HDX_FF_RESET_STATE    = BIND_RSP.HDX_FF_RESET_STATE;
    END;
  CALL CHAIN_RSP_SET;                                   /* PAGE 13-79    */
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      THE FOLLOWING SESSION RULES ARE SPECIFIED IN THE DEFINITION OF FM PROFILE 3.  │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  SCB.PRI_RSP_MODE = IMMEDIATE;
  SCB.SEC_RSP_MODE = IMMEDIATE;
  RETURN;
END FM_PROFILE_3;
```

```
FM_PROFILE_4: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO FILL IN PARAMETERS IN THE SCB FOR FM PROFILE 4.   │
   │                                                                      │
   │      INPUT:     NONE                                                 │
   │                                                                      │
   │      OUTPUT:    NONE                                                 │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      THE FOLLOWING PARAMETERS ARE FILLED IN FROM THE FM USAGE FIELD IN THE BIND RU.  │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF RRI = RQ THEN
    DO;
    . SCB.PRI_CHAIN_USE         = BIND_RQ.PRI_CHAIN_USE;
    . SCB.PRI_RQ_MODE           = BIND_RQ.PRI_RQ_MODE;
    . SCB.PRI_COMPR_IND         = BIND_RQ.PRI_COMPR_IND;
    . SCB.PRI_EB_IND            = BIND_RQ.PRI_EB_IND;
    . SCB.SEC_CHAIN_USE         = BIND_RQ.SEC_CHAIN_USE;
    . SCB.SEC_RQ_MODE           = BIND_RQ.SEC_RQ_MODE;
    . SCB.SEC_COMPR_IND         = BIND_RQ.SEC_COMPR_IND;
    . SCB.SEC_EB_IND            = BIND_RQ.SEC_EB_IND;
    . SCB.FM_HDR_USAGE          = BIND_RQ.FM_HDR_USAGE;
    . SCB.BRACKETS_RESET_STATE  = BIND_RQ.BRACKETS_USAGE;
    . SCB.BRKT_TERM_RULE        = BIND_RQ.BRKT_TERM_RULE;
    . SCB.ALT_CODE              = BIND_RQ.ALT_CODE;
    . SCB.SEND_RCV_MODE         = BIND_RQ.SEND_RCV_MODE;
    . SCB.RECOVERY_RESP         = BIND_RQ.RECOVERY_RESP;
    . SCB.CONT_WIN              = BIND_RQ.CONT_WIN_LOSE;
    . SCB.HDX_FF_RESET_STATE    = BIND_RQ.HDX_FF_RESET_STATE;
    END;
  ELSE
    DO;
    . SCB.PRI_CHAIN_USE         = BIND_RSP.PRI_CHAIN_USE;
    . SCB.PRI_RQ_MODE           = BIND_RSP.PRI_RQ_MODE;
    . SCB.PRI_COMPR_IND         = BIND_RSP.PRI_COMPR_IND;
    . SCB.PRI_EB_IND            = BIND_RSP.PRI_EB_IND;
    . SCB.SEC_CHAIN_USE         = BIND_RSP.SEC_CHAIN_USE;
    . SCB.SEC_RQ_MODE           = BIND_RSP.SEC_RQ_MODE;
    . SCB.SEC_COMPR_IND         = BIND_RSP.SEC_COMPR_IND;
    . SCB.SEC_EB_IND            = BIND_RSP.SEC_EB_IND;
    . SCB.FM_HDR_USAGE          = BIND_RSP.FM_HDR_USAGE;
    . SCB.BRACKETS_RESET_STATE  = BIND_RSP.BRACKETS_USAGE;
    . SCB.BRKT_TERM_RULE        = BIND_RSP.BRKT_TERM_RULE;
    . SCB.ALT_CODE              = BIND_RSP.ALT_CODE;
    . SCB.SEND_RCV_MODE         = BIND_RSP.SEND_RCV_MODE;
    . SCB.RECOVERY_RESP         = BIND_RSP.RECOVERY_RESP;
    . SCB.CONT_WIN              = BIND_RSP.CONT_WIN_LOSE;
    . SCB.HDX_FF_RESET_STATE    = BIND_RSP.HDX_FF_RESET_STATE;
    END;
  CALL CHAIN_RSP_SET;                                   /* PAGE 13-79    */
                                                                        */
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      THE FOLLOWING SESSION RULES ARE SPECIFIED IN THE DEFINITION OF FM PROFILE 4.  │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  SCB.PRI_RSP_MODE = IMMEDIATE;
  SCB.SEC_RSP_MODE = IMMEDIATE;
  RETURN;
END FM_PROFILE_4;
```

```
FM_PROFILE_5: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  FILLS IN PARAMETERS IN THE SCB FOR FM PROFILE 5.     │
  │                                                                      │
  │      INPUT:     NONE                                                 │
  │                                                                      │
  │      OUTPUT:    NONE                                                 │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  SCB.PRI_CHAIN_USE         = SINGLE;
  SCB.PRI_RQ_MODE           = DELAYED;
  SCB.PRI_NO_RSP_CHAIN      = NOT_ALLOWED;
  SCB.PRI_EXCP_RSP_CHAIN    = ALLOWED;
  SCB.PRI_DEF_RSP_CHAIN     = ALLOWED;
  SCB.PRI_COMPR_IND         = NO_COMPRESSION;
  SCB.PRI_EB_IND            = MAY_NOT_SEND;
  SCB.SEC_CHAIN_USE         = SINGLE;
  SCB.SEC_RQ_MODE           = DELAYED;
  SCB.SEC_NO_RSP_CHAIN      = NOT_ALLOWED;
  SCB.SEC_EXCP_RSP_CHAIN    = ALLOWED;
  SCB.SEC_DEF_RSP_CHAIN     = ALLOWED;
  SCB.SEC_COMPR_IND         = NO_COMPRESSION;
  SCB.SEC_EB_IND            = MAY_NOT_SEND;
  SCB.FM_HDR_USAGE          = NO_FM_HEADERS;
  SCB.BRACKETS_RESET_STATE  = BRACKETS_NOT_USED;
  SCB.ALT_CODE              = NOT_USED;
  SCB.SEND_RCV_MODE         = FULL_DUPLEX;
  SCB.SEC_RSP_MODE          = DELAYED;
  RETURN;
END FM_PROFILE_5;



FM_PROFILE_6: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  FILLS IN PARAMETERS IN THE SCB FOR FM PROFILE 6.     │
  │                                                                      │
  │      INPUT:     NONE                                                 │
  │                                                                      │
  │      OUTPUT:    NONE                                                 │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  SCB.PRI_CHAIN_USE         = SINGLE;
  SCB.PRI_RQ_MODE           = DELAYED;
  SCB.PRI_NO_RSP_CHAIN      = ALLOWED;
  SCB.PRI_EXCP_RSP_CHAIN    = ALLOWED;
  SCB.PRI_DEF_RSP_CHAIN     = ALLOWED;
  SCB.PRI_COMPR_IND         = NO_COMPRESSION;
  SCB.PRI_EB_IND            = MAY_NOT_SEND;
  SCB.SEC_CHAIN_USE         = SINGLE;
  SCB.SEC_RQ_MODE           = DELAYED;
  SCB.SEC_NO_RSP_CHAIN      = ALLOWED;
  SCB.SEC_EXCP_RSP_CHAIN    = ALLOWED;
  SCB.SEC_DEF_RSP_CHAIN     = ALLOWED;
  SCB.SEC_COMPR_IND         = NO_COMPRESSION;
  SCB.SEC_EB_IND            = MAY_NOT_SEND;
  SCB.FM_HDR_USAGE          = NO_FM_HEADERS;
  SCB.BRACKETS_RESET_STATE  = BRACKETS_NOT_USED;
  SCB.ALT_CODE              = NOT_USED;
  SCB.SEND_RCV_MODE         = FULL_DUPLEX;
  SCB.PRI_RSP_MODE          = DELAYED;
  SCB.SEC_RSP_MODE          = DELAYED;
  RETURN;
END FM_PROFILE_6;
```

```
FM_PROFILE_7: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO FILL IN PARAMETERS IN THE SCB FOR FM PROFILE 7.    │
   │                                                                      │
   │      INPUT:     NONE                                                 │
   │                                                                      │
   │      OUTPUT:    NONE                                                 │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      THE FOLLOWING PARAMETERS ARE FILLED IN FROM THE FM USAGE FIELD IN THE BIND RU.  │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF RRI = RQ THEN
    DO;
    . SCB.PRI_CHAIN_USE         = BIND_RQ.PRI_CHAIN_USE;
    . SCB.PRI_RQ_MODE           = BIND_RQ.PRI_RQ_MODE;
    . SCB.PRI_COMPR_IND         = BIND_RQ.PRI_COMPR_IND;
    . SCB.PRI_EB_IND            = BIND_RQ.PRI_EB_IND;
    . SCB.SEC_CHAIN_USE         = BIND_RQ.SEC_CHAIN_USE;
    . SCB.SEC_RQ_MODE           = BIND_RQ.SEC_RQ_MODE;
    . SCB.SEC_COMPR_IND         = BIND_RQ.SEC_COMPR_IND;
    . SCB.SEC_EB_IND            = BIND_RQ.SEC_EB_IND;
    . SCB.FM_HDR_USAGE          = BIND_RQ.FM_HDR_USAGE;
    . SCB.BRACKETS_RESET_STATE  = BIND_RQ.BRACKETS_USAGE;
    . SCB.BRKT_TERM_RULE        = BIND_RQ.BRKT_TERM_RULE;
    . SCB.ALT_CODE              = BIND_RQ.ALT_CODE;
    . SCB.SEND_RCV_MODE         = BIND_RQ.SEND_RCV_MODE;
    . SCB.RECOVERY_RESP         = BIND_RQ.RECOVERY_RESP;
    . SCB.CONT_WIN              = BIND_RQ.CONT_WIN_LOSE;
    . SCB.HDX_FF_RESET_STATE    = BIND_RQ.HDX_FF_RESET_STATE;
    END;
  ELSE
    DO;
    . SCB.PRI_CHAIN_USE         = BIND_RSP.PRI_CHAIN_USE;
    . SCB.PRI_RQ_MODE           = BIND_RSP.PRI_RQ_MODE;
    . SCB.PRI_COMPR_IND         = BIND_RSP.PRI_COMPR_IND;
    . SCB.PRI_EB_IND            = BIND_RSP.PRI_EB_IND;
    . SCB.SEC_CHAIN_USE         = BIND_RSP.SEC_CHAIN_USE;
    . SCB.SEC_RQ_MODE           = BIND_RSP.SEC_RQ_MODE;
    . SCB.SEC_COMPR_IND         = BIND_RSP.SEC_COMPR_IND;
    . SCB.SEC_EB_IND            = BIND_RSP.SEC_EB_IND;
    . SCB.FM_HDR_USAGE          = BIND_RSP.FM_HDR_USAGE;
    . SCB.BRACKETS_RESET_STATE  = BIND_RSP.BRACKETS_USAGE;
    . SCB.BRKT_TERM_RULE        = BIND_RSP.BRKT_TERM_RULE;
    . SCB.ALT_CODE              = BIND_RSP.ALT_CODE;
    . SCB.SEND_RCV_MODE         = BIND_RSP.SEND_RCV_MODE;
    . SCB.RECOVERY_RESP         = BIND_RSP.RECOVERY_RESP;
    . SCB.CONT_WIN              = BIND_RSP.CONT_WIN_LOSE;
    . SCB.HDX_FF_RESET_STATE    = BIND_RSP.HDX_FF_RESET_STATE;
    END;
  CALL CHAIN_RSP_SET;                                   /* PAGE 13-79   */
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      THE FOLLOWING SESSION RULES ARE SPECIFIED IN THE DEFINITION OF FM PROFILE 7.  │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  SCB.PRI_RSP_MODE = IMMEDIATE;
  SCB.SEC_RSP_MODE = IMMEDIATE;
  RETURN;
END FM_PROFILE_7;
```

```
FM_PROFILE_17: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  FILLS IN PARAMETERS IN THE SCB FOR FM PROFILE 17.    │
   │                                                                      │
   │      INPUT:     NONE                                                 │
   │                                                                      │
   │      OUTPUT:    NONE                                                 │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  SCB.PRI_CHAIN_USE         = SINGLE;
  SCB.PRI_RQ_MODE           = DELAYED;
  SCB.PRI_NO_RSP_CHAIN      = NOT_ALLOWED;
  SCB.PRI_EXCP_RSP_CHAIN    = NOT_ALLOWED;
  SCB.PRI_DEF_RSP_CHAIN     = ALLOWED;
  SCB.PRI_COMPR_IND         = NO_COMPRESSION;
  SCB.PRI_EB_IND            = MAY_NOT_SEND;
  SCB.SEC_CHAIN_USE         = SINGLE;
  SCB.SEC_RQ_MODE           = DELAYED;
  SCB.SEC_NO_RSP_CHAIN      = NOT_ALLOWED;
  SCB.SEC_EXCP_RSP_CHAIN    = NOT_ALLOWED;
  SCB.SEC_DEF_RSP_CHAIN     = ALLOWED;
  SCB.SEC_COMPR_IND         = NO_COMPRESSION;
  SCB.SEC_EB_IND            = MAY_NOT_SEND;
  SCB.FM_HDR_USAGE          = NO_FM_HEADERS;
  SCB.BRACKETS_RESET_STATE  = BRACKETS_NOT_USED;
  SCB.ALT_CODE              = NOT_USED;
  SCB.SEND_RCV_MODE         = FULL_DUPLEX;
  SCB.PRI_RSP_MODE          = IMMEDIATE;
  SCB.SEC_RSP_MODE          = IMMEDIATE;
  RETURN;
END FM_PROFILE_17;
```

```
FM_PROFILE_18: PROCEDURE;

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  FILLS IN PARAMETERS IN THE SCB FOR FM PROFILE 18.    │
   │                                                                      │
   │      INPUT:     NONE                                                 │
   │                                                                      │
   │      OUTPUT:    NONE                                                 │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      THE FOLLOWING PARAMETERS ARE FILLED IN FROM THE FM USAGE FIELD IN THE BIND RU.  │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  IF RRI = RQ THEN
    DO;
    . SCB.PRI_CHAIN_USE         = BIND_RQ.PRI_CHAIN_USE;
    . SCB.PRI_RQ_MODE           = BIND_RQ.PRI_RQ_MODE;
    . SCB.PRI_COMPR_IND         = BIND_RQ.PRI_COMPR_IND;
    . SCB.PRI_EB_IND            = BIND_RQ.PRI_EB_IND;
    . SCB.SEC_CHAIN_USE         = BIND_RQ.SEC_CHAIN_USE;
    . SCB.SEC_RQ_MODE           = BIND_RQ.SEC_RQ_MODE;
    . SCB.SEC_COMPR_IND         = BIND_RQ.SEC_COMPR_IND;
    . SCB.SEC_EB_IND            = BIND_RQ.SEC_EB_IND;
    . SCB.FM_HDR_USAGE          = BIND_RQ.FM_HDR_USAGE;
    . SCB.BRACKETS_RESET_STATE  = BIND_RQ.BRACKETS_USAGE;
    . SCB.BRKT_TERM_RULE        = BIND_RQ.BRKT_TERM_RULE;
    . SCB.ALT_CODE              = BIND_RQ.ALT_CODE;
    . SCB.SEND_RCV_MODE         = BIND_RQ.SEND_RCV_MODE;
    . SCB.RECOVERY_RESP         = BIND_RQ.RECOVERY_RESP;
    . SCB.CONT_WIN              = BIND_RQ.CONT_WIN_LOSE;
    . SCB.HDX_FF_RESET_STATE    = BIND_RQ.HDX_FF_RESET_STATE;
    END;
  ELSE
    DO;
    . SCB.PRI_CHAIN_USE         = BIND_RSP.PRI_CHAIN_USE;
    . SCB.PRI_RQ_MODE           = BIND_RSP.PRI_RQ_MODE;
    . SCB.PRI_COMPR_IND         = BIND_RSP.PRI_COMPR_IND;
    . SCB.PRI_EB_IND            = BIND_RSP.PRI_EB_IND;
    . SCB.SEC_CHAIN_USE         = BIND_RSP.SEC_CHAIN_USE;
    . SCB.SEC_RQ_MODE           = BIND_RSP.SEC_RQ_MODE;
    . SCB.SEC_COMPR_IND         = BIND_RSP.SEC_COMPR_IND;
    . SCB.SEC_EB_IND            = BIND_RSP.SEC_EB_IND;
    . SCB.FM_HDR_USAGE          = BIND_RSP.FM_HDR_USAGE;
    . SCB.BRACKETS_RESET_STATE  = BIND_RSP.BRACKETS_USAGE;
    . SCB.BRKT_TERM_RULE        = BIND_RSP.BRKT_TERM_RULE;
    . SCB.ALT_CODE              = BIND_RSP.ALT_CODE;
    . SCB.SEND_RCV_MODE         = BIND_RSP.SEND_RCV_MODE;
    . SCB.RECOVERY_RESP         = BIND_RSP.RECOVERY_RESP;
    . SCB.CONT_WIN              = BIND_RSP.CONT_WIN_LOSE;
    . SCB.HDX_FF_RESET_STATE    = BIND_RSP.HDX_FF_RESET_STATE;
    END;
  CALL CHAIN_RSP_SET;                                   /* PAGE 13-79    */
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      THE FOLLOWING SESSION RULES ARE SPECIFIED IN THE DEFINITION OF FM PROFILE 18.  │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  SCB.PRI_RSP_MODE = IMMEDIATE;
  SCB.SEC_RSP_MODE = IMMEDIATE;
  RETURN;
END FM_PROFILE_18;
```

```
CHAIN_RSP_SET: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE SETS UP THE CHAIN RESPONSE ALLOWED FIELDS.
  │
  │      INPUT:     NONE
  │
  │      OUTPUT:    NONE
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL CHAIN_RSP_PRI BIT(2);
  DCL CHAIN_RSP_SEC BIT(2);
  IF RRI = RQ THEN
    DO;
    . CHAIN_RSP_PRI = BIND_RQ.PRI_CHAIN_RSP;
    . CHAIN_RSP_SEC = BIND_RQ.SEC_CHAIN_RSP;
    END;
  ELSE
    DO;
    . CHAIN_RSP_PRI = BIND_RSP.PRI_CHAIN_RSP;
    . CHAIN_RSP_SEC = BIND_RSP.SEC_CHAIN_RSP;
    END;
  SELECT ANYORDER(CHAIN_RSP_PRI);
  .
  . WHEN(NO_RESPONSE)
  .   DO;
  .   . SCB.PRI_NO_RSP_CHAIN   = ALLOWED;
  .   . SCB.PRI_EXCP_RSP_CHAIN = NOT_ALLOWED;
  .   . SCB.PRI_DEF_RSP_CHAIN  = NOT_ALLOWED;
  .   END;
  .
  . WHEN(EXCP_RESPONSE)
  .   DO;
  .   . SCB.PRI_NO_RSP_CHAIN   = NOT_ALLOWED;
  .   . SCB.PRI_EXCP_RSP_CHAIN = ALLOWED;
  .   . SCB.PRI_DEF_RSP_CHAIN  = NOT_ALLOWED;
  .   END;
  .
  . WHEN(DEF_RESPONSE)
  .   DO;
  .   . SCB.PRI_NO_RSP_CHAIN   = NOT_ALLOWED;
  .   . SCB.PRI_EXCP_RSP_CHAIN = NOT_ALLOWED;
  .   . SCB.PRI_DEF_RSP_CHAIN  = ALLOWED;
  .   END;
  .
  . WHEN(DEF_OR_EXCP_RESPONSE)
  .   DO;
  .   . SCB.PRI_NO_RSP_CHAIN   = NOT_ALLOWED;
  .   . SCB.PRI_EXCP_RSP_CHAIN = ALLOWED;
  .   . SCB.PRI_DEF_RSP_CHAIN  = ALLOWED;
  .   END;
  END;
  SELECT ANYORDER(CHAIN_RSP_SEC);
  .
  . WHEN(NO_RESPONSE)
  .   DO;
  .   . SCB.SEC_NO_RSP_CHAIN   = ALLOWED;
  .   . SCB.SEC_EXCP_RSP_CHAIN = NOT_ALLOWED;
  .   . SCB.SEC_DEF_RSP_CHAIN  = NOT_ALLOWED;
  .   END;
  .
  . WHEN(EXCP_RESPONSE)
  .   DO;
  .   . SCB.SEC_NO_RSP_CHAIN   = NOT_ALLOWED;
  .   . SCB.SEC_EXCP_RSP_CHAIN = ALLOWED;
  .   . SCB.SEC_DEF_RSP_CHAIN  = NOT_ALLOWED;
  .   END;
  .
  . WHEN(DEF_RESPONSE)
  .   DO;
  .   . SCB.SEC_NO_RSP_CHAIN   = NOT_ALLOWED;
  .   . SCB.SEC_EXCP_RSP_CHAIN = NOT_ALLOWED;
  .   . SCB.SEC_DEF_RSP_CHAIN  = ALLOWED;
  .   END;
  .
  . WHEN(DEF_OR_EXCP_RESPONSE)
  .   DO;
  .   . SCB.SEC_NO_RSP_CHAIN   = NOT_ALLOWED;
  .   . SCB.SEC_EXCP_RSP_CHAIN = ALLOWED;
  .   . SCB.SEC_DEF_RSP_CHAIN  = ALLOWED;
  .   END;
  END;
  RETURN;
END CHAIN_RSP_SET;
```

```
TS_PROFILE_1: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  FILLS IN PARAMETERS IN THE SCB FOR TS PROFILE 1.         │
   │                                                                          │
   │      INPUT:     NONE                                                     │
   │                                                                          │
   │      OUTPUT:    NONE                                                     │
   │                                                                          │
   │      NOTE:      USED ONLY IN ACTLU AND ACTPU TO PERIPHERAL NODES          │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SCB.SEC_STAGING_IND     = 0;                          /* ONE-STAGE PACING     */
  SCB.SEC_RCV_PACING_CNT  = 0;                          /* NO PACING            */
  SCB.SEC_SEND_PACING_CNT = 0;                          /* NO PACING            */
  IF SCB.TYPE_OF_SESSION = SSCP_PU THEN
    DO;
    . SCB.SEC_SEND_MAX_RU_SIZE = 0;
    . SCB.PRI_SEND_MAX_RU_SIZE = 0;
    END;
  SCB.PRI_STAGING_IND     = 0;                          /* TWO-STAGE PACING     */
  SCB.PRI_SEND_PACING_CNT = 0;                          /* NO PACING            */
  SCB.PRI_RCV_PACING_CNT  = 0;                          /* NO PACING            */
  SCB.SQN_USAGE           = IDENTIFIERS;
  SCB.SC_CLEAR            = NOT_ALLOWED;
  SCB.SC_RQR              = NOT_ALLOWED;
  SCB.SC_SDT              = NOT_ALLOWED;
  SCB.SC_STSN             = NOT_ALLOWED;
  SCB.SC_CRV              = NOT_ALLOWED;
  RETURN;
END TS_PROFILE_1;



TS_PROFILE_2: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  FILLS IN PARAMETERS IN THE SCB FOR TS PROFILE 2.         │
   │                                                                          │
   │      INPUT:     NONE                                                     │
   │                                                                          │
   │      OUTPUT:    NONE                                                     │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF RRI = RQ THEN
    DO;
    . SCB.SEC_STAGING_IND      = BIND_RQ.SEC_STAGING_IND;
    . SCB.SEC_SEND_PACING_CNT  = BIND_RQ.SEC_SEND_PACING_CNT;
    . SCB.SEC_RCV_PACING_CNT   = BIND_RQ.SEC_RCV_PACING_CNT;
    . SCB.SEC_SEND_MAX_RU_SIZE = BIND_RQ.SEC_SEND_MAX_RU_SIZE;
    . SCB.PRI_STAGING_IND      = BIND_RQ.PRI_STAGING_IND;
    . SCB.PRI_SEND_PACING_CNT  = BIND_RQ.PRI_SEND_PACING_CNT;
    . SCB.PRI_RCV_PACING_CNT   = BIND_RQ.PRI_RCV_PACING_CNT;
    . SCB.PRI_SEND_MAX_RU_SIZE = BIND_RQ.PRI_SEND_MAX_RU_SIZE;
    END;
  ELSE
    DO;
    . SCB.SEC_STAGING_IND      = BIND_RSP.SEC_STAGING_IND;
    . SCB.SEC_SEND_PACING_CNT  = BIND_RSP.SEC_SEND_PACING_CNT;
    . SCB.SEC_RCV_PACING_CNT   = BIND_RSP.SEC_RCV_PACING_CNT;
    . SCB.SEC_SEND_MAX_RU_SIZE = BIND_RSP.SEC_SEND_MAX_RU_SIZE;
    . SCB.PRI_STAGING_IND      = BIND_RSP.PRI_STAGING_IND;
    . SCB.PRI_SEND_PACING_CNT  = BIND_RSP.PRI_SEND_PACING_CNT;
    . SCB.PRI_RCV_PACING_CNT   = BIND_RSP.PRI_RCV_PACING_CNT;
    . SCB.PRI_SEND_MAX_RU_SIZE = BIND_RSP.PRI_SEND_MAX_RU_SIZE;
    END;
  IF NCB.PU_TYPE = T1 THEN
    SCB.SQN_USAGE = NO_SNF;
  ELSE
    SCB.SQN_USAGE = SEQUENCE_NUMBERS;
  SCB.SC_CLEAR = ALLOWED;
  SCB.SC_RQR   = NOT_ALLOWED;
  SCB.SC_SDT   = NOT_ALLOWED;
  SCB.SC_STSN  = NOT_ALLOWED;
  SCB.SC_CRV   = NOT_ALLOWED;
  RETURN;
END TS_PROFILE_2;
```

```
TS_PROFILE_3: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  FILLS IN PARAMETERS IN THE SCB FOR TS PROFILE 3.             │
  │                                                                              │
  │      INPUT:     NONE                                                         │
  │                                                                              │
  │      OUTPUT:    NONE                                                         │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF RRI = RQ THEN
    DO;
    . SCB.SEC_STAGING_IND      = BIND_RQ.SEC_STAGING_IND;
    . SCB.SEC_SEND_PACING_CNT  = BIND_RQ.SEC_SEND_PACING_CNT;
    . SCB.SEC_RCV_PACING_CNT   = BIND_RQ.SEC_RCV_PACING_CNT;
    . SCB.SEC_SEND_MAX_RU_SIZE = BIND_RQ.SEC_SEND_MAX_RU_SIZE;
    . SCB.PRI_STAGING_IND      = BIND_RQ.PRI_STAGING_IND;
    . SCB.PRI_SEND_PACING_CNT  = BIND_RQ.PRI_SEND_PACING_CNT;
    . SCB.PRI_RCV_PACING_CNT   = BIND_RQ.PRI_RCV_PACING_CNT;
    . SCB.PRI_SEND_MAX_RU_SIZE = BIND_RQ.PRI_SEND_MAX_RU_SIZE;
    . IF BIND_RQ.CRYPTOGRAPHY_LENGTH ¬= 0 THEN
    .   DO;
    .   . SCB.CRYPTOGRAPHY_SESSION_LEVEL     = BIND_RQ.CRYPTOGRAPHY_SESSION_LEVEL;
    .   . SCB.CRYPTOGRAPHY_KEY_ENCIPH_METHOD = BIND_RQ.CRYPTOGRAPHY_KEY_ENCIPH_METHOD;
    .   . SCB.CRYPTOGRAPHY_CIPHER_METHOD     = BIND_RQ.CRYPTOGRAPHY_CIPHER_METHOD;
    .   END;
    END;
  ELSE
    DO;
    . SCB.SEC_STAGING_IND      = BIND_RSP.SEC_STAGING_IND;
    . SCB.SEC_SEND_PACING_CNT  = BIND_RSP.SEC_SEND_PACING_CNT;
    . SCB.SEC_RCV_PACING_CNT   = BIND_RSP.SEC_RCV_PACING_CNT;
    . SCB.SEC_SEND_MAX_RU_SIZE = BIND_RSP.SEC_SEND_MAX_RU_SIZE;
    . SCB.PRI_STAGING_IND      = BIND_RSP.PRI_STAGING_IND;
    . SCB.PRI_SEND_PACING_CNT  = BIND_RSP.PRI_SEND_PACING_CNT;
    . SCB.PRI_RCV_PACING_CNT   = BIND_RSP.PRI_RCV_PACING_CNT;
    . SCB.PRI_SEND_MAX_RU_SIZE = BIND_RSP.PRI_SEND_MAX_RU_SIZE;
    END;
  IF NCB.PU_TYPE = T1 THEN
    SCB.SQN_USAGE = NO_SNF;
  ELSE
    SCB.SQN_USAGE = SEQUENCE_NUMBERS;
  SCB.SC_CLEAR = ALLOWED;
  SCB.SC_RQR   = NOT_ALLOWED;
  SCB.SC_SDT   = ALLOWED;
  SCB.SC_STSN  = NOT_ALLOWED;
  IF SCB.CRYPTOGRAPHY_SESSION_LEVEL = (SELECTIVE | MANDATORY) THEN
    SCB.SC_CRV = ALLOWED;
  ELSE
    SCB.SC_CRV = NOT_ALLOWED;
  RETURN;
END TS_PROFILE_3;
```

```
TS_PROFILE_4: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  FILLS IN PARAMETERS IN THE SCB FOR TS PROFILE 4.             │
  │                                                                              │
  │      INPUT:     NONE                                                         │
  │                                                                              │
  │      OUTPUT:    NONE                                                         │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF RRI = RQ THEN
    DO;
    . SCB.SEC_STAGING_IND         = BIND_RQ.SEC_STAGING_IND;
    . SCB.SEC_SEND_PACING_CNT     = BIND_RQ.SEC_SEND_PACING_CNT;
    . SCB.SEC_RCV_PACING_CNT      = BIND_RQ.SEC_RCV_PACING_CNT;
    . SCB.SEC_SEND_MAX_RU_SIZE    = BIND_RQ.SEC_SEND_MAX_RU_SIZE;
    . SCB.PRI_STAGING_IND         = BIND_RQ.PRI_STAGING_IND;
    . SCB.PRI_SEND_PACING_CNT     = BIND_RQ.PRI_SEND_PACING_CNT;
    . SCB.PRI_RCV_PACING_CNT      = BIND_RQ.PRI_RCV_PACING_CNT;
    . SCB.PRI_SEND_MAX_RU_SIZE    = BIND_RQ.PRI_SEND_MAX_RU_SIZE;
    . SCB.PRI_TWO_PHASE_COMMIT    = BIND_RQ.PRI_TWO_PHASE_COMMIT;
    . SCB.SEC_TWO_PHASE_COMMIT    = BIND_RQ.SEC_TWO_PHASE_COMMIT;
    . IF BIND_RQ.CRYPTOGRAPHY_LENGTH ¬= 0 THEN
    .   DO;
    .   . SCB.CRYPTOGRAPHY_SESSION_LEVEL      = BIND_RQ.CRYPTOGRAPHY_SESSION_LEVEL;
    .   . SCB.CRYPTOGRAPHY_KEY_ENCIPH_METHOD  = BIND_RQ.CRYPTOGRAPHY_KEY_ENCIPH_METHOD;
    .   . SCB.CRYPTOGRAPHY_CIPHER_METHOD      = BIND_RQ.CRYPTOGRAPHY_CIPHER_METHOD;
    .   END;
    END;
  ELSE
    DO;
    . SCB.SEC_STAGING_IND         = BIND_RSP.SEC_STAGING_IND;
    . SCB.SEC_SEND_PACING_CNT     = BIND_RSP.SEC_SEND_PACING_CNT;
    . SCB.SEC_RCV_PACING_CNT      = BIND_RSP.SEC_RCV_PACING_CNT;
    . SCB.SEC_SEND_MAX_RU_SIZE    = BIND_RSP.SEC_SEND_MAX_RU_SIZE;
    . SCB.PRI_STAGING_IND         = BIND_RSP.PRI_STAGING_IND;
    . SCB.PRI_SEND_PACING_CNT     = BIND_RSP.PRI_SEND_PACING_CNT;
    . SCB.PRI_RCV_PACING_CNT      = BIND_RSP.PRI_RCV_PACING_CNT;
    . SCB.PRI_SEND_MAX_RU_SIZE    = BIND_RSP.PRI_SEND_MAX_RU_SIZE;
    . SCB.PRI_TWO_PHASE_COMMIT    = BIND_RQ.PRI_TWO_PHASE_COMMIT;
    . SCB.SEC_TWO_PHASE_COMMIT    = BIND_RQ.SEC_TWO_PHASE_COMMIT;
    END;
  IF NCB.PU_TYPE = T1 THEN
    SCB.SQN_USAGE = NO_SNF;
  ELSE
    SCB.SQN_USAGE = SEQUENCE_NUMBERS;
  SCB.SC_CLEAR = ALLOWED;
  SCB.SC_RQR   = ALLOWED;
  SCB.SC_SDT   = ALLOWED;
  SCB.SC_STSN  = ALLOWED;
  IF SCB.CRYPTOGRAPHY_SESSION_LEVEL = (SELECTIVE | MANDATORY) THEN
    SCB.SC_CRV = ALLOWED;
  ELSE
    SCB.SC_CRV = NOT_ALLOWED;
  RETURN;
END TS_PROFILE_4;



TS_PROFILE_5: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  FILLS IN PARAMETERS IN THE SCB FOR TS PROFILE 5.             │
  │                                                                              │
  │      INPUT:     NONE                                                         │
  │                                                                              │
  │      OUTPUT:    NONE                                                         │
  │                                                                              │
  │      NOTE:      USED ONLY IN ACTPU TO SUBAREA PU'S                           │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SCB.SEC_STAGING_IND      = 0;                          /* ONE-STAGE PACING     */
  SCB.SEC_RCV_PACING_CNT   = 0;                          /* NO PACING            */
  SCB.SEC_SEND_PACING_CNT  = 0;                          /* NO PACING            */
  SCB.SEC_SEND_MAX_RU_SIZE = 0;
  SCB.PRI_STAGING_IND      = 0;                          /* TWO-STAGE PACING     */
  SCB.PRI_SEND_PACING_CNT  = 0;                          /* NO PACING            */
  SCB.PRI_RCV_PACING_CNT   = 0;                          /* NO PACING            */
  SCB.PRI_SEND_MAX_RU_SIZE = 0;
  SCB.SQN_USAGE            = SEQUENCE_NUMBERS;
  SCB.SC_CLEAR             = NOT_ALLOWED;
  SCB.SC_RQR               = NOT_ALLOWED;
  SCB.SC_SDT               = ALLOWED;
  SCB.SC_STSN              = NOT_ALLOWED;
  SCB.SC_CRV               = NOT_ALLOWED;
  RETURN;
END TS_PROFILE_5;
```

```
TS_PROFILE_7: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  FILLS IN PARAMETERS  IN THE SCB FOR TS PROFILE 7.            │
  │                                                                              │
  │      INPUT:     NONE                                                         │
  │                                                                              │
  │      OUTPUT:    NONE                                                         │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  IF RRI = RQ THEN
    DO;
    . SCB.SEC_STAGING_IND      = BIND_RQ.SEC_STAGING_IND;
    . SCB.SEC_SEND_PACING_CNT  = BIND_RQ.SEC_SEND_PACING_CNT;
    . SCB.SEC_RCV_PACING_CNT   = BIND_RQ.SEC_RCV_PACING_CNT;
    . SCB.SEC_SEND_MAX_RU_SIZE = BIND_RQ.SEC_SEND_MAX_RU_SIZE;
    . SCB.PRI_STAGING_IND      = BIND_RQ.PRI_STAGING_IND;
    . SCB.PRI_SEND_PACING_CNT  = BIND_RQ.PRI_SEND_PACING_CNT;
    . SCB.PRI_RCV_PACING_CNT   = BIND_RQ.PRI_RCV_PACING_CNT;
    . SCB.PRI_SEND_MAX_RU_SIZE = BIND_RQ.PRI_SEND_MAX_RU_SIZE;
    . IF BIND_RQ.CRYPTOGRAPHY_LENGTH ¬= 0 THEN
    .   DO;
    .   . SCB.CRYPTOGRAPHY_SESSION_LEVEL     = BIND_RQ.CRYPTOGRAPHY_SESSION_LEVEL;
    .   . SCB.CRYPTOGRAPHY_KEY_ENCIPH_METHOD = BIND_RQ.CRYPTOGRAPHY_KEY_ENCIPH_METHOD;
    .   . SCB.CRYPTOGRAPHY_CIPHER_METHOD     = BIND_RQ.CRYPTOGRAPHY_CIPHER_METHOD;
    .   END;
    END;
  ELSE
    DO;
    . SCB.SEC_STAGING_IND      = BIND_RSP.SEC_STAGING_IND;
    . SCB.SEC_SEND_PACING_CNT  = BIND_RSP.SEC_SEND_PACING_CNT;
    . SCB.SEC_RCV_PACING_CNT   = BIND_RSP.SEC_RCV_PACING_CNT;
    . SCB.SEC_SEND_MAX_RU_SIZE = BIND_RSP.SEC_SEND_MAX_RU_SIZE;
    . SCB.PRI_STAGING_IND      = BIND_RSP.PRI_STAGING_IND;
    . SCB.PRI_SEND_PACING_CNT  = BIND_RSP.PRI_SEND_PACING_CNT;
    . SCB.PRI_RCV_PACING_CNT   = BIND_RSP.PRI_RCV_PACING_CNT;
    . SCB.PRI_SEND_MAX_RU_SIZE = BIND_RSP.PRI_SEND_MAX_RU_SIZE;
    END;
  IF NCB.PU_TYPE = T1 THEN
    SCB.SQN_USAGE = NO_SNF;
  ELSE
    SCB.SQN_USAGE = SEQUENCE_NUMBERS;
  SCB.SC_CLEAR = NOT_ALLOWED;
  SCB.SC_RQR   = NOT_ALLOWED;
  SCB.SC_SDT   = NOT_ALLOWED;
  SCB.SC_STSN  = NOT_ALLOWED;
  IF SCB.CRYPTOGRAPHY_SESSION_LEVEL = (SELECTIVE | MANDATORY) THEN
    SCB.SC_CRV = ALLOWED;
  ELSE
    SCB.SC_CRV = NOT_ALLOWED;
  RETURN;
END TS_PROFILE_7;



TS_PROFILE_17: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  FILLS IN PARAMETERS IN THE SCB FOR TS PROFILE 17.            │
  │                                                                              │
  │      INPUT:     NONE                                                         │
  │                                                                              │
  │      OUTPUT:    NONE                                                         │
  │                                                                              │
  │      NOTE:      THE PACING COUNTS  ARE CARRIED ON BOTH THE REQUEST  AND RESPONSE FOR
  │                 ACTCDRM AND THESE VALUES ARE FILLED IN THE SCB BY SESSACT.(REQUEST |
  │                 RESPONSE).                                                   │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  SCB.SEC_STAGING_IND      = 0;                          /* ONE-STAGE PACING    */
  SCB.SEC_SEND_MAX_RU_SIZE = 0;
  SCB.PRI_STAGING_IND      = 1;                          /* ONE-STAGE PACING    */
  SCB.PRI_SEND_MAX_RU_SIZE = 0;
  IF NCB.PU_TYPE = T1 THEN
    SCB.SQN_USAGE = NO_SNF;
  ELSE
    SCB.SQN_USAGE = IDENTIFIERS;
  SCB.SC_SDT   = ALLOWED;
  SCB.SC_STSN  = NOT_ALLOWED;
  SCB.SC_CRV   = NOT_ALLOWED;
  RETURN;
END TS_PROFILE_17;
```

This page intentionally left blank

```
BF_TS_PARAMETERS: PROCEDURE;

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  FILLS IN THE SESSION ACTIVATION  PARAMETERS IN THE BOUNDARY FUNCTION │
  │                 SCB.                                                 │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  SELECT(RQ_CODE);

  . WHEN(ACTLU)
  .   DO;
  .   . SCB.PACING_PARAMETERS = 0;
  .   . SCB.SQN_USAGE         = IDENTIFIERS;
  .   . SCB.SC_CLEAR          = NOT_ALLOWED;
  .   . IF RRI = RQ THEN
  .   .   SCB.TS_PROFILE      = PAD_4_BITS||ACTLU_RQ.TS_PROFILE;
  .   . ELSE
  .   .   IF DCF >= RSP_OF_LENGTH_TWO THEN
  .   .     SCB.TS_PROFILE = PAD_4_BITS||ACTLU_RSP.TS_PROFILE;
  .   END;
  .
  . WHEN(ACTPU)
  .   DO;
  .   . SCB.PACING_PARAMETERS = 0;
  .   . SCB.SQN_USAGE         = IDENTIFIERS;
  .   . SCB.SC_CLEAR          = NOT_ALLOWED;
  .   . IF RRI = RQ THEN
  .   .   SCB.TS_PROFILE = PAD_4_BITS||ACTPU_RQ.TS_PROFILE;
  .   END;
  .
  . WHEN(BIND)
  .   DO;
  .   . IF RRI = RQ THEN
  .   .   DO;
  .   .   . SCB.SEC_STAGING_IND     = BIND_RQ.SEC_STAGING_IND;
  .   .   . SCB.SEC_SEND_PACING_CNT = BIND_RQ.SEC_SEND_PACING_CNT;
  .   .   . SCB.SEC_RCV_PACING_CNT  = BIND_RQ.SEC_RCV_PACING_CNT;
  .   .   . SCB.PRI_STAGING_IND     = BIND_RQ.PRI_STAGING_IND;
  .   .   . SCB.PRI_SEND_PACING_CNT = BIND_RQ.PRI_SEND_PACING_CNT;
  .   .   . SCB.PRI_RCV_PACING_CNT  = BIND_RQ.PRI_RCV_PACING_CNT;
  .   .   . SCB.TS_PROFILE          = BIND_RQ.TS_PROFILE;
  .   .   END;
  .   . ELSE
  .   .   DO;
  .   .   . SCB.SEC_STAGING_IND     = BIND_RSP.SEC_STAGING_IND;
  .   .   . SCB.SEC_SEND_PACING_CNT = BIND_RSP.SEC_SEND_PACING_CNT;
  .   .   . SCB.SEC_RCV_PACING_CNT  = BIND_RSP.SEC_RCV_PACING_CNT;
  .   .   . SCB.PRI_STAGING_IND     = BIND_RSP.PRI_STAGING_IND;
  .   .   . SCB.PRI_SEND_PACING_CNT = BIND_RSP.PRI_SEND_PACING_CNT;
  .   .   . SCB.PRI_RCV_PACING_CNT  = BIND_RSP.PRI_RCV_PACING_CNT;
  .   .   . SCB.TS_PROFILE          = BIND_RSP.TS_PROFILE;
  .   .   END;
  .   . SCB.SQN_USAGE = SEQUENCE_NUMBERS;
  .   . IF SCB.TS_PROFILE = 7 THEN
  .   .   SCB.SC_CLEAR = NOT_ALLOWED;
  .   . ELSE
  .   .   SCB.SC_CLEAR = ALLOWED;
  .   END;
  END;
  RETURN;
END BF_TS_PARAMETERS;
```

```
SCB_CREATE: PROCEDURE;

                                                                                /*
 ┌──────────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  CREATES AN  SCB AND  INITIALIZES THE  SCB ACCORDING  TO THE  TYPE OF  │
 │                 SESSION THAT  WILL BE SUPPORTED.  ALLOCATES  SPACE FOR THE  TCCB FOR  │
 │                 USE BY CHAPTER 4.                                            │
 │                                                                              │
 │      INPUT:     NONE                                                         │
 │                                                                              │
 │      OUTPUT:    SCB IS CREATED                                               │
 └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */

  CREATE SCB;
  IF SCB_PTR ¬= NULL THEN                    /* SPACE HAS BEEN ALLOCATED          */
    DO;
    . CREATE SCB.TC_CB_PTR->TCCB;                                 /* FOR CHAPTER 4 */
    . IF SCB.TC_CB_PTR = NULL THEN           /* SPACE COULD NOT BE ALLOCATED      */
    .   DISCARD SCB;                         /* DEALLOCATE SPACE FOR SCB          */
    . ELSE
    .   DO;
    .   . IF CB_TYPE = BF_SESS THEN
    .   .   CREATE SCB.SEC_TO_BF_TC_CB_PTR->TCCB;                 /* FOR CHAPTER 4 */
    .   . IF SCB.SEC_TO_BF_TC_CB_PTR = NULL THEN /* SPACE COULD NOT BE ALLOCATED */
    .   .   DO;
    .   .   . DISCARD SCB.TC_CB_PTR->TCCB; /* DEALLOCATE SPACE FOR TCCB         */
    .   .   . DISCARD SCB;                 /* DEALLOCATE SPACE FOR SCB          */
    .   .   END;
    .   END;
    END;
  IF SCB_PTR ¬= NULL THEN
    DO;
    . INSERT SCB IN SCB_LIST;
    .
    . SELECT ANYORDER(NCB.PU_TYPE);
    . . WHEN(T1)
    . .   SCB.LOCAL_SESSION_ID = LSID;
    . . WHEN(T2)
    . .   IF MUCB.DIRECTION = SEND THEN
    . .     DO;
    . .     . SCB.THIS_ID = OAFPRIME;
    . .     . SCB.PARTNER_ID = DAFPRIME;
    . .     END;
    . .   ELSE
    . .     DO;
    . .     . SCB.THIS_ID = DAFPRIME;
    . .     . SCB.PARTNER_ID = OAFPRIME;
    . .     END;
    . . WHEN(T4 | T5)
    . .   DO;
    . .   . SCB.VRCBPTR = NULL;
    . .   . IF MUCB.DIRECTION = SEND THEN
    . .   .   DO;
    . .   .   . SCB.THIS_SA = OSAF;
    . .   .   . SCB.THIS_EA = OEF;
    . .   .   . SCB.PARTNER_SA = DSAF;
    . .   .   . SCB.PARTNER_EA = DEF;
    . .   .   END;
    . .   . ELSE                             /* MUCB.DIRECTION = RECEIVE          */
    . .   .   DO;
    . .   .   . SCB.THIS_SA = DSAF;
    . .   .   . SCB.THIS_EA = DEF;
    . .   .   . SCB.PARTNER_SA = OSAF;
    . .   .   . SCB.PARTNER_EA = OEF;
    . .   .   END;
    . .   END;

    . . IF CB_TYPE = BF_SESS THEN
    . .   DO;
    . .   . SCB.SUPPORTED_PU_TYPE = NRCB.RESOURCE_TYPE;
    . .   . IF SCB.SUPPORTED_PU_TYPE = T1 THEN
    . .   .   DO;
    . .   .   . SCB.LOCAL_SESSION_ID = LSID;
    . .   .   . SELECT ANYORDER(RQ_CODE);
    . .   .   . . WHEN(BIND)
    . .   .   . .   SCB.LOCAL_SESSION_ID(0:1) = B'11';
    . .   .   . . WHEN(ACTLU)
    . .   .   . .   SCB.LOCAL_SESSION_ID(0:1) = B'01';
    . .   .   . END;
    . .   .   END;
    . .   . ELSE                                          /* PU_T2             */
    . .   .   SELECT ANYORDER(RQ_CODE);
    . .   .   . WHEN(ACTLU | ACTPU)
    . .   .   .   SCB.PARTNER_ID = 0;        /* 0 DENOTES SSCP--SEE CHAPTER 2     */
    . .   .   . WHEN(BIND)
    . .   .   .   SCB.PARTNER_ID = 1;        /* SHARE LIMIT OF 1--THEREFORE, 1 IS UNIQUE */
    . .   .   END;
    . .   END;
    . END;
    . SELECT ANYORDER(RQ_CODE);
    . . WHEN(ACTCDRM)
    . .   SCB.#FSM_SESS = 'FSM_SESS_SSCP_SSCP_PRI_OR_SEC';
    . . WHEN(ACTLU)
    . .   DO;
    . .   . IF CB_TYPE = HALF_SESS THEN
    . .   .   DO;
```

```
      . .   .   . IF MUCB.DIRECTION = SEND THEN
      . .   .   .   SCB.@FSM_SESS = 'FSM_SESS_CP_LU_PRI';
      . .   .   . ELSE
      . .   .   .   SCB.@FSM_SESS = 'FSM_SESS_CP_LU_SEC';
      . .   .   END;
      . .   . ELSE
      . .   .   SCB.@FSM_SESS = 'FSM_SESS_BF_CP_LU';
      . .   END;
      . . WHEN(ACTPU)
      . .   DO;
      . .   . IF CB_TYPE = HALF_SESS THEN
      . .   .   DO;
      . .   .   . IF MUCB.DIRECTION = SEND THEN
      . .   .   .   SCB.@FSM_SESS = 'FSM_SESS_CP_PU_PRI';
      . .   .   . ELSE
      . .   .   .   SCB.@FSM_SESS = 'FSM_SESS_CP_PU_SEC';
      . .   .   END;
      . .   . ELSE
      . .   .   DO;
      . .   .   . IF NRCB.RESOURCE_TYPE = T2 THEN
      . .   .   .   SCB.@FSM_SESS = 'FSM_SESS_BF_CP_PU_T2';
      . .   .   . ELSE
      . .   .   .   SCB.@FSM_SESS = 'FSM_SESS_BF_CP_PU_T1';
      . .   .   END;
      . .   END;
      . . WHEN(BIND)
      . .   DO;
      . .   . IF CB_TYPE = HALF_SESS THEN
      . .   .   DO;
      . .   .   . IF MUCB.DIRECTION = SEND THEN
      . .   .   .   SCB.@FSM_SESS = 'FSM_SESS_LU_LU_PRI';
      . .   .   . ELSE
      . .   .   .   SCB.@FSM_SESS = 'FSM_SESS_LU_LU_SEC';
      . .   .   END;
      . .   . ELSE
      . .   .   SCB.@FSM_SESS = 'FSM_SESS_BF_LU_LU';
      . .   END;
      . END;
    END;
  RETURN;
END SCB_CREATE;
```

```
SCB_DISCARD: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  DISCARDS  THE SCB  (THAT INCLUDES  RESETTING  ALL LOCALLY  SUPPORTED  │
   │                 HALF-SESSION  OR  BOUNDARY FUNCTION  SUPPORTED  HALF-SESSION  FSM'S,  │
   │                 REMOVING ALL  ENTRIES FROM THE QUEUES  AND LISTS, AND  RESETTING ALL  │
   │                 VARIABLES FOR THIS  SESSION) AND REDUCES THE COUNT  OF SESSIONS THAT  │
   │                 ARE CONNECTED TO A VRCB.  IF THE  VRCB SESSION COUNT GOES TO ZERO, A  │
   │                 "SESS_COUNT=0" SIGNAL IS SENT TO THE VR_MGR.                          │
   │                                                                                       │
   │      INPUT:     ADDRESS OF THE CURRENT SCB                                            │
   │                                                                                       │
   │      OUTPUT:    THE SCB IS DISCARDED, AND A  "SESS_COUNT=0" SIGNAL IS SENT TO    THE  │
   │                 VR_MGR IF THE SESSION COUNT FOR A VRCB GOES TO ZERO.                  │
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  FIND VRCB IN VRCB_LIST WHERE(VRCB_PTR = SCB.VRCBPTR);
  IF VRCB_PTR ¬= NULL THEN
    DO;
    . VRCB.SESS_COUNT = VRCB.SESS_COUNT - 1;
    . IF VRCB.SESS_COUNT <= 0 THEN
    .   SEND 'SESS_COUNT=0' TO PU.SVC_MGR.PC_ROUTE_MGR.RCV;      /* CHAPTER 12      */
    END;
  REMOVE SCB FROM SCB_LIST DISCARD;
  RETURN;
END SCB_DISCARD;



UPM_GET_SEQ_ID: PROCEDURE RETURNS(BIT(64));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  RETRIEVES  THE   ACTIVATION  REQUEST/RESPONSE   SEQUENCE  IDENTIFIER  │
   │                 CONTAINED IN CONTROL VECTOR X'0C' FROM THE ACTCDRM, RSP(ACTCDRM), OR  │
   │                 ACTPU.                                                                │
   │                                                                                       │
   │      INPUT:     THE CURRENT MU--ACTPU, ACTCDRM, RSP(ACTCDRM)                          │
   │                                                                                       │
   │      OUTPUT:    THE ACTIVATION REQUEST/RESPONSE SEQUENCE IDENTIFIER CONTAINED IN THE   │
   │                 CURRENT MU                                                            │
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                        /* FUNCTION AS DESCRIBED  */
  RETURN;
END UPM_GET_SEQ_ID;



UPM_PS_PROFILE: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  SAVES PS  PROFILE AND ES USAGE  INFORMATION.  THIS UPM IS  CALLED BY  │
   │                 SESSACT.REQUEST.                                                      │
   │                                                                                       │
   │      INPUT:     PS PROFILE AND PS USAGE FIELDS                                        │
   │                                                                                       │
   │      OUTPUT:    NONE                                                                  │
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                        /*FUNCTION AS DESCRIBED  */
  RETURN;
END UPM_PS_PROFILE;
```

```
CREATE_DEACTIVATION_RSP: PROCEDURE RETURNS(PTR);

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS ROUTINE IS CALLED BY  CSC_MGR.T4_OR_T5.SEND OR CSC_MGR.RCV WHEN
  │                 A DEACTIVATION  REQUEST INDICATING THE  CLEANUP OPTION  IS RECEIVED.
  │                 THIS FUNCTION SETS THE APPROPRIATE FIELDS  TO MAKE THE RESPONSE LOOK
  │                 AS IF IT CAME FROM THE SESSION PARTNER.
  │
  │      INPUT:     MU_PTR POINTING TO THE DEACTIVATION REQUEST
  │
  │      OUTPUT:    NEW PTR POINTING TO THE POSITIVE RESPONSE
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RSP_PTR PTR;
  CREATE RSP_PTR->MU;
  RRI = RSP;
  MUCB.DIRECTION = RECEIVE;
  SELECT ANYORDER(FID);
  . WHEN(FID4)
  .   DO;
  .   . RSP_PTR->OSAF = DSAF;
  .   . RSP_PTR->OEF = DEF;
  .   . RSP_PTR->DSAF = OSAF;
  .   . RSP_PTR->DEF = OEF;
  .   END;
  . WHEN(FID2)
  .   DO;
  .   . RSP_PTR->OAFPRIME = DAFPRIME;
  .   . RSP_PTR->DAFPRIME = OAFPRIME;
  .   END;
  . WHEN(FID3)
  .   RSP_PTR->LSID = LSID;
  END;
  RETURN(RSP_PTR);
END CREATE_DEACTIVATION_RSP;
```

```
SON_TYPE: PROCEDURE RETURNS(BIT(8));

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  RETRIEVES THE  TYPE OF SON  CONDITION FROM THE  CURRENT DEACTIVATION │
   │                 REQUEST.                                             │
   │                                                                      │
   │      INPUT:     CURRENT MU_PTR                                       │
   │                                                                      │
   │      OUTPUT:    THE TYPE OF SON CODE CONTAINED IN THE MU             │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */

  SELECT ANYORDER(RQ_CODE);
  .
  . WHEN(UNBIND)
  .   RETURN(UNBIND_RQ.SON_CAUSE);
  .
  . WHEN(DACTPU)
  .   RETURN(DACTPU_RQ.SON_CAUSE);
  .
  . WHEN(DACTLU)
  .   RETURN(DACTLU_RQ.SON_CAUSE);
  .
  . WHEN(DACTCDRM)
  .   RETURN(DACTCDRM_RQ.SON_CAUSE);
  .
  END;
END SON_TYPE;
```

```
FSM_SESS_SSCP_SSCP_PRI_OR_SEC: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO REMEMBER THE STATE OF THE SSCP-SSCP HALF-SESSION  │
│                                                                      │
│      NOTE:      THE  CANNOT-OCCURS  IN  THE  RESET  STATE  ARE  CHECKED  BY
│                 FUNCTION_SUPPORTED PROCEDURE (PAGE 13-53) AND  A SENSE CODE 8005--NO
│                 SESSION-- IS GENERATED.                              │
└──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES------> / INPUTS	RESET 01	PEND ACTIVE SEND 02	PEND ACTIVE RCV 03	ACTIVE 04	PEND RESET SEND 05	PEND RESET RCV 06	PEND RESET SON 07
R, RQ,ACTCDRM	3(S)	3(S)	-(S)	3(S)	3(S)	>	>
R,-RSP,ACTCDRM, 080D	/	-(D)	-(D)	>	>	>	>
R,-RSP,ACTCDRM,¬080D	/	1	>	>	>	>	>
R,+RSP,ACTCDRM	/	4(T)	>	>	>	>	>
S, RQ,ACTCDRM	2(S)	>	>	>	>	>	>
S,+RSP,ACTCDRM	/	>	4(T)	>	>	>	>
S,-RSP,ACTCDRM	/	>	1(H)	>	>	>	>
R, RQ,DACTCDRM,¬SON	/	6	6	6	6	-	>
R, RQ,DACTCDRM,SON	/	7	7	7	7	7	>
R,±RSP,DACTCDRM	/	>	>	>	1	1	>
S, RQ,DACTCDRM	/	5	5	5	-	>	>
S,+RSP,DACTCDRM	/	>	>	>	>	1(H)	1(HK)

OUTPUT CODE	FUNCTION
D	SCB.HALF_SESSION = SEC; /* CONTENTION LOSER */ DISCARD MU;
H	CALL SCB_DISCARD; /* PAGE 13-88 */
HK	CALL SCB_DISCARD; /* PAGE 13-88 */ DISCARD MU;
S	CALL SESSACT.REQUEST; /* PAGE 13-66 */
T	CALL SESSACT.RESPONSE; /* PAGE 13-68 */

```
END FSM_SESS_SSCP_SSCP_PRI_OR_SEC;
```

```
FSM_SESS_CP_PU_PRI: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO REMEMBER THE STATE OF THE PRIMARY SSCP-PU HALF-SESSION
   │
   │      NOTE:      THE CANNOT-OCCURS IN THE RESET STATE ARE DETERMINED IN THE
   │                 FUNCTION_SUPPORTED PROCEDURE (PAGE 13-53) AND SENSE CODE 8005--NO
   │                 SESSION--IS GENERATED.
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES------> / INPUT	RESET 01	PEND ACTIVE 02	ACTIVE 03	PEND RESET 04	PEND VR OUT 05
S, RQ,ACTPU	2(S)	>	>	>	>
R,+RSP,ACTPU	/	3(T)	>	>	>
R,-RSP,ACTPU	/	1	>	>	>
S, RQ,DACTPU	/	4	4	-	>
R, RQ,DACTPU, VR_OUT	/	5	5	5	>
S,+RSP,DACTPU	/	>	>	>	1(HK)
R,±RSP,DACTPU	/	>	>	1	>

OUTPUT CODE	FUNCTION
HK	CALL SCB_DISCARD; /* PAGE 13-88 */ DISCARD MU;
S	CALL SESSACT.REQUEST; /* PAGE 13-66 */
T	CALL SESSACT.RESPONSE; /* PAGE 13-68 */

```
END FSM_SESS_CP_PU_PRI;

FSM_SESS_CP_PU_SEC: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO REMEMBER THE STATE OF THE SECONDARY SSCP-PU HALF-SESSION
   │
   │      NOTE:      THE CANNOT-OCCURS IN THE RESET STATE ARE DETERMINED IN THE
   │                 FUNCTION_SUPPORTED PROCEDURE (PAGE 13-53) AND SENSE CODE 8005--NO
   │                 SESSION--IS GENERATED.
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES------> / INPUT	RESET 01	PEND ACTIVE 02	ACTIVE 03	PEND RESET 04	PEND SON 05
R, RQ,ACTPU	2(S)	>	2(S)	>	>
S,+RSP,ACTPU,COLD	/	3(TK)	>	>	>
S,+RSP,ACTPU,ERP	/	3(T)	>	>	>
S,-RSP,ACTPU	/	1(H)	>	>	>
R, RQ,DACTPU,¬SON	/	4	4	-	>
R, RQ,DACTPU,SON	/	5	5	5	>
S,+RSP,DACTPU	/	>	>	1(HS)	1(HK)

OUTPUT CODE	FUNCTION
H	CALL SCB_DISCARD; /* PAGE 13-88 */
HK	CALL SCB_DISCARD; /* PAGE 13-88 */ DISCARD MU;
HS	SEND 'SSCP_GONE' TO CSC_MGR.SON; /* PAGE 13-47 */
S	CALL SESSACT.REQUEST; /* PAGE 13-66 */
T	CALL SESSACT.RESPONSE; /* PAGE 13-68 */
TK	CALL SESSACT.RESPONSE; /* PAGE 13-68 */ SEND 'HIERARCHICAL_RESET' TO CSC_MGR.SON; /* PAGE 13-47 */

```
END FSM_SESS_CP_PU_SEC;
```

```
FSM_SESS_CP_LU_PRI: FSM_DEFINITION CONTEXT(SCB);
```

/*

```
     FUNCTION:  TO REMEMBER THE STATE OF THE PRIMARY SSCP-LU HALF-SESSION

     NOTE:      THE  SHOULD-NOT-OCCURS IN  THE  RESET STATE  ARE  DETERMINED IN  THE
                FUNCTION_SUPPORTED PROCEDURE  (PAGE 13-53)  AND SENSE  CODE 8005--NO
                SESSION--IS GENERATED.
```

*/

STATE NAMES------> / INPUT	RESET 01	PEND ACTIVE 02	ACTIVE 03	PEND RESET 04	PEND VR OUT 05
S, RQ,ACTLU	2(S)	>	>	>	>
R,+RSP,ACTLU	/	3(T)	>	>	>
R,-RSP,ACTLU	/	1	>	>	>
S, RQ,DACTLU	/	4	4	-	>
R, RQ,DACTLU, VR_OUT	/	5	5	5	>
S,+RSP,DACTLU	/	>	>	>	1(HK)
R,±RSP,DACTLU	/	>	>	1	>

OUTPUT CODE	FUNCTION	
HK	CALL SCB_DISCARD; DISCARD MU;	/* PAGE 13-88 */
S	CALL SESSACT.REQUEST;	/* PAGE 13-66 */
T	CALL SESSACT.RESPONSE;	/* PAGE 13-68 */

```
END FSM_SESS_CP_LU_PRI;
```

```
FSM_SESS_CP_LU_SEC: FSM_DEFINITION CONTEXT(SCB);
```

/*

```
     FUNCTION:  TO REMEMBER THE STATE OF THE SECONDARY SSCP-LU HALF-SESSION

     NOTE:      THE  CANNOT-OCCURS  IN  THE  RESET  STATE  ARE  DETERMINED  IN  THE
                FUNCTION_SUPPORTED PROCEDURE  (PAGE 13-53)  AND SENSE  CODE 8005--NO
                SESSION--IS GENERATED.
```

*/

STATE NAMES------> / INPUT	RESET 01	PEND ACTIVE 02	ACTIVE 03	PEND RESET 04	PEND SON 05
R, RQ,ACTLU	2(S)	>	2(S)	>	>
S,+RSP,ACTLU,COLD	/	3(TK)	>	>	>
S,+RSP,ACTLU,ERP	/	3(T)	>	>	>
S,-RSP,ACTLU	/	1(H)	>	>	>
R, RQ,DACTLU,¬SON	/	4	4	-	>
R, RQ,DACTLU,SON	/	5	5	5	>
S,+RSP,DACTLU	/	>	>	1(HS)	1(HK)

OUTPUT CODE	FUNCTION	
H	CALL SCB_DISCARD;	/* PAGE 13-88 */
HK	CALL SCB_DISCARD; DISCARD MU;	/* PAGE 13-88 */
HS	SEND 'SSCP_GONE' TO CSC_MGR.SON;	/* PAGE 13-47 */
S	CALL SESSACT.RRQUEST;	/* PAGE 13-66 */
T	CALL SESSACT.RESPONSE;	/* PAGE 13-68 */
TK	CALL SESSACT.RESPONSE; SEND 'HIERARCHICAL_RESET' TO CSC_MGR.SON;	/* PAGE 13-68 */ /* PAGE 13-47 */

```
END FSM_SESS_CP_LU_SEC;
```

```
FSM_SESS_LU_LU_PRI: FSM_DEFINITION CONTEXT(SCB);
```

/*

FUNCTION: TO REMEMBER THE STATE OF THE PRIMARY LU-LU HALF-SESSION

NOTE: THE CANNOT-OCCURS IN THE RESET STATE ARE DETERMINED IN THE FUNCTION_SUPPORTED PROCEDURE (PAGE 13-53) AND SENSE CODE 8005--NO SESSION--IS GENERATED.

*/

STATE NAMES----> / INPUT	RESET 01	PEND ACT 02	ACTIVE 03	PEND RSET NRSP 04	PEND RSET PRI 05	PEND RSET SEC 06	PEND RSET BOTH 07	PEND SON 08
S, RQ,BIND	2(S)	>	>	>	>	>	>	>
R,+RSP,BIND	/	3(T)	>	>	>	>	>	>
R,-RSP,BIND,¬084D,¬084E	/	1	>	>	>	>	>	>
R,-RSP,BIND, 084D,084E	/	4	>	>	>	>	>	>
S, RQ,UNBIND	/	5	5	5	-	>	-	>
R,±RSP,UNBIND	/	>	>	>	1	>	1	>
R, RQ,UNBIND,¬SON	/	>	6	6	7	-	-	>
S,±RSP,UNBIND	/	>	>	>	>	1(H)	1(H)	1(HK)
R, RQ,UNBIND,SON	/	8	8	8	8	8	8	>

OUTPUT CODE	FUNCTION
H	CALL SCB_DISCARD; /* PAGE 13-88 */
HK	CALL SCB_DISCARD; /* PAGE 13-88 */ DISCARD MU;
S	CALL SESSACT.REQUEST; /* PAGE 13-66 */
T	CALL SESSACT.RESPONSE; /* PAGE 13-68 */

```
END FSM_SESS_LU_LU_PRI;
```

```
FSM_SESS_LU_LU_SEC: FSM_DEFINITION CONTEXT(SCB);
```

/*

FUNCTION: TO REMEMBER THE STATE OF THE SECONDARY LU-LU HALF-SESSION

NOTE: THE CANNOT-OCCURS IN THE RESET STATE ARE DETERMINED IN THE FUNCTION_SUPPORTED PROCEDURE (PAGE 13-53) AND SENSE CODE 8005--NO SESSION--IS GENERATED.

*/

STATE NAMES----> / INPUT	RESET 01	PEND ACT 02	ACTIVE 03	PEND RSET PRI 04	PEND RSET SEC 05	PEND RSET BOTH 06	PEND SON 07
R, RQ, BIND	2(S)	>	>	>	>	>	>
S,+RSP, BIND	/	3(T)	>	>	>	>	>
S,-RSP, BIND	/	1(H)	>	>	>	>	>
R, RQ,UNBIND,¬SON	/	4	4	-	6	-	>
R, RQ,UNBIND, SON	/	7	7	7	7	7	>
S,±RSP,UNBIND	/	>	>	1(H)	>	1(H)	1(HK)
S, RQ,UNBIND	/	>	5	>	-	-	>
R,±RSP,UNBIND	/	>	>	>	1	1	>

OUTPUT CODE	FUNCTION
H	CALL SCB_DISCARD; /* PAGE 13-88 */
HK	CALL SCB_DISCARD; /* PAGE 13-88 */ DISCARD MU;
S	CALL SESSACT.REQUEST; /* PAGE 13-66 */
T	CALL SESSACT.RESPONSE; /* PAGE 13-68 */

```
END FSM_SESS_LU_LU_SEC;
```

```
FSM_SESS_BF_CP_PU_T1: FSM_DEFINITION CONTEXT(SCB);
                                                                        /*
```

FUNCTION: TO REMEMBER THE STATE OF THE SSCP-PU_T1 HALF-SESSION THAT IS SUPPORTED BY THE BOUNDARY FUNCTION IN THE NODE.

NOTES:

1. THE CANNOT-OCCURS IN THE RESET STATE ARE DETERMINED IN THE FUNCTION_SUPPORTED PROCEDURE (PAGE 13-53) AND SENSE CODE 8005--NO SESSION--IS GENERATED.
2. DACTPU IS NOT ROUTED TO PU_T1'S. THE BF.PU.SVC_MGR GENERATES THE RSP(DACTPU) ON BEHALF OF THE PU_T1.

```
                                                                        */
```

STATE NAMES------> / INPUT	RESET 01	PEND ACTIVE 02	ACTIVE 03	PEND RESET 04	PEND SON 05
R, RQ,ACTPU	2(S)	>	2(S)	>	>
S,+RSP,ACTPU,COLD	/	3(TK)	>	>	>
S,+RSP,ACTPU,ERP	/	3(T)	>	>	>
S,-RSP,ACTPU	/	1(H)	>	>	>
R, RQ,DACTPU,¬SON	/	4	4(HS)	-	>
R, RQ,DACTPU,SON	/	5	5	5	>
S,±RSP,DACTPU	/	>	>	1(H)	1(HX)
'RESET'	/	1(H)	1(H)	1(H)	1(H)

OUTPUT CODE	FUNCTION
H	CALL SCB_DISCARD; /* PAGE 13-88 */
HS	SEND 'SSCP_GONE' TO CSC_MGR.SON; /* PAGE 13-47 */
HX	CALL SCB_DISCARD; /* PAGE 13-88 */ DISCARD MU;
S	CALL SESSACT.REQUEST; /* PAGE 13-66 */
T	CALL SESSACT.RESPONSE; /* PAGE 13-68 */
TK	CALL SESSACT.RESPONSE; /* PAGE 13-68 */ SEND 'HIERARCHICAL_RESET' TO CSC_MGR.SON; /* PAGE 13-47 */

```
END FSM_SESS_BF_CP_PU_T1;
```

```
FSM_SESS_BF_CP_PU_T2: FSM_DEFINITION CONTEXT(SCB);
                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO  REMEMBER  THE  STATE  OF THE  SSCP-PU_T2  HALF-SESSION  THAT  IS  │
   │                 SUPPORTED BY BOUNDARY FUNCTION                       │
   │                                                                      │
   │      NOTE:      THE  CANNOT-OCCURS  IN  THE  RESET  STATE  ARE  DETERMINED  IN  THE  │
   │                 FUNCTION_SUPPORTED PROCEDURE  (PAGE 13-53)  AND SENSE  CODE 8005--NO  │
   │                 SESSION--IS GENERATED.                               │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES------>	RESET	PEND ACT	ACTIVE	PEND RESET	PEND SON
INPUT	01	02	03	04	05
R, RQ,ACTPU,PRI	2(S)	>	2(S)	>	>
S, RQ,ACTPU,SEC	/	-	>	>	>
R,±RSP,ACTPU,SEC	/	-	>	-(N)	-(N)
S,+RSP,ACTPU,PRI,COLD	/	3(TK)	>	>	>
S,+RSP,ACTPU,PRI,ERP	/	3(T)	>	>	>
S,-RSP,ACTPU,PRI	/	1(H)	>	>	>
R, RQ,DACTPU,¬SON,PRI	/	4	4(HS)	-	>
R, RQ,DACTPU,SON,PRI	/	5	5	5	>
S, RQ,DACTPU,SEC	/	>	>	-	1(HX)
R,±RSP,DACTPU,SEC	/	>	>	-	-
S,±RSP,DACTPU,PRI	/	>	>	1(H)	1(HX)
'RESET'	/	1(H)	1(H)	1(H)	1(H)

OUTPUT CODE	FUNCTION
H	CALL SCB_DISCARD; /* PAGE 13-88 */
HS	SEND 'SSCP_GONE' TO CSC_MGR.SON; /* PAGE 13-47 */
HX	CALL SCB_DISCARD; /* PAGE 13-88 */ DISCARD MU;
N	DISCARD MU;
S	CALL SESSACT.REQUEST; /* PAGE 13-66 */
T	CALL SESSACT.RESPONSE; /* PAGE 13-68 */
TK	CALL SESSACT.RESPONSE; /* PAGE 13-68 */ SEND 'HIERARCHICAL_RESET' TO CSC_MGR.SON; /* PAGE 13-47 */

```
END FSM_SESS_BF_CP_PU_T2;
```

```
FSM_SESS_BF_CP_LU: FSM_DEFINITION CONTEXT(SCB);
```

```
/*
   FUNCTION:  TO REMEMBER THE STATE OF THE  SSCP-LU HALF-SESSION THAT IS SUPPORTED
              BY BOUNDARY FUNCTION

   NOTE:      THE  CANNOT-OCCURS  IN  THE  RESET  STATE  ARE  DETERMINED  IN  THE
              FUNCTION_SUPPORTED PROCEDURE  (PAGE 13-53)  AND SENSE  CODE 8005--NO
              SESSION--IS GENERATED.
*/
```

STATE NAMES------>	RESET	PEND ACT	ACTIVE	PEND RSET	PEND SON
INPUT	01	02	03	04	05
R, RQ,ACTLU,PRI	2(S)	>	2(S)	>	>
S, RQ,ACTLU,SEC	/	-	>	>	>
R,±RSP,ACTLU,SEC	/	-	>	-(N)	-(N)
S,+RSP,ACTLU,PRI,COLD	/	3(TK)	>	>	>
S,+RSP,ACTLU,PRI,ERP	/	3(T)	>	>	>
S,-RSP,ACTLU,PRI	/	1(H)	>	>	1(HX)
R, RQ,DACTLU,¬SON,PRI	/	4	4	-	>
R, RQ,DACTLU,SON,PRI	/	5	5	5	>
S, RQ,DACTLU,SEC	/	>	>	-(TK)	1(HC)
R,±RSP,DACTLU,SEC	/	>	>	-	-
S,±RSP,DACTLU,PRI	/	>	>	1(H)	1(HX)
'RESET'	-	1(H)	1(H)	1(H)	1(H)

OUTPUT CODE	FUNCTION
H	CALL SCB_DISCARD; /* PAGE 13-88 */
HX	CALL SCB_DISCARD; /* PAGE 13-88 */ DISCARD MU;
HC	CALL SCB_DISCARD; /*PAGE 13-88 */ IF NRCB.RESOURCE_TYPE = PU_T2 THEN /* DON'T ROUTE DACTLU TO SECONDARY FOR SON */ /* SEND DACTLU TO PU_T1'S SINCE THE T1 DIDN'T GET THE DACTPU ¦ ACTPU */ DISCARD MU;
S	CALL SESSACT.REQUEST; /* PAGE 13-66 */
T	CALL SESSACT.RESPONSE; /* PAGE 13-68 */
TK	SEND 'HIERARCHICAL_RESET' TO CSC_MGR.SON; /* PAGE 13-47 */
N	DISCARD MU;

```
END FSM_SESS_BF_CP_LU;
```

```
FSM_SESS_BF_LU_LU: FSM_DEFINITION CONTEXT(SCB);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  TO REMEMBER THE STATE OF THE LU-LU HALF-SESSION THAT IS SUPPORTED BY
   │                 BOUNDARY FUNCTION                                    │
   │                                                                      │
   │      NOTE:      THE  CANNOT-OCCURS  IN  THE  RESET  STATE  ARE  DETERMINED  IN  THE
   │                 FUNCTION_SUPPORTED PROCEDURE  (PAGE 13-53)  AND SENSE  CODE 8005--NO
   │                 SESSION--IS GENERATED.                               │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
```

STATE NAMES----->	RESET	PEND ACT	ACTIVE	PEND UNBND PRI	PEND UNBND SEC	PEND UNBND RSP	PEND RESET 084D	PEND VR OUT	PEND REX OUT
INPUT	01	02	03	04	05	06	07	08	09
R, RQ, BIND,PRI	2(S)	>	>	>	>	>	>	>	>
S, RQ, BIND,SEC	/	-	>	>	>	>	>	>	>
R,±RSP, BIND,SEC	/	-	>	-(D)	>	>	>	>	>
S,+RSP, BIND,PRI	/	3(T)	>	>	>	>	>	>	>
S,-RSP, BIND,PRI,¬084D	/	1(H)	>	>	>	>	>	>	>
S,-RSP, BIND,PRI, 084D	/	7	>	>	>	>	>	>	>
R, RQ,UNBIND,PRI,¬VR_OUT	/	4	4	-	6	-	4	>	-(D)
S, RQ,UNBIND,SEC	/	>	6	-	>	-	>	-	>
R,RQ,UNBIND,SEC,¬REX_OUT	/	5	5	6	-	-	-(D)	-(D)	>
S,RQ,UNBIND,PRI	/	>	6	>	-	-	>	>	-
R,±RSP,UNBIND,SEC	/	>	>	-	>	-	-	-	>
S,±RSP,UNBIND,PRI	/	>	>	1(H)	>	1(H)	1(HK)	1(HK)	1(H)
R,±RSP,UNBIND,PRI	/	>	>	>	-	-	>	>	-
S,±RSP,UNBIND,SEC	/	>	>	>	1(H)	1(H)	>	>	1(HK)
R, RQ,UNBIND,PRI,VR_OUT	/	8	8	8(D)	8	8(D)	8	>	1(HK)
R, RQ,UNBIND,SEC,REX_OUT	/	9	9	9	9(D)	9(D)	1(HK)	1(HK)	>

OUTPUT CODE	FUNCTION
D	DISCARD MU;
H	CALL SCB_DISCARD; /* PAGE 13-88 */
HK	CALL SCB_DISCARD; /* PAGE 13-88 */ DISCARD MU;
S	CALL SESSACT.REQUEST; /* PAGE 13-66 */
T	CALL SESSACT.RESPONSE; /* PAGE 13-68 */

```
END FSM_SESS_BF_LU_LU;

FSM_INPUT_DEFINITION:

 ACTCDRM                 RQ_CODE = ACTCDRM;
 ACTLU                   RQ_CODE = ACTLU;
 ACTPU                   RQ_CODE = ACTPU;
 BIND                    RQ_CODE = BIND;
 COLD                    ACT_RU.TYPE_ACTIVATION  = COLD;/* PAGE 13-99 */
 DACTCDRM                RQ_CODE = DACTCDRM;
 DACTLU                  RQ_CODE = DACTLU;
 DACTPU                  RQ_CODE = DACTPU;
 ERP                     ACT_RU.TYPE_ACTIVATION  = ERP; /* PAGE 13-99 */
 PRI                     (MUCB.DIRECTION = RECEIVE & NRCB.ELEMENT_ADDRESS = DEF) |
                         (MUCB.DIRECTION = SEND & NRCB.ELEMENT_ADDRESS = OEF);
 R                       MUCB.DIRECTION = RECEIVE;
 REX_OUT                 SON_TYPE = REX_INOP; /* PAGE 13-90 */
 'RESET'                 INPUT('RESET');
±RSP                     RRI = RSP;
-RSP                     RRI = RSP & RTI = NEGATIVE;
+RSP                     RRI = RSP & RTI = POSITIVE;
 RQ                      RRI = RQ;
 S                       MUCB.DIRECTION = SEND;
 SEC                     (MUCB.DIRECTION = RECEIVE & NRCB.ELEMENT_ADDRESS = OEF) |
                         (MUCB.DIRECTION = SEND & NRCB.ELEMENT_ADDRESS = DEF);
 SON                     DEACT_RU.TYPE_DEACTIVATION = SESS_OUT;/* PAGE 13-99 */
 UNBIND                  RQ_CODE = UNBIND;
 VR_OUT                  SON_TYPE = (VR_INOP | DACTVR_FORCED); /* PAGE 13-90 */
 080D                    SNC = X'080D';
 084D                    SNC = X'084D';
 084E                    SNC = X'084E';
END FSM_INPUT_DEFINITION;
```

```
DECLARE_LOCAL_VARIABLES: PROCEDURE;

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE HOUSES THE DECLARES FOR LOCAL VARIABLES.      │
  │                                                                              │
  │      INPUT:     NONE                                                         │
  │                                                                              │
  │      OUTPUT:    NONE                                                         │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL CB_TYPE BIT(1);                    /* 0 = BF_SESS, 1 = HALF_SESS          */

  DCL 1 ACT_RU BASED(ADDR(RU)),          /* USED IN FSM INPUT DEFINITIONS       */
        2 RESERVED BIT(8),                                /* REQUEST CODE       */
        2 RESERVED BIT(4),                                /* FORMAT             */
        2 TYPE_ACTIVATION BIT(4);

  DCL 1 DEACT_RU BASED(ADDR(RU)),        /* USED IN FSM INPUT DEFINITIONS       */
        2 RESERVED BIT(8),                                /* REQUEST CODE       */
        2 TYPE_DEACTIVATION BIT(8);

END DECLARE_LOCAL_VARIABLES;
```

This page intentionally left blank

APPENDIX A. NODE DATA STRUCTURES AND CONSTANTS

This appendix defines all node data structures and all constants used in the FAPL procedures. This includes information about the node's resources, the half-sessions, the SSCP's domain resources, as well as routing tables for path control. Some of the structures are initialized during system definition, while others can be created as needed. Some of the structures may also be modified dynamically by commands such as SETCV and RNAA. The following structures are defined:

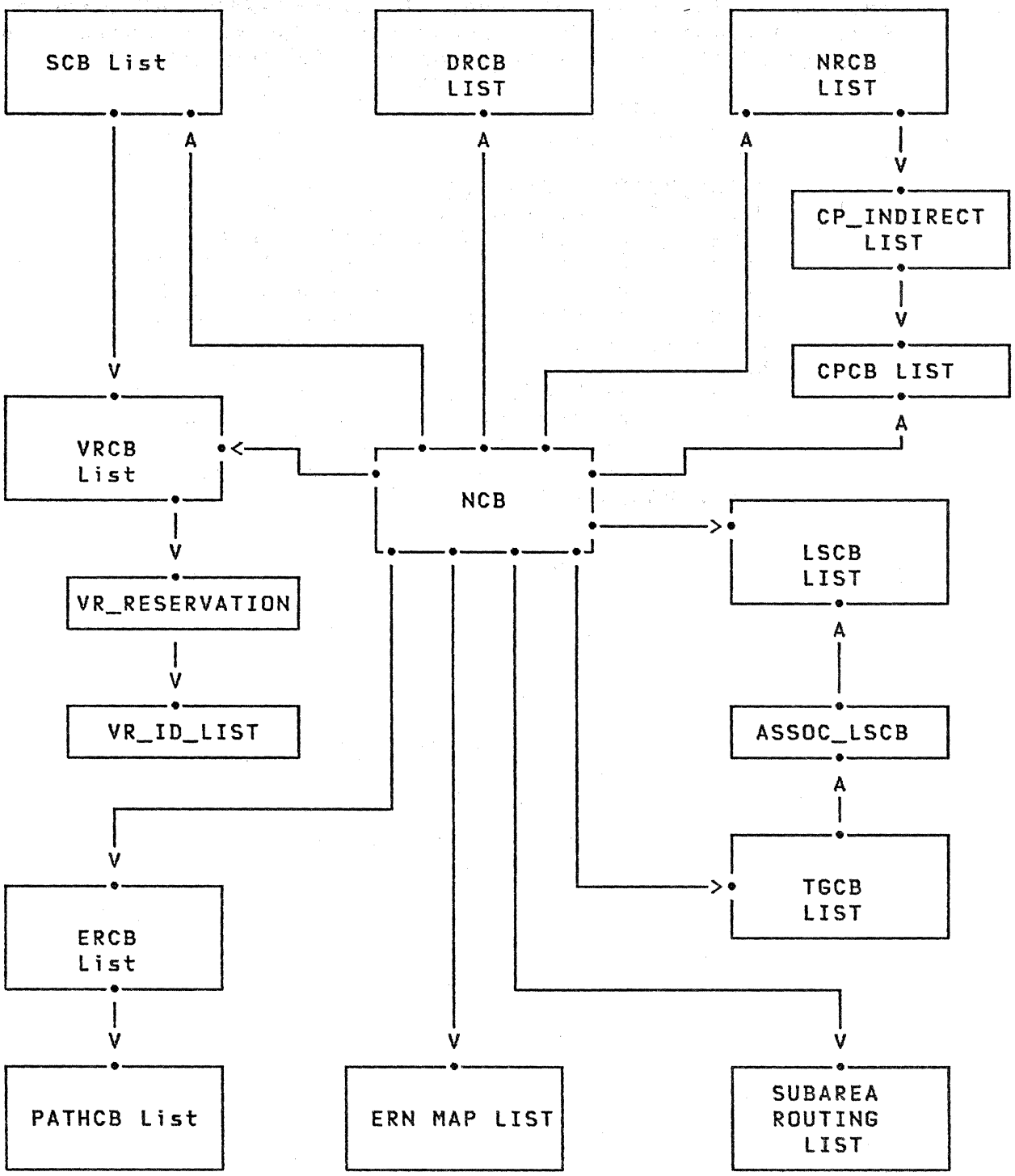


Figure A-1. Structure of Node Control Blocks

The following list of control blocks contain associated FSMs. A pointer to the current entry for each control block is part of the implicit environment (See Appendix C.)

```
CONTROL_BLOCK_DEFINITION:
  NCB       /* NODE CONTROL BLOCK                          */
  LSCB      /* LINK STATION CONTROL BLOCK                  */
  TGCB      /* TRANSMISSION GROUP CONTROL BLOCK            */
  ERCB      /* EXPLICIT ROUTE CONTROL BLOCK                */
  VRCB      /* VIRTURAL ROUTE CONTROL BLOCK                */
  SCB       /* SESSION CONTROL BLOCK                       */
  NRCB      /* NODE RESOURCE CONTROL BLOCK                 */
  DRCB      /* DOMAIN RESOURCE CONTROL BLOCK               */
  CPCB      /* CONTROL POINT CONTROL BLOCK                 */
  TCCB      /* TRANMISSION CONTROL CONTROL BLOCK           */
  PCCB      /* PATH CONTROL CONTROL BLOCK                  */
END CONTROL_BLOCK_DEFINITION;
```

This page intentionally left blank

NODE CONTROL BLOCK

The node control block contains information that is common to the entire node. This includes the node subarea address and the PUCP element address. In addition, major lists are referenced by pointers stored in this control block. Default pointers that are used to refer to individual entries are also defined. The current state values for finite-state machines relating to the node's resources are contained in this control block.

PATH CONTROL CONTROL BLOCK

A single copy of this data structure is maintained by each node. It provides working storage and attributes required by path control components to send and receive PIUs over route extensions.

NODE RESOURCE CONTROL BLOCK LIST AND CPCB LIST

The node resource list contains one entry for each supported resource, of which seven categories are defined: PU, LU, SSCP, BF.PU, BF.LU, link, and adjacent link station. Each node resource is identified in the node resource list by its element address. Each entry contains information regarding the resource it represents (e.g., link entries contain a share-limit parameter) and the element address of a hierarchically superior resource associated with it. For simplicity of structure, not all the architectural relationships between resources are specified in all directions. For example, a BF.PU entry does not contain the address of each associated BF.LU; however, the set of associated BF.LUs can be obtained by scanning the node resource list for BF.LU entries that are associated with the BF.PU of interest.

The node resource list is referred to by numerous node components. For example, the PU.SVC_MGR.CSC_MGR uses it to determine the type of NAU associated with a given element address, and the PU.SVC_MGR.NS uses it for determining what to reset when DACTPU is received.

Each resource of the node may be controlled by one or more control points (CPs). A list of pointers to the current control points for each resource is maintained in the CP_INDIRECT list that is anchored in the node resource entry. Each entry in this list points to an entry in the CPCB list. The CPCB list entry contains the current status of the CP-PU session with respect to the SDT request, and the CP's full network address.

SESSION CONTROL BLOCK

The SCBs contain session parameters that are initialized and referred to throughout the book. There are two kinds of SCBs, as distinguished by the field SCB_TYPE: half-session control blocks (HSCBs) and boundary function session control blocks (BFSCBs). The SCB is defined in two parts; a header, which applies to both half-session and boundary function session control blocks, and an extension, that applies only to half-sessions. SCBs are created at session activation and destroyed at session deactivation (see Chapter 13 for details).

TRANSMISSION CONTROL CONTROL BLOCK

The variables associated with session-level pacing and maximum RU size are maintained in this structure. All half-sessions have one associated TCCB. Boundary function support for a session session requires two TCCB's, one for each stage for two stage pacing.

DOMAIN RESOURCE CONTROL BLOCK LIST

Each entry contains information related to a single resource that the SSCP can activate or control.

LINK STATION CONTROL BLOCK LIST

Data link control maintains information about the status of each link and its adjacent link stations in this control block.

TRANSMISSION GROUP CONTROL BLOCK LIST

The TGCB contains the definition and current status of a transmission group.

Each transmission group is made up of one or more adjacent link stations. A list of the LSCBs for these stations is maintained in each TGCB.

A list of the PIU's that are to be transmitted as a single BTU is maintained in each TGCB also.

VIRTUAL ROUTE CONTROL BLOCK LIST

Each VRCB is used to maintain the definition and status of a virtual route.

VIRTUAL ROUTE RESERVATION LIST

This list contains information concerning session activation requests that are waiting on the activation of a VR.

EXPLICIT ROUTE CONTROL BLOCK LIST

Each ERCB is used to maintain the definition and status related to an explicit route.

SUBAREA ROUTING LIST

The SUBAREA_ROUTING_LIST provides the next transmission group number and adjacent subarea for a given destination subarea and explicit route number. The list is intialized during system definition, and may be optionally updated during network operation. It is used by PC.ERC and PC_ROUTE_MGR.ER_MGR in providing information on subarea routing of PIUs, as described in the explicit route control section in Chapter 3 and NC.ER_MGR in Chapter 12.

There is an entry in the list for each (destination) subarea (identified in the SUBAREA_ADDR field) in the network. The EXPLICIT_ROUTE structure provides sixteen entries, one for each ERN to the subarea specified in the SUBAREA_ADDR field. Each ERN entry includes a bit, ERN_SYSDEF, to indicate whether the ERN is defined to that destination subarea. In each entry, there is also an adjacent subarea (ADJ_SA) and transmission group number (TGN) field that identifies the particular path that the explicit route uses.

ERN MAP LIST

This structure provides mapping between VRN AND A ERN for a route between a destination subarea and this node.

VIRTUAL ROUTE IDENTIFIER LIST

Along with a session activation RU, the SSCP.SVC_MGR or LU.SVC_MGR passes to PU.SVC_MGR.CSC_MGR a set of parameters to be used by PU.SVC_MGR.PC_ROUTE_MGR.VR_MGR in assigning a virtual route to the session.

FAPL CONSTANTS

This structure contains all FAPL variables that are constants. The list is alphabetical. Multiple names are used to refer to the same value in order to improve the understandability of the FAPL procedures.

```
/*
+-------------------------------------------------------------------------+
|                                                                         |
|                        NODE CONTROL BLOCK (NCB)                         |
|                                                                         |
|     FUNCTION:  THIS DATA STRUCTURE CONTAINS THE PU ELEMENT ADDRESS AND NODE SUBAREA |
|                ADDRESS AS WELL AS CURRENT STATE VALUES FOR THE RESOURCE FSM'S |
|                WITHIN THE NODE.  THIS STRUCTURE PROVIDES STORAGE FOR POINTERS TO |
|                ALL MAJOR LISTS IN THE NODE, AS WELL AS THE DEFAULT ENTITY POINTERS |
|                FOR SOME OF THOSE LISTS.                                 |
|                                                                         |
+-------------------------------------------------------------------------+
*/

DCL 1 NCB BASED(NCB_PTR),                    /* NODE CONTROL BLOCK          */

      2 LIST_POINTERS,
        3 VRCB_LIST                PTR,      /* POINTER TO LIST HEADER FOR VRCB'S  */
        3 ERCB_LIST                PTR,      /* POINTER TO LIST HEADER FOR ERCB'S  */
        3 TGCB_LIST                PTR,      /* POINTER TO LIST HEADER FOR TGCB'S  */
        3 LSCB_LIST                PTR,      /* POINTER TO LIST HEADER FOR LSCB'S  */
        3 SCB_LIST                 PTR,      /* POINTER TO LIST HEADER FOR SCB'S   */
        3 NRCB_LIST                PTR,      /* POINTER TO LIST HEADER FOR NRCB'S  */
        3 DRCB_LIST                PTR,      /* POINTER TO LIST HEADER FOR DRCB'S  */
        3 ERN_MAP_LIST             PTR,      /* POINTER TO LIST HEADER FOR        */
                                             /*   ERN MAP LIST                    */
        3 CPCB_LIST                PTR,      /* POINTER TO LIST HEADER FOR CP'S   */
        3 SUBAREA_ROUTING_LIST     PTR,      /* POINTER TO LIST HEADER FOR        */
                                             /*   SUBAREA ROUTING                 */

      2 ENTITY_POINTERS,
        3 VRCB_PTR                 PTR,      /* DEFAULT POINTER FOR VRCB          */
        3 ERCB_PTR                 PTR,      /* DEFAULT POINTER FOR ERCB          */
        3 TGCB_PTR                 PTR,      /* DEFAULT POINTER FOR TGCB          */
        3 LSCB_PTR                 PTR,      /* DEFAULT POINTER FOR LSCB          */
        3 SCB_PTR                  PTR,      /* DEFAULT POINTER FOR SCB           */
        3 PATHCB_PTR               PTR,      /* DEFAULT POINTER FOR PATHCB        */
        3 SUBAREA_ROUTING_PTR      PTR,      /* DEFAULT POINTER FOR SUBAREA       */
        3 NRCB_PTR                 PTR,      /* DEFAULT POINTER FOR NRCB          */
        3 DRCB_PTR                 PTR,      /* DEFAULT POINTER FOR DRCB          */
        3 CPCB_PTR                 PTR,      /* DEFAULT POINTER FOR CPCB          */
        3 TCCB_PTR                 PTR,      /* DEFAULT POINTER FOR TCCB          */
        3 CP_INDIRECT_PTR          PTR,      /* DEFAULT POINTER FOR CP INDIRECT   */
        3 VR_RESERVATION_PTR       PTR,      /* DEFAULT POINTER FOR VR_RESERVATION */
        3 ERN_MAP_PTR              PTR,      /* DEFAULT POINTER FOR ERN MAP       */
        3 ASSOC_LSCB_ENTITY_PTR    PTR,      /* DEFAULT POINTER FOR ACT_LSCB_ENTITY */
        3 VR_ID_LIST_PTR           PTR,      /* DEFAULT POINTER FOR VR_ID_LIST    */
        3 PARM_DEFINE_ER_TO_TG_PTR PTR,      /* DEFAULT POINTER FOR               */
                                             /*          PARM_DEFINE_ER_TO_TG     */

      2 PU_NA,                               /* NETWORK ADDRESS OF THE PU IN THIS NODE */

        3 NODE_SUBAREA_ADDRESS     BIT(32),  /* SUBAREA ADDRESS OF THIS NODE STORED */
                                             /* RIGHT-JUSTIFIED PADDED WITH 0'S.  */

        3 PU_EA                    BIT(16),  /* ELEMENT ADDRESS OF THE PU IN THIS NODE */
                                             /* STORED RIGHT-JUSTIFIED PADDED WITH */
                                             /* 0'S.  THIS FIELD IS 0 FOR PU_T4|5. */

      2 SSCP_ELEMENT_ADDRESS       BIT(16),  /* ELEMENT ADDRESS OF THE SSCP STORED */
                                             /* RIGHT-JUSTIFIED PADDED WITH 0'S.  */
                                             /* THIS FIELD IS VALID FOR PU_T5'S ONLY. */

      2 SSCP_ID                    BIT(48),  /* ID USED TO RESOLVE ACTCDRM CONTENTION */

      2 SUBAREA_LEN                BIT(8),   /* NUMBER OF SIGNIFICANT BITS IN THE */
                                             /* SUBAREA ADDRESS.  IT IS EQUAL TO  */
                                             /* THE NUMBER OF B'1'S IN THE        */
                                             /* NODE_ELEMENT_MASK FIELD.          */

      2 NODE_SUBAREA_MASK          BIT(16),  /* BIT MASK CONTAINING B'1' IN EACH  */
                                             /* POSITION CORRESPONDING TO A BIT OF THE */
                                             /* SUBAREA ADDRESS FIELD AND B'0' IN EACH */
                                             /* POSITION CORRESPONDING TO A BIT OF THE */
                                             /* ELEMENT ADDRESS FIELD.  USED TO CONVERT */
                                             /* A FID1 NETWORK ADDRESS TO FID4 ADDRESS */

      2 NODE_ELEMENT_MASK          BIT(16),  /* BIT MASK CONTAINING B'1' IN EACH  */
                                             /* POSITION CORRESPONDING TO A BIT OF THE */
                                             /* ELEMENT ADDRESS FIELD AND B'0' IN EACH */
                                             /* POSITION CORRESPONDING TO A BIT OF THE */
                                             /* SUBAREA ADDRESS FIELD.            */

      2 PUCP_EA                    BIT(16),  /* ELEMENT ADDRESS OF THE PUCP FOR THIS */
                                             /* NODE                              */

      2 PU_TYPE                    BIT(4),   /* PU_TYPE OF THIS PU:               */
                                             /*    PU_T1, PU_T2, PU_T4, OR PU_T5  */
```

```
    2 INTERMEDIATE_FUNCTION        BIT(1),     /* FOR PU_T4|5:                      */
                                               /*   DOES THIS NODE PROVIDE INTERMEDIATE */
                                               /*   NODE ROUTING                    */

    2 MAX_ER_NUM                   BIT(8),     /* LARGEST ER NUMBER SUPPORTED BY    */
                                               /*   THIS NODE (0 ORIGIN)            */

    2 MAX_VR_NUM                   BIT(8),     /* LARGEST VR NUMBER SUPPORTED BY    */
                                               /*   THIS NODE (0 ORIGIN)            */

    2 ERN_DEFINITION_CAPABILITY BIT(1);        /* FOR PU_T4|5:  SPECIFIES           */
                                               /*   WHETHER THIS NODE CAN DEFINE THE */
                                               /*   MAPPING OF ERN TO TG_ID DYNAMICALLY */
                                               /*   OR WHETHER THE MAPPING MUST BE  */
                                               /*   ESTABLISHED AT SYSTEM DEFINITION. */
                                               /* B'0' STATIC_ONLY                  */
                                               /* B'1' ¬STATIC_ONLY                 */
```

```
/*
┌──────────────────────────────────────────────────────────────────────────┐
│                                                                          │
│                    PATH CONTROL CONTROL BLOCK (PCCB)                     │
│                                                                          │
│  FUNCTION:  THIS DATA STRUCTURE IS MAINTAINED FOR EACH PERIPHERAL NODE AND FOR │
│             EACH SUBAREA NODE THAT PROVIDES BOUNDARY FUNCTION SUPPORT.  IT │
│             PROVIDES WORKING STORAGE AND ATTRIBUTES REQUIRED BY PATH CONTROL │
│             COMPONENTS TO SEND AND RECEIVE PIU'S OVER ROUTE EXTENSIONS.   │
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
*/
```

```
DCL 1 PCCB BASED(PCCB_PTR),
    2 STATES(1:10)                 FIXED BIN,  /* FSM STATE INFORMATION          */
                                               /* FSM_STATION_BIU_ASSEMBLY       */

    2 Q_BTU_RCV                    PTR,        /* POINTER TO BTU RECEIVE QUEUE   */

    2 BIU_ASSEMBLY_OPTION          BIT(4),     /* PU_T1 OR PU_T2 BIU ASSEMBLY OPTION */
                                               /* X'0' NO_ASSEMBLY               */
                                               /* X'1' STATION_ASSEMBLY          */
                                               /* X'2' SESSION_ASSEMBLY          */

    2 PARTIAL_BIU_PTR              PTR,        /* POINTER TO PARTIALLY           */
                                               /*   ASSEMBLED BIU                */

    2 PIU_SEND_LIST                PTR;        /* POINTER TO PIU SEND LIST       */
```

```
/*
+-------------------------------------------------------------------------+
|                                                                         |
|                    NODE RESOURCE CONTROL BLOCK (NRCB) LIST              |
|                                                                         |
|   FUNCTION:  THIS DATA  STRUCTURE CONTAINS INFORMATION ABOUT  RESOURCES |
|              SUPPORTED BY THIS  NODE.  THE  INFORMATION IS CREATED  BY  |
|              A  SYSTEM DEFINITION PROCEDURE AND ENTRIES MAY BE DYNAMICAL|
|              LY ADDED, MODIFIED, OR DELETED BY A CONTROL  POINT (I.E.,  |
|              SSCP OR PUCP).  THE  STRUCTURE IS MANAGED BY THE PU.SVC_MGR|
|              .NS, BUT IS ACCESSED BY OTHER COMPONENTS WITHIN THE NODE.  |
|              SEE THE  PU.SVC_MGR.NS (CHAPTER 11) FOR A  DESCRIPTION OF  |
|              THE HIERARCHICAL STRUCTURE OF THESE RESOURCE ELEMENTS.     |
|                                                                         |
|   NOTE:      FOR EACH  RESOURCE CATEGORY ONLY A  SUBSET OF THE FIELDS  |
|              APPLY; THE FIELDS THAT DO  NOT APPLY ARE SET  TO 0 AND ARE |
|              NEVER REFERENCED IN THE PROCEDURES FOR THAT RESOURCE       |
|              CATEGORY.                                                  |
|                                                                         |
+-------------------------------------------------------------------------+
*/

ENTITY(NRCB),

  2 STATES(1:20)              FIXED BIN(8),/* FSM STATE INFORMATION          */
                                           /* FSM_ADJ_PU_LOAD               */
                                           /* FSM_ALS_CONNECTED_RES         */
                                           /* FSM_ALS_CONTACT_DISCONTACT_RES*/
                                           /* FSM_ALS_SEC_DUMP_RES          */
                                           /* FSM_ALS_SEC_IPL_RES           */
                                           /* FSM_ALS_SEC_RPO_RES           */
                                           /* FSM_ALS_TEST_RES              */
                                           /* FSM_ALS_SEC_XID_RES           */
                                           /* FSM_LINK_ACT_RES              */
                                           /* FSM_LINK_CONNIN_RES           */
                                           /* FSM_LINK_CONNOUT_RES          */
                                           /* FSM_LINK_TRACE_RES            */
                                           /* FSM_PU_ACT_RES                */
                                           /* FSM_PU_T2_LOAD                */

  2 RESOURCE_CATEGORY         BIT(4),      /* X'0' PU                       */
                                           /* X'1' LU                       */
                                           /* X'3' LINK                     */
                                           /* X'4' ALS                      */
                                           /* X'5' BF.PU                    */
                                           /* X'6' BF.LU                    */
                                           /* X'7' SSCP                     */

  2 ELEMENT_ADDRESS           BIT(16),     /* ELEMENT ADDRESS OF RESOURCE   */

  2 ASSOCIATED_RESOURCE       BIT(16),     /* ALS:   LINK ELEMENT ADDRESS   */
                                           /* BF.PU: ALS ELEMENT ADDRESS    */
                                           /* BF.LU: BF.PU ELEMENT ADDRESS  */
                                           /* LU:  PU ELEMENT ADDRESS IF NOT A*/
                                           /*      PRIMARY LU USED FOR PARALLEL*/
                                           /*      SESSIONS.                */
                                           /*  OR: LU ELEMENT ADDRESS IF A PRIMARY*/
                                           /*      LU USED FOR PARALLEL SESSIONS.*/

  2 LCP_RESET_OPTION          BIT(1),      /* FOR PU, LINK, OR ALS  RESIDING IN*/
                                           /*   A PU_T1|2 NODE: B'0' RESET  */
                                           /*                   B'1' CONTINUE*/

  2 SWITCHED_LINK             BIT(1),      /* FOR LINK OR ALS: B'0' NONSWITCHED*/
                                           /*                  B'1' SWITCHED*/

  2 ALS_DLC_HDR_ADDR          BIT(16),     /* THE DLC HEADER ADDRESS FOR ALS*/

  2 PRI_SEC_ROLE              BIT(1),      /* FOR LINK: B'0' ¬CONFIGURABLE  */
                                           /*           B'1'  CONFIGURABLE  */

  2 LINK_DLC_ROLE             BIT(4),      /* FOR LINK OR ALS: X'1' PRIMARY */
                                           /*                  X'2' SECONDARY*/

  2 CP_INDIRECT_LIST          POINTER,     /* FOR PU, LINK, OR ALS:  THE    */
                                           /* POINTER TO A LIST OF POINTERS TO THE*/
                                           /* CP'S THAT CONCURRENTLY SHARE THIS*/
                                           /* RESOURCE.  SEE CP INDIRECT LIST ON*/
                                           /* PAGE 6NDSB18..                */

  2 SHARE_LIMIT               FIXED BIN,   /* FOR PU, LINK, OR ALS:         */
                                           /* THE MAXIMUM COUNT OF SSCP'S THAT MAY*/
                                           /* SHARE THIS RESOURCE.  FOR ALS */
                                           /* THE LIMIT IS LESS THAN OR EQUAL TO*/
                                           /* THAT FOR THE ASSOCIATED LINK. */
                                           /* THE LIMIT FOR A LINK IS LESS THAN OR*/
                                           /* EQUAL TO THAT FOR THE PU.     */
```

```
2 BF_LOCAL_ID                  BIT(8),     /* LOCAL FORM OF ADDRESS FOR BF.(PU|LU)     */

2 ASSIGNING_CP_SCB_ID          PTR,        /* IDENTIFIER FOR SSCP-PU HALF-SESSION      */
                                           /* OF THE RNAA ISSUER FOR BF.PU OR BF.LU.   */
                                           /* A VALUE OF NULL DESIGNATES ASSIGNMENT    */
                                           /* BY SYSTEM DEFINITION                     */

2 SEC_RCV_PACING_CNT           BIT(6),     /* SECONDARY CPMGR'S RECEIVE PACING         */
                                           /* COUNT FOR BF.LU                          */

2 RESOURCE_TYPE                BIT(4),     /* PU TYPE FOR BF.PU: PU_T1 OR PU_T2        */

2 SAVE_RU_FOR_RETRY_LIST       PTR;        /* LIST OF RU'S WAITING ON A                */
                                           /* LINK OR ALS                              */
```

This page intentionally left blank

```
/*
┌──────────────────────────────────────────────────────────────────────┐
│                                                                      │
│                    CP INDIRECT (CP_INDIRECT) LIST                    │
│                                                                      │
│  FUNCTION:  THIS DATA STRUCTURE CONTAINS A LIST OF POINTERS TO ENTRIES IN THE │
│             CPCB LIST.  THE CPCB'S IDENTIFIY CONTROL POINTS THAT CURRENTLY SHARE │
│             THE NODE RESOURCE FOR WHICH THE LIST IS MAINTAINED.  CONTROL POINTS │
│             ARE PLACED IN THE LIST WHEN A CONTROL POINT SUCCESSFULLY INITIATES │
│             SHARED CONTROL OF A RESOURCE (E.G., ACTLINK FOR A LINK, CONTACT FOR │
│             AN ADJACENT LINK STATION).  ENTRIES ARE REMOVED WHEN THE CONTROL │
│             POINT TERMINATES CONTROL, THE RESOURCE BECOMES INOPERATIVE, OR THE │
│             SESSION BETWEEN THE CONTROL POINT AND THE NODE'S PU IS DEACTIVATED. │
│             THE LIST IS MANAGED AND USED BY THE PU.SVC_MGR.NS.       │
└──────────────────────────────────────────────────────────────────────┘
*/

ENTITY(CP_INDIRECT),

  2 CP_ENTRY_PTR                PTR;        /* POINTER TO CPCB LIST ENTRY FOR      */
                                            /* THE CONTROL POINT THAT              */
                                            /* HAS ACQUIRED THE RESOURCE           */

/*
┌──────────────────────────────────────────────────────────────────────┐
│                                                                      │
│                CONTROL POINT CONTROL BLOCK (CPCB) LIST               │
│                                                                      │
│  FUNCTION:  THIS DATA STRUCTURE CONTAINS THE LIST OF CONTROL POINTS THAT │
│             CURRENTLY SHARE THE PU.  CONTROL POINT ADDRESSES ARE PLACED IN THE │
│             LIST WHEN A CONTROL POINT SUCCESSFULLY INITIATES SHARED CONTROL OF A │
│             PU WITH ACTPU.  ENTRIES ARE REMOVED WHEN THE CONTROL POINT │
│             TERMINATES CONTROL WITH DACTPU, OR THE SESSION BETWEEN THE CONTROL │
│             POINT AND THE NODE'S PU BECOMES INOPERATIVE.  THE LIST IS MANAGED │
│             AND USED BY THE PU.SVC_MGR.NS.                           │
└──────────────────────────────────────────────────────────────────────┘
*/

ENTITY(CPCB),

  2 STATES(1:10)                FIXED BIN(8),/* FSM STATE INFORMATION          */
                                            /* FSM_CP_SESS_SDT                 */

  2 CP_SCB_ID                   PTR,        /*HALF-SESSION IDENTIFIER FOR SSCP-PU */
                                            /* SESSION                         */

  2 ER_VR_SUPP                  BIT(1),     /* B'0'  PRE_ER_VR; ADJACENT NODE DOES NOT */
                                            /*     SUPPORT ER AND VR PROTOCOLS */
                                            /* B'1' ¬PRE_ER_VR; ADJACENT NODE  */
                                            /*     SUPPORTS ER AND VR PROTOCOLS */

  2 NS_LSA_RQD                  BIT(1);     /* B'0' ¬NS_LSA_REQUIRED           */
                                            /* B'1'  NS_LSA_REQUIRED           */
```

```
/*
+-------------------------------------------------------------------------+
|                                                                         |
|                      SESSION CONTROL BLOCK LIST (SCB)                   |
|                                                                         |
|   FUNCTION:  THIS DATA STRUCTURE IS CREATED FOR EACH HALF-SESSION AND/OR BOUNDARY |
|              FUNCTION HALF-SESSION.  IT  MAINTAINS THE STATUS OF  THAT PARTICULAR |
|              HALF-SESSION.                                              |
|                                                                         |
+-------------------------------------------------------------------------+
*/

ENTITY(SCB),

  2 STATES(1:60)              FIXED BIN(8),/* FSM STATE INFORMATION          */
                                           /* FSM_BSM_BIDDER                */
                                           /* FSM_BSM_FSP                   */
                                           /* FSM_CHAIN_RCV                 */
                                           /* FSM_CHAIN_SEND                */
                                           /* FSM_CNTL_IMMED_EXP            */
                                           /* FSM_CONTROL_BSM_RSP_RCV       */
                                           /* FSM_CONTROL_BSM_RSP_SEND      */
                                           /* FSM_CONTROL_HDX_RSP_RCV       */
                                           /* FSM_CONTROL_HDX_RSP_RCV_ERP_DL*/
                                           /* FSM_CONTROL_HDX_RSP_RCV_ERP_IM*/
                                           /* FSM_CONTROL_HDX_RSP_SEND      */
                                           /* FSM_CONTROL_HDX_RSP_SEND_ERP_DL*/
                                           /* FSM_CONTROL_HDX_RSP_SEND_ERP_IM*/
                                           /* FSM_CRV_RCV                   */
                                           /* FSM_CRV_SEND                  */
                                           /* FSM_DT_RCV_CLEAR              */
                                           /* FSM_DT_RCV_SDT                */
                                           /* FSM_DT_RCV_SDT_AND_CLEAR      */
                                           /* FSM_DT_SEND_CLEAR             */
                                           /* FSM_DT_SEND_SDT               */
                                           /* FSM_DT_SEND_SDT_AND_CLEAR     */
                                           /* FSM_EBCD_RCV                  */
                                           /* FSM_EBCD_SEND                 */
                                           /* FSM_HDX_CONT_LOSER            */
                                           /* FSM_HDX_CONT_WINNER           */
                                           /* FSM_HDX_FF                    */
                                           /* FSM_IMM_RQ_MODE_RCV           */
                                           /* FSM_IMM_RQ_MODE_SEND          */
                                           /* FSM_QEC_RCV                   */
                                           /* FSM_QEC_SEND                  */
                                           /* FSM_QRI_CHECK_SEND            */
                                           /* FSM_QRI_CHAIN_RCV             */
                                           /* FSM_QRI_CHAIN_SEND            */
                                           /* FSM_RES                       */
                                           /* FSM_RQR_RCV                   */
                                           /* FSM_RQR_SEND                  */
                                           /* FSM_RTR_BIDDER                */
                                           /* FSM_RTR_FSP                   */
                                           /* FSM_SBI_RCV                   */
                                           /* FSM_SBI_SEND                  */
                                           /* FSM_SESS_BF_SSCP_LU           */
                                           /* FSM_SESS_BF_SSCP_PU_T1        */
                                           /* FSM_SESS_BF_SSCP_PU_T2        */
                                           /* FSM_SESS_BF_LU_LU             */
                                           /* FSM_SESS_LU_LU_PRI            */
                                           /* FSM_SESS_LU_LU_SEC            */
                                           /* FSM_SESS_SSCP_LU_PRI          */
                                           /* FSM_SESS_SSCP_LU_SEC          */
                                           /* FSM_SESS_SSCP_PU_PRI          */
                                           /* FSM_SESS_SSCP_PU_SEC          */
                                           /* FSM_SESS_SSCP_SSCP_PRI_OR_SEC */
                                           /* FSM_SESSION_BIU_ASSEMBLY      */
                                           /* FSM_SHUTD_RCV                 */
                                           /* FSM_SHUTD_SEND                */
                                           /* FSM_STSN_RCV                  */
                                           /* FSM_STSN_SEND                 */

  2 SCB_TYPE                  BIT(1),      /* SESSION TYPE:   B'0' HALF_SESS */
                                           /*                 B'1' BF_SESS   */

  2 #SVC_MGR                  GENERIC VALUES(LU.SVC_MGR.SS.RCV,
                                             SSCP.SVC_MGR.CS.RCV,
                                             PU.SVC_MGR.NS.RCV,
                                             BF.LU.SVC_MGR,
                                             BF.PU.SVC_MGR,
                                             PUCP),
```

```
2 THIS_NA,                              /* NAU ASSOCIATED WITH THIS END OF      */
                                        /*   THE SESSION                        */
  3 THIS_SA                BIT(32),     /* SUBAREA ADDRESS                      */
  3 THIS_EA                BIT(16),     /* ELEMENT ADDRESS                      */

2 PARTNER_NA,                           /* NAU ASSOCIATED WITH THE OTHER END    */
                                        /*   OF THE SESSION                     */
  3 PARTNER_SA             BIT(32),     /* SUBAREA ADDRESS                      */
  3 PARTNER_EA             BIT(16),     /* ELEMENT ADDRESS                      */

2 VRCBPTR                  PTR,         /* POINTER TO VRCB FOR THIS SESSION     */

2 THIS_ID                  BIT(8),      /* LOCAL ID OF NAU ASSOCIATED WITH      */
                                        /*   THIS END OF THE SESSION            */
2 PARTNER_ID               BIT(8),      /* LOCAL ID OF NAU ASSOCIATED WITH      */
                                        /*   OTHER END OF SESSION.              */

2 LOCAL_SESSION_ID         BIT(8),      /* LSID FOR FID3 SESSIONS               */
2 BF_ALS_EA                BIT(16),     /* ALS ELEMENT ADDRESS FOR PERIPHERAL NODE */
2 SUPPORTED_NODE_TYPE      BIT(8),      /* PERIPHERAL NODE TYPE                 */

2 PARTIAL_BIU_PTR          PTR,         /* POINTER TO PARTIAL BIU BEING         */
                                        /* ASSEMBLED ON A SESSION BASIS         */

2 Q_TC_TO_DFC              PTR,         /*                                      */

2 CORRELATION_TABLES,
  3 CT_RCV_RQ_EXP          PTR,         /*                                      */
  3 CT_RCV_RQ_NORM         PTR,         /*                                      */
  3 CT_SEND_RQ_EXP         PTR,         /*                                      */
  3 CT_SEND_RQ_NORM        PTR,         /*                                      */

2 ENTITY_POINTERS,                      /*                                      */
  3 CT_RCV_RQ_EXP_ENTRY_PTR PTR,        /*                                      */
  3 CT_SEND_RQ_EXP_ENTRY_PTR PTR,       /*                                      */
  3 CT_NORM_ENTRY_PTR      PTR,         /*                                      */
  3 PAC_RSP_SIGNAL_PTR     PTR,         /*                                      */
2 DFC_VARIABLES,                        /*                                      */
  3 CT_PTR                 PTR,         /*                                      */
  3 SCAN_PTR               PTR,         /*                                      */
  3 MU_PTR_SAVE            PTR,         /*                                      */
  3 SQN_SEND_CNT           FIXED BIN,   /*                                      */
  3 KEY                    FIXED BIN,   /*                                      */
  3 SNC_BSM_RCVD           BIT(32),     /*                                      */
  3 SNC_BSM_SENT           BIT(32),     /*                                      */
  3 SNC_HDX_RCVD           BIT(32),     /*                                      */
  3 SNC_HDX_SENT           BIT(32),     /*                                      */
  3 CT_ENTRY               BIT(1),      /*                                      */

2 TC_VARIABLES,                         /*                                      */
  3 #FSM_RQR               FIXED BIN(8), /*                                     */
  3 #FSM_STSN              FIXED BIN(8), /*                                     */
  3 #FSM_DT                FIXED BIN(8), /*                                     */
  3 #FSM_CRV               FIXED BIN(8), /*                                     */
  3 TC_CB_PTR              PTR,         /* POINTER TO TCCB                      */
  3 SEC_TO_BF_TC_CB_PTR    PTR,         /* ONLY IN BF                           */
  3 LAST_CLEAR_SNF         FIXED BIN(16), /* SNF VALUE OF LAST CLEAR SENT       */
  3 SEND_NORM_SNF          FIXED BIN(16), /* LAST NORMAL-FLOW SNF VALUE         */
                                        /* SENT BY PU_T1                        */
  3 SEND_EXP_SNF           FIXED BIN(16), /* LAST EXPEDITED-FLOW SNF VALUE      */
                                        /* SENT BY PU_T1                        */
  3 SQN_RCV_CNT            FIXED BIN(16), /* NORMAL-FLOW SNF VALUE EXPECTED     */
  3 RCV_EXP_SNF            FIXED BIN(16), /* LAST EXPEDITED-FLOW SNF            */
                                        /* VALUE SENT TO PU_T1                  */

2 ACT_RQ_RSP_SEQ_ID        BIT(64),     /* ID RECEIVED IN ACTIVATION RQ         */
```

```
/*
┌──────────────────────────────────────────────────────────────────────┐
│ THIS EXTENSION OF THE SCB CONTAINS PARAMETERS THAT ARE OBTAINED FROM │
│ ACTIVATION RU AND SPECIFIED BY THE TS AND FM PROFILES. OTHER PARAMETERS ARE │
│ DERIVED FROM THE ACTIVATION OPTIONS AND ARE PLACED IN THE STRUCTURE BY SESSACT │
│ (SEE CHAPTERS 4, 5, AND 13).                                         │
│                                                                      │
│ THE FOLLOWING PARAMETERS ARE OBTAINED FROM THE ACTIVATION RU AND TS AND FM │
│ PROFILES BY PU.SVC_MGR.CSC_MGR (SEE CHAPTER 13). THE ORDER OF THE PARAMETERS │
│ CORRESPONDS TO THE BIND FORMAT (SEE APPENDIX E).                     │
└──────────────────────────────────────────────────────────────────────┘
*/
/*
        ┌──────────────────────────────────┐
        │ CORRESPONDS TO BYTE 2 OF BIND    │
        └──────────────────────────────────┘
                                                                        */
2 FM_PROFILE                BIT(8),     /* X'00' FM PROFILE 0          */
                                        /* X'02' FM PROFILE 2          */
                                        /* X'03' FM PROFILE 3          */
                                        /* X'04' FM PROFILE 4          */
                                        /* X'05' FM PROFILE 5          */
                                        /* X'06' FM PROFILE 6          */
                                        /* X'07' FM PROFILE 7          */
                                        /* X'10' FM PROFILE 16         */
                                        /* X'11' FM PROFILE 17         */
                                        /* X'12' FM PROFILE 18         */
                                                                        /*
        ┌──────────────────────────────────┐
        │ CORRESPONDS TO BYTE 3 OF BIND    │
        └──────────────────────────────────┘
                                                                        */
2 TS_PROFILE                BIT(8),     /* X'01' TS PROFILE 1          */
                                        /* X'02' TS PROFILE 2          */
                                        /* X'03' TS PROFILE 3          */
                                        /* X'04' TS PROFILE 4          */
                                        /* X'05' TS PROFILE 5          */
                                        /* X'07' TS PROFILE 7          */
                                        /* X'10' TS PROFILE 16         */
                                        /* X'11' TS PROFILE 17         */
                                                                        /*
        ┌──────────────────────────────────┐
        │ CORRESPONDS TO BYTE 4 OF BIND PRIMARY LU │
        │ PROTOCOLS                        │
        └──────────────────────────────────┘
                                                                        */
2 PRI_CHAIN_USE             BIT(1),     /* B'0' SINGLE                 */
                                        /* B'1' MULTIPLE               */

2 PRI_RQ_MODE               BIT(1),     /* B'0' IMMEDIATE              */
                                        /* B'1' DELAYED                */

2 PRI_TWO_PHASE_COMMIT      BIT(1),     /* B'0' ¬SUPPORTED             */
                                        /* B'1'  SUPPORTED             */
                                                                        /*
        ┌──────────────────────────────────┐
        │ THE FIELD CORRESPONDING TO THE PRIMARY CHAIN │
        │ RESPONSE FIELD IN BIND RU IS MAPPED INTO THE │
        │ CHAIN_RSP FIELD                  │
        └──────────────────────────────────┘
                                                                        */
2 PRI_COMPR_IND             BIT(1),     /* B'0' NO_COMPRESSION         */
                                        /* B'1' COMPRESSION            */

2 PRI_EB_IND                BIT(1),     /* B'0' MAY_NOT_SEND           */
                                        /* B'1' MAY_SEND               */
                                                                        /*
        ┌──────────────────────────────────┐
        │ CORRESPONDS TO BYTE 5 OF BIND SECONDARY LU │
        │ PROTOCOLS                        │
        └──────────────────────────────────┘
                                                                        */
2 SEC_CHAIN_USE             BIT(1),     /* B'0' SINGLE                 */
                                        /* B'1' MULTIPLE               */

2 SEC_RQ_MODE               BIT(1),     /* B'0' IMMEDIATE              */
                                        /* B'1' DELAYED                */
                                                                        /*
        ┌──────────────────────────────────┐
        │ THE FIELD CORRESPONDING TO THE SECONDARY │
        │ CHAIN RESPONSE FIELD IN THE BIND RU IS MAPPED │
        │ INTO THE CHAIN_RSP FIELD         │
        └──────────────────────────────────┘
                                                                        */
2 SEC_COMPR_IND             BIT(1),     /* B'0' NO_COMPRESSION         */
                                        /* B'1' COMPRESSION            */

2 SEC_EB_IND                BIT(1),     /* B'0' MAY_NOT_SEND           */
                                        /* B'1' MAY_SEND               */

2 SEC_TWO_PHASE_COMMIT      BIT(1),     /* B'0' ¬SUPPORTED             */
                                        /* B'1'  SUPPORTED             */
```

```
                                                                        /*
        ┌──────────────────────────────────────────────────────┐
        │ CORRESPONDS TO BYTE 6 OF BIND COMMON LU              │
        │ PROTOCOLS                                            │
        └──────────────────────────────────────────────────────┘
                                                                        */
2 FM_HDR_USAGE                  BIT(1),     /* B'0' NO_FM_HEADERS      */
                                            /* B'1' FM_HEADERS         */

2 BRACKETS_RESET_STATE          BIT(1),     /* B'0' INB OR BRACKETS NOT USED */
                                            /* B'1' BETB               */

2 BRKT_TERM_RULE                BIT(1),     /* B'0' UNCONDITIONAL      */
                                            /* B'1' CONDITIONAL        */

2 ALT_CODE                      BIT(1),     /* B'0' NOT_USED           */
                                            /* B'1' MAY_BE_USED        */
                                                                        /*
        ┌──────────────────────────────────────────────────────┐
        │ CORRESPONDS TO BYTE 7 OF BIND                        │
        └──────────────────────────────────────────────────────┘
                                                                        */

2 SEND_RCV_MODE                 BIT(2),     /* B'00' FULL_DUPLEX       */
                                            /* B'01' HDX_CONTENTION    */
                                            /* B'10' HDX_FLIP_FLOP     */

2 RECOVERY_RESP                 BIT(1),     /* B'0' LOSER_RESPONSIBLE  */
                                            /* B'1' SYMMETRIC          */

2 CONT_WIN                      BIT(1),     /* B'0' SEC                */
                                            /* B'1' PRI                */

2 HDX_FF_RESET_STATE            BIT(1),     /* B'0' SEND_FOR_SEC       */
                                            /* B'1' SEND_FOR_PRI       */
                                                                        /*
        ┌──────────────────────────────────────────────────────┐
        │ CORRESPONDS TO BYTE 8 OF BIND                        │
        └──────────────────────────────────────────────────────┘
                                                                        */
2 PACING_PARAMETERS,
  3 SEC_STAGING_IND             BIT(1),     /* B'0' ONE-STAGE          */
                                            /* B'1' TWO-STAGE          */
                                                                        /*
        ┌──────────────────────────────────────────────────────┐
        │ THE MEANINGS OF B'0' AND B'1' ARE REVERSED           │
        │ FROM PRI_STAGING_IND.                                │
        └──────────────────────────────────────────────────────┘
                                                                        */
  3 SEC_RCV_PACING_CNT          BIT (6),
                                                                        /*
        ┌──────────────────────────────────────────────────────┐
        │ CORRESPONDS TO BYTE 9 OF BIND                        │
        └──────────────────────────────────────────────────────┘
                                                                        */
  3 SEC_SEND_PACING_CNT         BIT(6),
                                                                        /*
        ┌──────────────────────────────────────────────────────┐
        │ CORRESPONDS TO BYTE 10 OF BIND                       │
        └──────────────────────────────────────────────────────┘
                                                                        */
  3 SEC_SEND_MAX_RU_SIZE        BIT(8),
                                                                        /*
        ┌──────────────────────────────────────────────────────┐
        │ CORRESPONDS TO BYTE 11 OF BIND                       │
        └──────────────────────────────────────────────────────┘
                                                                        */
  3 PRI_SEND_MAX_RU_SIZE        BIT(8),
                                                                        /*
        ┌──────────────────────────────────────────────────────┐
        │ CORRESPONDS TO BYTE 12 OF BIND                       │
        └──────────────────────────────────────────────────────┘
                                                                        */
  3 PRI_STAGING_IND             BIT(1),     /* B'0' TWO-STAGE          */
                                            /* B'1' ONE-STAGE          */
                                                                        /*
        ┌──────────────────────────────────────────────────────┐
        │ THE MEANINGS OF B'0' AND B'1' ARE REVERSED           │
        │ FROM SEC_STAGING_IND.                                │
        └──────────────────────────────────────────────────────┘
                                                                        */
  3 PRI_SEND_PACING_CNT         BIT(6),
                                                                        /*
        ┌──────────────────────────────────────────────────────┐
        │ CORRESPONDS TO BYTE 13 OF BIND                       │
        └──────────────────────────────────────────────────────┘
                                                                        */
  3 PRI_RCV_PACING_CNT          BIT(6),
                                                                        /*
        ┌──────────────────────────────────────────────────────┐
        │ CORRESPONDS TO BYTE 14 OF BIND                       │
        └──────────────────────────────────────────────────────┘
                                                                        */
2 PS_PROFILE,
  3 PS_USAGE_FMT                BIT(1),     /* B'0' BASIC              */
                                            /* B'1' RESERVED           */
  3 LU_TYPE                     BIT(7),
```

```
                                                                        /*
            ┌──────────────────────────────────────────┐
            │      CORRESPONDS TO BYTES 15 TO 25 OF BIND      │
            └──────────────────────────────────────────┘
                                                                        */
2 PS_USAGE                    CHAR(11),                                 /*
            ┌──────────────────────────────────────────┐
            │         CORRESPONDS TO BYTE 26 OF BIND          │
            └──────────────────────────────────────────┘
                                                                        */
2 CRYPTOGRAPHY_SESSION_LEVEL BIT(2),        /* B'00' NONE              */
                                            /* B'01' SELECTIVE         */
                                            /* B'11' MANDATORY         */
                                                                        /*
            ┌──────────────────────────────────────────┐
            │         CORRESPONDS TO BYTE 27 OF BIND          │
            └──────────────────────────────────────────┘
                                                                        */
2 CRYPTOGRAPHY_KEY_ENCIPH_METHOD BIT(2),/* B'00' SLU_KEY               */

2 CRYPTOGRAPHY_CIPHER_METHOD   BIT(3),      /* B'000' BLOCK_CHAINING_WITH_SEED */
                                            /*  (AND CIPHER TEXT FEEDBACK)     */
                                                                        /*
            ┌──────────────────────────────────────────┐
            │       CORRESPONDS TO BYTES 27-34 OF BIND        │
            └──────────────────────────────────────────┘
                                                                        */
2 SESS_CRYPTOGRAPHY_KEY        CHAR(8),     /* CRYPTOGRAPHY KEY FIELD  */
                                            /* FROM BIND               */
                                                                        /*
            ┌──────────────────────────────────────────┐
            │  THIS ENDS  THE PORTION  OF THE  CONTROL BLOCK  │
            │  CORRESPONDING TO THE BIND PARAMETERS.          │
            └──────────────────────────────────────────┘
                                                                        */
                                                                        /*
            ┌──────────────────────────────────────────┐
            │  THE  FOLLOWING PARAMETERS  ARE OBTAINED  FROM  │
            │  THE TS AND FM  PROFILES BY PU.SVC_MGR.CSC_MGR  │
            │  (SEE CHAPTER 13).                              │
            └──────────────────────────────────────────┘
                                                                        */
2 SQN_USAGE                    BIT(2),      /* B'00' IDENTIFIERS       */
                                            /* B'01' SEQUENCE_NUMBERS  */
                                            /* B'10' NO_SNF            */

2 PRI_RSP_MODE                 BIT(1),      /* B'0' IMMEDIATE          */
                                            /* B'1' DELAYED            */

2 SEC_RSP_MODE                 BIT(1),      /* B'0' IMMEDIATE          */
                                            /* B'1' DELAYED            */

2 SC_CLEAR                     BIT(1),      /* B'0' ¬ALLOWED           */
                                            /* B'1' ALLOWED            */

2 SC_RQR                       BIT(1),      /* B'0' ¬ALLOWED           */
                                            /* B'1' ALLOWED            */

2 SC_SDT                       BIT(1),      /* B'0' ¬ALLOWED           */
                                            /* B'1' ALLOWED            */

2 SC_STSN                      BIT(1),      /* B'0' ¬ALLOWED           */
                                            /* B'1' ALLOWED            */

2 SC_CRV                       BIT(1),      /* B'0' ¬ALLOWED           */
                                            /* B'1' ALLOWED            */
```

```
                                                                        /*
        ┌──────────────────────────────────────────────────┐
        │ THE FOLLOWING PARAMETERS ARE DERIVED BY          │
        │ PU.SVC_MGR.CSC_MGR FROM ACTIVATION OPTIONS       │
        │ (SEE CHAPTER 13).                                │
        └──────────────────────────────────────────────────┘
                                                                        */
2 TYPE_OF_SESSION              BIT(3),     /* B'000' SSCP_PU           */
                                           /* B'001' SSCP_LU           */
                                           /* B'010' SSCP_SSCP         */
                                           /* B'011' LU_LU             */
                                           /* B'100' PUCP_PU           */

2 HALF_SESSION                 BIT(1),     /* B'1' PRI                 */
                                           /* B'0' SEC                 */

2 THIS_HALF_SESSION_SSCP_ID    BIT(48),
2 PARTNER_HALF_SESSION_SSCP_ID BIT(48),
2 #FSM_SESS                    FIXED BIN(8),
2 PU_FMD_RU_USAGE,
  3 REQUEST_RECEIVE            BIT(1),     /* B'0' ¬ALLOWED            */
                                           /* B'1' ALLOWED             */

                                                                        /*
        ┌──────────────────────────────────────────────────┐
        │ THESE VALUES ARE OBTAINED FROM THE BIND RU       │
        └──────────────────────────────────────────────────┘
                                                                        */
2 CHAIN_RSP,                               /* B'0' ¬ALLOWED | B'1' ALLOWED */

  3 CHAIN_RSP_FOR_PRIMARY,                 /* FOR CHAINS SENT BY PRIMARY */
    4 PRI_NO_RSP_CHAIN         BIT(1),
    4 PRI_EXCP_RSP_CHAIN       BIT(1),
    4 PRI_DEF_RSP_CHAIN        BIT(1),

  3 CHAIN_RSP_FOR_SECONDARY,               /* FOR CHAINS SENT BY SECONDARY */
    4 SEC_NO_RSP_CHAIN         BIT(1),
    4 SEC_EXCP_RSP_CHAIN       BIT(1),
    4 SEC_DEF_RSP_CHAIN        BIT(1),

                                                                        /*
        ┌──────────────────────────────────────────────────┐
        │ THE FOLLOWING PARAMETERS ARE DERIVED FROM THE    │
        │ ACTIVATION OPTIONS AND ARE INITIALIZED BY        │
        │ SESSACT.DFC_INITIALIZE (SEE CHAPTER 5).          │
        └──────────────────────────────────────────────────┘
                                                                        */
                                                                        /*
        ┌──────────────────────────────────────────────────┐
        │ DFC REQUESTS ALLOWED FOR THIS HALF-SESSION       │
        └──────────────────────────────────────────────────┘
                                                                        */
2 DFC_NORMAL_REQUESTS,                     /* B'0' ¬ALLOWED | B'1' ALLOWED */
  3 DFC_BID_RCV                BIT(1),
  3 DFC_BID_SEND               BIT(1),
  3 DFC_BIS_RCV                BIT(1),
  3 DFC_BIS_SEND               BIT(1),
  3 DFC_CANCEL_RCV             BIT(1),
  3 DFC_CANCEL_SEND            BIT(1),
  3 DFC_CHASE_RCV              BIT(1),
  3 DFC_CHASE_SEND             BIT(1),
  3 DFC_LUSTAT_RCV             BIT(1),
  3 DFC_LUSTAT_SEND            BIT(1),
  3 DFC_QC_RCV                 BIT(1),
  3 DFC_QC_SEND                BIT(1),
  3 DFC_RTR_RCV                BIT(1),
  3 DFC_RTR_SEND               BIT(1),

2 DFC_EXPEDITED_REQUESTS,                  /* B'0' ¬ALLOWED | B'1' ALLOWED */
  3 DFC_QEC_RCV                BIT(1),
  3 DFC_QEC_SEND               BIT(1),
  3 DFC_RELQ_RCV               BIT(1),
  3 DFC_RELQ_SEND              BIT(1),
  3 DFC_RSHUTD_RCV             BIT(1),
  3 DFC_RSHUTD_SEND            BIT(1),
  3 DFC_SBI_RCV                BIT(1),
  3 DFC_SBI_SEND               BIT(1),
  3 DFC_SHUTC_RCV              BIT(1),
  3 DFC_SHUTC_SEND             BIT(1),
  3 DFC_SHUTD_RCV              BIT(1),
  3 DFC_SHUTD_SEND             BIT(1),
  3 DFC_SIG_RCV                BIT(1),
  3 DFC_SIG_SEND               BIT(1),
```

```
2 DFC_FSM_USAGE,
  3 #FSM_BSM                    FIXED BIN(8),
  3 #FSM_CHAIN_RCV              FIXED BIN(8),
  3 #FSM_CHAIN_SEND             FIXED BIN(8),
  3 #FSM_CONTROL_BSM_RSP_RCV    FIXED BIN(8),
  3 #FSM_CONTROL_BSM_RSP_SEND   FIXED BIN(8),
  3 #FSM_CONTROL_HDX_RSP_RCV    FIXED BIN(8),
  3 #FSM_CONTROL_HDX_RSP_SEND   FIXED BIN(8),
  3 #FSM_EBCD_RCV               FIXED BIN(8),
  3 #FSM_EBCD_SEND              FIXED BIN(8),
  3 #FSM_HDX                    FIXED BIN(8),
  3 #FSM_IMM_RQ_MODE_RCV        FIXED BIN(8),
  3 #FSM_IMM_RQ_MODE_SEND FIXED BIN(8),
  3 #FSM_QEC_RCV                FIXED BIN(8),
  3 #FSM_QEC_SEND               FIXED BIN(8),
  3 #FSM_QRI_CHAIN_RCV          FIXED BIN(8),
  3 #FSM_QRI_CHAIN_SEND         FIXED BIN(8),
  3 #FSM_QRI_CHECK_SEND         FIXED BIN(8),
  3 #FSM_RES                    FIXED BIN(8),
  3 #FSM_RTR                    FIXED BIN(8),
  3 #FSM_SBI_RCV                FIXED BIN(8),
  3 #FSM_SBI_SEND               FIXED BIN(8),
  3 #FSM_SHUTD                  FIXED BIN(8),

2 DFC_MISC_SESSION_PARMS,
  3 USING_BRACKETS              BIT(1),     /* B'1' YES | B'0' NO                        */

  3 FIRST_SPEAKER               BIT(1),     /* B'1' YES | B'0' NO (NO IMPLIES BIDDER)    */

  3 THIS_HALF_SESSION_RQ_MODE   BIT(1),     /* B'0' IMMEDIATE | B'1' DELAYED             */

  3 PARTNER_HALF_SESSION_RQ_MODE BIT(1),    /* B'0' IMMEDIATE | B'1' DELAYED             */

  3 THIS_HALF_SESSION_RSP_MODE BIT(1),      /* B'0' IMMEDIATE | B'1' DELAYED             */

  3 PARTNER_HALF_SESSION_RSP_MODE BIT(1);   /* B'0' IMMEDIATE | B'1' DELAYED             */
```

```
/*
+-------------------------------------------------------------------------+
|                                                                         |
|                  TRANSMISSION CONTROL CONTROL BLOCK (TCCB)              |
|                                                                         |
|     FUNCTION:  THIS  CONTROL  BLOCK  CONTAINS  ALL  VARIABLES ASSOCIATED  WITH |
|                SESSION-LEVEL PACING  AND THE MAXIMUM  RU SIZE  THAT CAN BE  SENT OR |
|                RECEIVED.  THERE IS ONE TCCB FOR  EACH HALF-SESSION AND TWO FOR EACH |
|                SESSION SUPPORTED BY A BOUNDARY FUNCTION.  THE TWO CONTROL BLOCKS IN |
|                A BOUNDARY FUNCTION ARE USED BECAUSE THE PACING BETWEEN THE BOUNDARY |
|                FUNCTION AND THE PRIMARY IS INDEPENDENT OF THAT BETWEEN THE BOUNDARY |
|                FUNCTION AND THE SECONDARY.                              |
|                                                                         |
+-------------------------------------------------------------------------+
*/

ENTITY(TCCB),

   2 STATES(1:2)                FIXED BIN(8), /* FSM STATE INFORMATION          */
                                              /* FSM_PAC_RQ_SEND                */
                                              /* FSM_PAC_RQ_RCV                 */

   2 SEND_PACING                BIT(1),       /* B'1' YES                       */
                                              /* B'0' NO                        */

   2 RCV_PACING                 BIT(1),       /* B'1' YES                       */
                                              /* B'0' NO                        */

   2 MAX_RCV_RU_SIZE            BIT(32),      /* NOT_SPECIFIED = 0;             */
                                              /* OTHERWISE, A VALUE             */

   2 MAX_SEND_RU_SIZE           BIT(32),      /* NOT_SPECIFIED = 0;             */
                                              /* OTHERWISE, A VALUE             */

   2 PACING_COUNT               FIXED BIN(16), /* NUMBER OF RQ'S THAT CAN BE SENT */
                                              /* BEFORE RECEIVING PACING RSP    */

   2 WINDOW_SIZE                FIXED BIN(16), /* SIZE OF PACING GROUP           */

   2 #PC                        GENERIC VALUES(PC_T1.SEND,
                                               PC_T2.SEND,
                                               PC_SA.VRC.SEND,
                                               BF.PC.SEND),
                                              /* PC COMPONENT THAT MESSAGES      */
                                              /*  PROCESSED WITH THIS CB ARE SENT TO */

   2 Q_PAC                      PTR;          /* SESSION-LEVEL PACING QUEUE OF MU'S */
```

This page intentionally left blank

```
/*
+-------------------------------------------------------------------+
|                                                                   |
|                    DOMAIN RESOURCE CONTROL BLOCK LIST             |
|                                                                   |
|  FUNCTION:  THIS DATA  STRUCTURE CONTAINS INFORMATION ABOUT  RESOURCES SUPPORTED |
|             BY THIS  SSCP.  THE  INFORMATION IS CREATED  BY A  SYSTEM DEFINITION |
|             PROCEDURE AND ENTRIES MAY BE DYNAMICALLY ADDED, MODIFIED, OR DELETED |
|             BY THE SSCP.                                          |
|                                                                   |
|  NOTE:      FOR EACH  RESOURCE CATEGORY ONLY A  SUBSET OF THE FIELDS  APPLY; THE |
|             FIELDS THAT DO  NOT APPLY ARE SET  TO 0 AND ARE  NEVER REFERENCED IN |
|             THE PROCEDURES FOR THAT RESOURCE CATEGORY.            |
|                                                                   |
+-------------------------------------------------------------------+
*/

ENTITY(DRCB),

  2 STATES(1:20)              FIXED BIN(8),/* FSM STATE INFORMATION                  */
                                           /* FSM_ALS_CONNECTED_DOM_RES             */
                                           /* FSM_ALS_CONTACT_DOM_RES               */
                                           /* FSM_ALS_DUMP_DOM_RES                  */
                                           /* FSM_ALS_IPL_DOM_RES                   */
                                           /* FSM_ALS_RPO_DOM_RES                   */
                                           /* FSM_LINK_ACT_DOM_RES                  */
                                           /* FSM_LINK_CONNIN_DOM_RES               */
                                           /* FSM_LINK_CONNOUT_DOM_RES              */
                                           /* FSM_LU_ACT_DOM_RES                    */
                                           /* FSM_PROC_DOM_RES                      */
                                           /* FSM_PU_ACT_DOM_RES                    */

  2 RESOURCE_CATEGORY         BIT(4),      /* X'1' SUBAREA LU                       */
                                           /* X'2' SUBAREA_PU                       */
                                           /* X'3' LINK                             */
                                           /* X'4' ALS                              */
                                           /* X'5' PERIPHERAL_PU                    */
                                           /* X'6' PERIPHERAL_LU                    */

  2 NETWORK_NAME              CHAR(8),     /* NETWORK NAME OF RESOURCE              */

  2 NETWORK_ADDRESS           BIT(48),     /* NETWORK ADDRESS OF RESOURCE           */

  2 ASSOCIATED_RES_PTR        POINTER,     /* POINTER TO ASSOCIATED DOMAIN RESOURCE */
                                           /* ONE LEVEL HIGHER IN THE CONFIGURATION */
                                           /* HIERARCHY                             */
                                           /* FOR LINK:  ENTRY FOR A SUBAREA_PU     */
                                           /* FOR ALS:  ENTRY FOR A LINK            */
                                           /* FOR PERIPHERAL_PU: ENTRY FOR AN ALS   */
                                           /* FOR PERIPHERAL_LU: ENTRY FOR A        */
                                           /*                    PERIPHERAL_PU      */
                                           /* FOR LU: ENTRY FOR A SUBAREA NODE OR   */
                                           /*          A SECONDARY LU               */

  2 SESSION_ID                POINTER,     /* POINTER TO THE SESSION CONTROL BLOCK  */
                                           /* FOR THE SESSION WITH THIS PU OR LU    */

  2 NODE_SLOW                 BIT(1),      /* SUBAREA_PU(PU_T4)  IS IN SLOWDOWN:    */
                                           /*                      B'0' ¬IN_SLOWDOWN */
                                           /*                      B'1' IN_SLOWDOWN  */

  2 NODE_LINK_ADDR            BIT(16),     /* FOR SUBAREA_PU: CURRENT ELEMENT       */
                                           /* ADDRESS OF LINK TO LOCAL NODE         */

  2 SWITCHED_LINK             BIT(1),      /* FOR LINK OR ALS:      B'0' NONSWITCHED */
                                           /*                       B'1' SWITCHED    */

  2 LINK_DLC_ROLE             BIT(4),      /* FOR LINK OR ALS:    B'0001' PRIMARY   */
                                           /*                     B'0010' SECONDARY */

  2 BF_LOCAL_ID               BIT(8),      /* LOCAL FORM OF ADDRESS FOR PERIPHERAL  */
                                           /*    LU OR PU                           */

  2 PERIPHERAL_PU_TYPE        BIT(4),      /* PU TYPE FOR PERIPHERAL_PU             */

  2 SAVE_RU_FOR_RETRY_LIST    PTR,         /* LIST OF RU'S WAITING ON A RESOURCE    */

  2 DIAL_DIGITS               CHAR(20),    /* FOR PERIPHERAL PU: FOR SWITCHED LINKS */

  2 SEND_CONTACT_IMMEDIATELY  BIT(8),      /* FOR ALS: FROM REQDISCONT              */

  2 XID_IMAGE,                             /* FOR PERIPHERAL PU: XID THAT           */
                                           /* SHOULD BE IN REQCONT                  */
    3 FORMAT                  BIT(4),      /* XID FORMAT                            */
    3 PU_TYPE                 BIT(4),      /* PU TYPE                               */
    3 NODE_ID                 BIT(48),     /* NODE ID                               */
    3 FORMAT_SPECIFIC_DATA    CHAR(*);     /* SEE APPENDIX E                        */
```

```
/*
┌──────────────────────────────────────────────────────────────────────┐
│                                                                      │
│                    LINK STATION CONTROL BLOCK (LSCB) LIST            │
│                                                                      │
│  FUNCTION:  WHEN DLC CODE IS EXECUTING THERE IS ONE LSCB REPRESENTING THE LINK  │
│             AS A WHOLE AND ONE LSCB REPRESENTING EACH ADJACENT LINK STATION.    │
│             THE LSCB THAT REPRESENTS THE LINK IS IDENTIFIED BY THE LINK FLAG    │
│             CONTAINED IN THE LSCB. THE LINK LSCB PROVIDES A PLACE TO ANCHOR     │
│             FSM'S THAT PERTAIN TO THE ENTIRE LINK OR TO THE LOCAL STATION. THE  │
│             LSCB THAT REPRESENTS THE ADJACENT STATION CONTAINS PARAMETERS OF    │
│             BOTH THE ADJACENT AND LOCAL STATIONS, AS WELL AS PARAMETERS OF THE  │
│             SDLC COMMUNICATION BETWEEN STATIONS AND COPIES OF THE XID FIELDS    │
│             MOST RECENTLY SENT AND RECEIVED.                                    │
└──────────────────────────────────────────────────────────────────────┘
*/

ENTITY(LSCB),

  2 STATES(1:40)              FIXED BIN(8),/* FSM STATE INFORMATION          */
                                           /* FSM_TGN                       */
                                           /* FSM_XID_FORMAT_2              */

  2 LSCB_TYPE                 BIT(4),      /* X'3' LINK                     */
                                           /* X'4' ALS                      */

  2 EA                        BIT(16),     /* ELEMENT ADDRESS OF LINK OR ALS */

  2 DLC_TYPE                  BIT(8),      /* X'01' SDLC; X'02' CHAN370     */

  2 TGCBPTR                   PTR,         /* POINTER TO TRANSMISSION GROUP CONTROL */
                                           /* BLOCK                         */

  2 TGCB_RESET_IND            BIT(1),      /* B'0' TGCBPTR IS DYNAMICALLY ASSIGNED */
                                           /*      AFTER XID SWAP           */
                                           /* B'1' TGCBPTR IS STATICALLY ASSIGNED */
                                           /*      AND SHOULD NOT BE RESET  */

  2 SWITCHED_LINK             BIT(1),      /* B'1' SWITCHED                 */
                                           /* B'0' NONSWITCHED              */

  2 LOCAL_STATION,

    3 STATION_TYPE            BIT(1),      /* B'1' PRIMARY; B'0' SECONDARY  */

    3 STA_XMT_RCV_CAP         BIT(1),      /* B'0' TWO-WAY ALTERNATING      */
                                           /* B'1' TWO-WAY SIMULTANEOUS     */

    3 MAX_BTU_LENGTH          BIT(16),     /* LONGEST BTU THIS LINK STATION IN */
                                           /* THIS NODE CAN RECEIVE         */
                                           /*
                ┌──────────────────────────────────┐
                │  LSCB FOR A LINK TERMINATES HERE │
                └──────────────────────────────────┘
                                                 */

                                           /*
                ┌──────────────────────────────────┐
                │ PARAMETERS OF ADJACENT LINK STATION │
                └──────────────────────────────────┘
                                                 */
  2 ADJ_STATION,

    3 DLC_ADDR                BIT(8),      /* DLC ADDRESS IN BLU SENT TO THE ALS */

    3 LINK_LSCB_PTR           PTR,         /* POINTS TO CORRESPONDING LINK LSCB */

    3 STATION_TYPE            BIT(1),      /* B'1' PRIMARY; B'0' SECONDARY  */

    3 STA_XMT_RCV_CAP         BIT(1),      /* B'0' TWO-WAY ALTERNATING      */
                                           /* B'1' TWO-WAY SIMULTANEOUS     */

    3 MAX_BTU_LENGTH          BIT(16),     /* LONGEST BTU ADJACENT LINK STATION */
                                           /* CAN RECEIVE                   */

    3 BTU_SEND_LIST           PTR,         /* LIST OF BTU'S TO TRANSMIT     */
```

```
                                                                        /*
          ┌──────────────────────────────────────┐
          │  PARAMETERS OF SDLC ERROR RECOVERY   │
          └──────────────────────────────────────┘
                                                                        */
2 SDLC_ERP,
  3 NS                          BIT(7),     /* SEND SEQUENCE NUMBER FOR OUTBOUND      */
                                            /* I FRAMES                               */
  3 NR                          BIT(7),     /* RCV SEQUENCE NUMBER FOR OUTBOUND FRAMES */
  3 LAST_NR_RCVD                BIT(7),     /* RECEIVE SEQ NUMBER LAST RECEIVED       */
  3 NS_CHECKPT                  BIT(7),     /* SEND SEQUENCE NUMBER OF POLL           */
  3 REJECT_ERP                  BIT(1),     /* B'1' SUPPORTED; B'0' ¬SUPPORTED        */
  3 MAX_ERP_RETRYS              FIXED BIN(15),/* MAX ATTEMPTED RETRANSMISSIONS        */
  3 NSEQ_CMD_OUTSTANDING        CHAR(4),    /* OUTSTANDING CMD REQUIRING EXPLICIT     */
                                            /* RSP, I.E., SNRM, SIM, DISC, XID,       */
                                            /* TEST, OR CFGR; MAY BE 'NONE'           */
  3 TIMEOUT,                                /* INITIAL VALUES OF SDLC TIMERS          */
    4 IDLE_STATE_DET            FIXED BIN(15),/* PRIMARY ONLY IN NRM                  */
    4 NON_PROD_RCV              FIXED BIN(15),/* PRIMARY ONLY IN NRM                  */
    4 INACTIVITY                FIXED BIN(15),/* ONLY SECONDARY, ONLY SWITCHED LINKS  */
                                                                        /*
          ┌──────────────────────────────────────┐
          │  PARAMETERS OF CONTACT PROCEDURE     │
          └──────────────────────────────────────┘
                                                                        */
2 CONTACTED_STATUS              CHAR(1),    /* STATUS CODE TO BE SET IN CONTACTED      */
                                                                        /*
          ┌──────────────────────────────────────┐
          │  XID MOST RECENTLY SENT              │
          │                                      │
          │  XID FORMAT 2 IS ILLUSTRATED HERE.   SEE
          │  APPENDIX E FOR MORE DETAIL ON XID FORMAT 2
          │  AND DESCRIPTIONS OF FORMATS 0 AND 1.
          └──────────────────────────────────────┘
                                                                        */
2 XID_SEND,
  3 FORMAT                      BIT(4),
  3 PU_TYPE                     BIT(4),
  3 LENGTH                      BIT(8),
  3 NODE_ID,
    4 BLOCK_NUM                 BIT(12),
    4 ID_NUM                    BIT(20),
    4 RESERVED                  BIT(16),
  3 TG_STATUS                   BIT(1),
  3 MULTI_LINK                  BIT(1),
  3 SEG_ASSEM_CAP               BIT(2),
  3 RESERVED                    BIT(4),
  3 FID_0_SUPPORTED             BIT(1),
  3 FID_1_SUPPORTED             BIT(1),
  3 RESERVED                    BIT(2),
  3 FID_4_SUPPORTED             BIT(1),
  3 RESERVED                    BIT(11),
  3 MAX_PIU_LENGTH              BIT(16),
  3 TGN                         BIT(8),
  3 SA                          BIT(32),
  3 RESERVED                    BIT(1),
  3 ERROR_STATUS                BIT(4),
  3 RESERVED                    BIT(3),
  3 CONTACT_OR_LOAD_STAT        BIT(8),
  3 IPL_LOAD_MODULE_NAME        CHAR(8),
  3 RESERVED                    BIT(16),
  3 DLC_TYPE                    BIT(8),
  3 RESERVED                    BIT(2),
  3 STA_ROLE_SEC                BIT(1),
  3 STA_ROLE_PRI                BIT(1),
  3 RESERVED                    BIT(2),
  3 STA_XMIT_RCV_CAP            BIT(2),
  3 MAX_RECEIVABLE_I_FIELD      BIT(16),
  3 RESERVED                    BIT(4),
  3 CMD_RSP_PROFILE             BIT(4),
  3 RESERVED                    BIT(2),
  3 SDLC_INIT_MODE,
    4 SDLC_INIT_SEND            BIT(1),
    4 SDLC_INIT_RCV             BIT(1),
  3 RESERVED                    BIT(21),
  3 MAXIN                       BIT(7),
  3 RESERVED                    BIT(40),
                                                                        /*
          ┌──────────────────────────────────────┐
          │  XID MOST RECENTLY RECEIVED          │
          └──────────────────────────────────────┘
                                                                        */
2 XID_RCV LIKE XID_SEND;
```

```
/*
+-------------------------------------------------------------------------+
|                                                                         |
|                  TRANSMISSION GROUP CONTROL BLOCK (TGCB) LIST           |
|                                                                         |
|   FUNCTION:  THIS  DATA STRUCTURE  IS  MAINTAINED  FOR EACH  TRANSMISSION  GROUP.  |
|              TRANSMISSION GROUP  FUNCTIONAL ATTRIBUTES ARE ESTABLISHED  AT SYSTEM  |
|              DEFINITION TIME  OR DERIVED  DURING XID  (FORMAT 2)  PROCESSING (SEE  |
|              CHAPTER  12).   TRANSMISSION  GROUP  CONTROL  (SEE  CHAPTER  3)  AND  |
|              PU.SVC_MGR.PC_ROUTE_MGR (  SEE CHAPTER 12)  USE THE TGCB  TO CONTROL  |
|              THE FUNCTIONS ASSOCIATED WITH A TRANSMISSION GROUP.        |
|                                                                         |
+-------------------------------------------------------------------------+
*/

ENTITY(TGCB),

  2 STATES(1:10)                FIXED BIN(8),/* FSM STATE INFORMATION                  */
                                             /* FSM_SUSPEND_TG_SEND                    */
                                             /* FSM_TG_SWEEP                           */
                                             /* FSM_VR_WINDOW_SIZE                     */

  2 TG_ID,                                   /* TRANSMISSION GROUP IDENTIFICATION      */

    3 TGN                       BIT(8),      /* TRANSMISSION GROUP NUMBER              */

    3 ADJ_SA                    BIT(32),     /* ADJACENT SUBAREA ADDRESS               */

  2 TG_FUNCTIONAL_ATTRIBUTES,

    3 MULTI_LINK_SUPP           BIT(1),      /* B'0' ¬MULTI_LINK_TG; SINGLE-LINK TG    */
                                             /* B'1'  MULTI_LINK_TG; MULTIPLE-LINK TG  */

    3 ER_VR_SUPP                BIT(1),      /* B'0'  PRE_ER_VR; ADJACENT NODE DOES NOT */
                                             /*     SUPPORT ER AND VR PROTOCOLS        */
                                             /* B'1' ¬PRE_ER_VR; ADJACENT NODE         */
                                             /*     SUPPORTS ER AND VR PROTOCOLS       */

    3 BLOCKING_SUPP             BIT(1),      /* B'0' ¬BLOCKING;  NOT SUPPORTED         */
                                             /* B'1'  BLOCKING;  SUPPORTED             */

    3 DEBLOCKING_SUPP           BIT(1),      /* B'0' ¬DEBLOCKING;  NOT SUPPORTED       */
                                             /* B'1'  DEBLOCKING; SUPPORTED            */

    3 MAX_SEND_BTU_LENGTH       BIT(16),     /* BYTE COUNT INDICATING MAXIMUM BTU      */
                                             /* LENGTH PERMITTED TO BE TRANSMITTED     */
                                             /* ON THE TRANSMISSION GROUP              */

    3 ASSOC_LSCB_LIST           PTR,         /* POINTS TO LIST OF LINK STATION         */
                                             /* CONTROL BLOCKS ASSOCIATED WITH THIS TG. */
                                             /* SEE FOLLOWING PAGE FOR DETAILS.        */

  2 TGC_WORKING_DATA,

    3 PRTY_SEND_PIU_LIST        PTR,         /* POINTS TO TG PRIORITY SEND PIU LIST    */

    3 RETRANSMIT_BTU_LIST       PTR,         /* POINTS TO TG RETRANSMIT BTU LIST       */

    3 Q_BTU_RCV                 PTR,         /* POINTS TO TG RECEIVE BTU QUEUE         */

    3 REPIFO_PIU_LIST           PTR,         /* POINTS TO TG REPIFO PIU LIST           */

    3 OUTSTANDING_BTU_CNT       BIT(16),     /* COUNT OF SEND BTU'S PASSED TO DLC      */
                                             /* FOR LINK STATIONS ASSIGNED TO TG,      */
                                             /* THAT HAVE NOT YET BEEN SUCCESSFULLY    */
                                             /* TRANSMITTED                            */

    3 TG_SNF_SEND_CNTR          BIT(12),     /* TG SEQ NUMBER FIELD SEND COUNTER       */

    3 TG_SNF_RCV_CNTR           BIT(12),     /* TG SEQ NUMBER FIELD RCV COUNTER        */

    3 TG_SNF_WRAP_ACK_SEND_CNTR BIT(16),     /* TG SEQ NUMBER FIELD WRAP ACK           */
                                             /* SEND COUNTER                           */

    3 TG_SNF_WRAP_ACK_RCV_CNTR  BIT(16),     /* TG SEQ NUMBER FIELD WRAP ACK           */
                                             /* RECEIVE COUNTER                        */

    3 SEND_BTU_PIU_VECTOR_LIST  PTR,         /* POINTS TO SEND BTU PIU VECTOR LIST     */

    3 TG_TRACE                  BIT(1);      /* B'0' ¬TRACE; TG TRACE NOT ACTIVE       */
                                             /* B'1'  TRACE; TG TRACE ACTIVE           */
```

```
/*
+-------------------------------------------------------------------------+
|                                                                         |
|                    ASSOCIATED LSCB (ASSOC_LSCB_ENTITY) LIST             |
|                                                                         |
|   FUNCTION:  THIS DATA STRUCTURE CONTAINS A LIST OF POINTERS TO ALL     |
|              ADJACENT LINK STATION CONTROL BLOCKS (LSCB'S) THAT ARE     |
|              CURRENTLY ACTIVE IN THE TRANSMISSION GROUP.  ELEMENTS     |
|              ON THE LIST ARE CREATED WHEN AN ADJACENT LINK STATION     |
|              HAS BEEN CONTACTED AND ARE DESTROYED WHEN THE ADJACENT    |
|              LINK STATION BECOMES INOPERATIVE OR HAS BEEN DISCONTACTED.|
|              THE LIST OF ASSOCIATED LSCB ENTITIES IS MAINTAINED IN THE |
|              ASSOC_LSCB_LIST OF THE TGCB.                               |
|                                                                         |
|              THE LIST IS MANAGED BY THE PU.SVC_MGR.NS (SEE CHAPTER 11). |
|                                                                         |
+-------------------------------------------------------------------------+
*/

ENTITY(ASSOC_LSCB_ENTITY),

  2 LSCBPTR                    PTR;        /* POINTER TO THE LINK STATION CONTROL   */
                                           /* BLOCK FOR THIS ADJACENT LINK STATION  */

/*
+-------------------------------------------------------------------------+
|                                                                         |
|                         PIU VECTOR (PIU_VECTOR) LIST                    |
|                                                                         |
|   FUNCTION:  THIS DATA STRUCTURE IS CREATED BY PATH CONTROL COMPONENTS  |
|              TO INDICATE THE LOCATION AND LENGTH OF A PIU TO BE         |
|              TRANSMITTED BY DATA LINK CONTROL.  IT IS DISCARDED ALONG   |
|              WITH THE PIU WHEN THE PIU IS SUCCESSFULLY TRANSMITTED OR   |
|              THE TRANSMISSION OF THE PIU IS ABANDONED.                  |
|                                                                         |
+-------------------------------------------------------------------------+
*/

ENTITY(PIU_VECTOR),

  2 PIU_PTR                    PTR,        /* POINTER TO PIU                */

  2 PIU_LENGTH                 FIXED BINARY(15); /* LENGTH OF PIU           */
```

```
/*
+-------------------------------------------------------------------------+
|                                                                         |
|                  VIRTUAL ROUTE CONTROL BLOCK (VRCB) LIST                |
|                                                                         |
|   FUNCTION:  THIS DATA STRUCTURE CONTAINS THE VIRTUAL ROUTE CONTROL     |
|              BLOCKS.  AN INSTANCE OF THE VRCB IS CREATED BY THE         |
|              VIRTUAL ROUTE MANAGER WHEN A VIRTUAL ROUTE IS ACTIVATED,  |
|              AND IS INITIALIZED BY VALUES OBTAINED FROM THE NC_ACTVR    |
|              REQUEST OR THE ERCB OF THE UNDERLYING ER; IT IS DESTROYED  |
|              BY THE VIRTUAL ROUTE MANAGER WHEN THE VIRTUAL ROUTE IS     |
|              DEACTIVATED.  THE ONLY EXCEPTION TO THIS IS THE VRCB FOR A |
|              VIRTUAL ROUTE ENTIRELY WITHIN THE SUBAREA OF THIS NODE;    |
|              SUCH A VRCB IS CREATED DURING SYSTEM DEFINITION AND        |
|              REMAINS, REPRESENTING AN ACTIVE VR, UNTIL THE NODE IS      |
|              DEACTIVATED; IN SUCH A VRCB, THE FIELDS VRCB_VRID,         |
|              VRCB_ERN, AND VRCB_REVERSE_ERN ALL HAVE VALUE 0.  VRCB'S   |
|              ARE KEPT IN A LIST CALLED VRCB_LIST.                       |
|                                                                         |
+-------------------------------------------------------------------------+
*/

ENTITY(VRCB),

  2 STATES(1:40)                FIXED BIN,   /* FSM STATE INFORMATION          */
                                             /* FSM_DACTVR_DIRECTION           */
                                             /* FSM_SET_CWRI                   */
                                             /* FSM_VR                         */
                                             /* FSM_VRPRQ_SEND                 */
                                             /* FSM_VRPRQ_RCV                  */

  2 VR_ID,                                   /* VIRTUAL ROUTE IDENTIFIER       */
    3 VR_NUM                    BIT(4),      /* VIRTUAL ROUTE NUMBER           */
    3 RESERVED                  BIT(2),
    3 TP_FIELD                  BIT(2),      /* TRANSMISSION PRIORITY          */

  2 PARTNER_SA                  BIT(32),     /* SUBAREA AT OTHER END OF THE VR */

  2 ER_NUM                      BIT(4),      /* EXPLICIT ROUTE NUMBER          */

  2 RER_NUM                     BIT(4),      /* REVERSE EXPLICIT ROUTE NUMBER  */

  2 WINDOW_SIZE                 BIT(8),

  2 MIN_WINDOW_SIZE             BIT(8),

  2 MAX_WINDOW_SIZE             BIT(8),

  2 WINDOW_SIZE_CHANGE          BIT(8),      /* 1 IS THE ONLY VALUE DEFINED    */

  2 PACING_COUNT                BIT(8),      /* REMAINING NUMBER OF PIU'S THAT */
                                             /*   CAN BE SENT                  */

  2 SNF_SEND_CNTR               BIT(12),     /* SNF_SEND FOR PIU'S SENT        */

  2 SNF_RCV_CNTR                BIT(12),     /* SNF_SEND FOR PIU'S RECEIVED    */

  2 Q_VR_PAC                    PTR,         /* POINTER TO VR PACING QUEUE     */

  2 PIU_SEND_LIST               PTR,         /* POINTER TO LIST TO HOLD PIU'S FROM A */
                                             /* HALF-SESSION THAT ARE TO BE SENT */
                                             /* OVER THE VR                    */

  2 ER_VR_SUPP                  BIT(1),      /* THIS BIT HAS VALUE PRE_ER_VR IF AND */
                                             /* ONLY IF THE VRCB REFERS TO A VR THAT */
                                             /* CONTAINS ONE OR MORE NODES THAT DO NOT */
                                             /* SUPPORT EXPLICIT AND VIRTUAL ROUTES */

  2 SESS_COUNT                  BIT(8),      /* A COUNT OF THE SESSIONS        */
                                             /* USING THIS VR--THE LENGTH OF THIS */
                                             /* FIELD IS IMPLEMENTATION-DEPENDENT */

  2 VR_RESERVATION_LIST         PTR;         /* POINTER TO LIST OF SESSION     */
                                             /* ACTIVATION REQUESTS AWAITING   */
                                             /* ACTIVATION OF THIS VR          */
```

```
/*
+-------------------------------------------------------------------+
|                                                                   |
|              VIRTUAL ROUTE RESERVATION (VR_RESERVATION) LIST      |
|                                                                   |
|  FUNCTION:  THIS DATA STRUCTURE STORES A SESSION ACTIVATION REQUEST THAT |
|             REQUIRES A VR ACTIVATION. IT RELATES ALL SESSION ACTIVATION REQUESTS |
|             THAT ARE AWAITING THE ACTIVATION OF A VR TO THE PARTICULAR VRCB |
|             REPRESENTING THE VR. FOR INSTANCE, IT IS CREATED WHEN A SESSION |
|             ACTIVATION REQUEST FROM CSC_MGR CAUSES NC.VR_MGR TO SEND AN |
|             ACTIVATE_ER SIGNAL TO NC.ER_MGR. IT IS DISCARDED WHEN THE VR |
|             BECOMES ACTIVE AND THE SESSION IS ASSIGNED TO THE VR, OR WHEN THE VR |
|             IS RESET, WHICH PRECLUDES ASSIGNING THE SESSION TO THE VR. |
|                                                                   |
|             THIS DATA STRUCTURE IS ALSO THE MEANS BY WHICH THE NC.VR_MGR CAN |
|             ASSOCIATE A SESSION DEACTIVATION REQUEST TO A SESSION ACTIVATION |
|             REQUEST THAT IS PENDING VR ACTIVATION.                |
|                                                                   |
+-------------------------------------------------------------------+
*/

ENTITY(VR_RESERVATION),

  2 SESSION_ACT_RQ              PTR,        /* POINTER TO SESSION ACTIVATION RQ      */

  2 VR_LIST                     PTR,        /* POINTER TO COS_VR_LIST                */

  2 SCBPTR                      PTR,        /* POINTER TO SCB                        */

  2 VR_LIST_INDEX               FIXED BIN(8);/* INDEX OF COS_VR_LIST ENTRY THAT      */
                                            /* WAS BEING PROCESSED WHEN THIS         */
                                            /* ENTITY WAS CREATED                    */
```

```
/*
┌──────────────────────────────────────────────────────────────────────────┐
│                                                                          │
│                    EXPLICIT ROUTE CONTROL BLOCK LIST (ERCB)              │
│                                                                          │
│     FUNCTION:  THIS LIST CONTAINS THE EXPLICIT ROUTE CONTROL BLOCKS. AN  ERCB │
│                ENTITY IS CREATED BY THE EXPLICIT ROUTE MANAGER WHEN AN EXPLICIT │
│                ROUTE BECOMES OPERATIONAL.  THE ER_NUM AND PARTNER_SA ARE │
│                INITIALIZED ON RECEIVING AN NC_ER_OP REQUEST.  THE ER_LEN, │
│                RERN_MASK, AND ER_VR_SUPP ARE INITIALIZED ON RECEIVING AN NC_ER_ACT │
│                OR AN NC_ER_ACT_REPLY REQUEST.  AN ERCB ENTRY IS DELETED AND │
│                DESTROYED WHEN AN EXPLICIT ROUTE BECOMES INOPERATIVE.    │
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
*/

ENTITY(ERCB),

   2 STATES(1)                  FIXED BIN(8),/* FSM STATE INFORMATION      */
                                             /* FSM_ERN                    */

   2 PARTNER_SA                 BIT(32),     /* SUBAREA AT OTHER END OF THE ER */

   2 ER_NUM                     BIT(4),      /* EXPLICIT ROUTE NUMBER      */

   2 RERN_MASK                  BIT(16),     /* A BIT MASK CORRESPONDING TO THE */
                                             /* POSSIBLE REVERSE EXPLICIT ROUTE */
                                             /* NUMBER FOR PIU'S RECEIVED  */
                                             /* FROM THIS SUBAREA          */

   2 ER_LEN                     BIT(8),      /* NUMBER OF TRANSMISSION GROUPS IN */
                                             /* THIS EXPLICIT ROUTE        */

   2 PENDING_VRNUMS             BIT(16),     /* VRN'S TO BE SUPPORTED BY THIS ERN */
                                             /*  WHILE THE ER IS IN THE PROCESS OF */
                                             /*  BEING ACTIVATED, THIS FIELD */
                                             /*  INDICATES THE VRN'S WAITING TO BE */
                                             /*  SUPPORTED BY THIS ER.  AT ALL */
                                             /*  OTHER TIMES THIS FIELD IS RESERVED */

   2 ER_VR_SUPP                 BIT(1),      /* SPECIFIES WHETHER THERE IS A NODE ON */
                                             /* THE ER THAT DOES NOT SUPPORT ER AND */
                                             /* VR PROTOCOLS:              */
                                             /* 1 = PRE_ER_VR              */
                                             /*   THERE IS A NODE ON THE ER THAT DOES */
                                             /*   NOT SUPPORT ER-VR PROTOCOLS. */
                                             /* 0 = ¬PRE_ER_VR             */
                                             /*   EVERY NODE ON THE ER SUPPORTS */
                                             /*   ER-VR PROTOCOLS.         */

   2 PATHCB_LIST                PTR;         /* LIST OF PATHCB ENTITIES    */


/*
┌──────────────────────────────────────────────────────────────────────────┐
│                                                                          │
│                       PATH CONTROL BLOCK (PATHCB) LIST                   │
│                                                                          │
│     FUNCTION:  THIS DATA STRUCTURE PROCESSED ONLY BY THE ER MANAGER, CONTAINS │
│                INFORMATION ABOUT AN EXPLICITY ROUTE ALONG A PARTICULAR │
│                (TRANSMISSION GROUP NUMBER, ADJACENT SUBAREA) ROUTE FROM THIS │
│                SUBAREA NODE.  A PATHCB IS CREATED WHEN AN NC_ER_OP IS RECEIVED, AND │
│                DESTROYED WHEN AN NC_ER_INOP IS RECEIVED.                 │
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
*/

ENTITY(PATHCB),

   2 STATES(1:10)               FIXED BIN(8),/* FSM STATE INFORMATION      */
                                             /* FSM_PATH                   */

   2 TG_ID,

     3 TGN                      BIT(8),      /* TRANSMISSION GROUP NUMBER  */

     3 ADJ_SA                   BIT(32),     /* ADJACENT SUBAREA ADDRESS   */

   2 ACT_SEQ_ID                 CHAR(8);     /* SEQUENCE ID FROM ACTIVATION REQUEST */
```

```
/*
+-------------------------------------------------------------------------+
|                                                                         |
|                    SUBAREA ROUTING (SUBAREA_ROUTING) LIST               |
|                                                                         |
|   FUNCTION:  THE   SUBAREA   ROUTING   LIST   IS   USED   BY   PC.ERC   AND |
|              PU.SVC_MGR.PC_ROUTE_MGR  TO  CHECK  OR DETERMINE  ROUTING  TO  OTHER |
|              SUBAREAS.  IT  IS INITIALIZED DURING  SYSTEM DEFINITION, AND  MAY BE |
|              UPDATED BY PC_ROUTE_MGR.ER_MGR DURING NETWORK OPERATION.   |
|                                                                         |
+-------------------------------------------------------------------------+
*/

ENTITY(SUBAREA_ROUTING),

  2 DEST_SA                     BIT(32),    /* SUBAREA ADDRESS                    */

  2  EXPLICIT_ROUTE(16),

    3 ER_SYSDEF                 BIT(1),     /* B'0' STATIC_DEFINITION--SYSTEM DEFINED */
                                            /* B'1' DYNAMIC_DEFINITION--          */
                                            /*              DEFINED BY NC_ER_OP   */

    3 TG_ID,

      4 TGN                     BIT(8),     /* TGN FOR THIS ERN AND DEST_SA       */

      4 ADJ_SA                  BIT(32);    /* NEXT SUBAREA FOR THIS ERN AND DEST_SA */


/*
+-------------------------------------------------------------------------+
|                                                                         |
|                          ERN MAP (ERN_MAP) LIST                         |
|                                                                         |
|   FUNCTION:  THIS DATA STRUCTURE IS USED TO  PROVIDE A TWO-WAY MAPPING BETWEEN AN |
|              VRN  AND AN  ERN  FOR A  GIVEN  DSA.  IT  IS  INITIALIZED AT  SYSTEM |
|              DEFINITION  TIME  AND  IS ACCESSED  BY  THE  PU.SVC_MGR.PC_ROUTE_MGR |
|              (CHAPTER 12).                                              |
|                                                                         |
+-------------------------------------------------------------------------+
*/

ENTITY(ERN_MAP),

  2 DEST_SA                     BIT(32),    /* DESTINATION SUBAREA FOR THESE      */
                                            /* EXPLICIT ROUTES                    */

  2 ER_NUM(16)                  BIT(4);     /* ERN VALUE FOR THE DESTINATION      */
                                            /* SUBAREA ADDRESS AND VIRTUAL ROUTE  */
                                            /* NUMBER                             */
```

This page intentionally left blank

```
/*
+-------------------------------------------------------------------------+
|                                                                         |
|                  VIRTUAL ROUTE IDENTIFER LIST (VR_ID_LIST)              |
|                                                                         |
|   FUNCTION:  THIS DATA STRUCTURE IS CREATED BY THE LU.SVC_MGR OR THE    |
|              SSCP.SVC_MGR AND IS USED BY THE PU.SVC_MGR.PC_ROUTE_MGR.   |
|              VR_MGR WHEN ASSIGNING A VIRTUAL ROUTE TO A SESSION THAT    |
|              IS BEING ACTIVATED. THE DATA STRUCTURE IS DISCARDED AFTER  |
|              A VR HAS BEEN ASSIGNED.                                    |
|                                                                         |
+-------------------------------------------------------------------------+
*/

ENTITY(VR_ID_LIST),

  2 COS_NAME                      CHAR(8),      /* CLASS OF SERVICE NAME          */
  2 LENGTH_OF_VR_INFO             FIXED BIN(8), /* LENGTH OF REMAINDER OF TABLE   */
  2 FORMAT_OF_VR_INFO             BIT(8),       /* X'00' ONLY VALUE ALLOWED       */
  2 TYPE_OF_VR                    BIT(1),       /* B'0' VR MAPPED TO ER0          */
                                                /* B'1' VR MAY BE MAPPED TO ANY ER*/
  2 NUMBER_OF_VR_IDS              FIXED BIN(8),
  2 VR_ID(1:REFER(NUMBER_OF_VR_IDS)),
    3 VR_NUM                      BIT(4),       /* VIRTUAL ROUTE NUMBER           */
    3 RESERVED                    BIT(2),
    3 TP_FIELD                    BIT(2);       /* TRANSMISSION PRIORITY          */
```

```
/*
+-------------------------------------------------------------------+
|                                                                   |
|                          FAPL CONSTANTS                           |
|                                                                   |
+-------------------------------------------------------------------+
*/

DCL 1 CONST BASED(CON_PTR),
     2 ABCONN                          BIT(8)          CONSTANT(X'0F'),
     2 ABCONNOUT                       BIT(8)          CONSTANT(X'18'),
     2 ACTCDRM                         BIT(8)          CONSTANT(X'14'),
     2 ACTCONNIN                       BIT(8)          CONSTANT(X'16'),
     2 ACTIVE                          BIT(1)          CONSTANT(B'1'),
     2 ACTLINK                         BIT(8)          CONSTANT(X'0A'),
     2 ACTLU                           BIT(8)          CONSTANT(X'0D'),
     2 ACTPU                           BIT(8)          CONSTANT(X'11'),
     2 ACTTRACE                        BIT(8)          CONSTANT(X'02'),
     2 ADDLINK                         BIT(8)          CONSTANT(X'1E'),
     2 ADDLINKSTA                      BIT(8)          CONSTANT(X'21'),
     2 ALL                             BIT(8)          CONSTANT(X'00'),
     2 ALL_NO                          BIT(256)        CONSTANT((256)B'0'),
     2 ALL_ON                          BIT(256)        CONSTANT((256)B'1'),
     2 ALL_OFF                         BIT(256)        CONSTANT((256)B'0'),
     2 ALL_ONES                        BIT(256)        CONSTANT((256)B'1'),
     2 ALL_YES                         BIT(256)        CONSTANT((256)B'1'),
     2 ALL_ZEROS                       BIT(256)        CONSTANT((256)B'0'),
     2 ALLOWED                         BIT(1)          CONSTANT(B'1'),
     2 ALS                             BIT(4)          CONSTANT(B'0100'),
     2 ANA                             BIT(8)          CONSTANT(X'19'),
     2 ANY                             BIT(8)          CONSTANT(X'FF'),
     2 AVAILABLE                       BIT(1)          CONSTANT(B'1'),

     2 BASIC                           BIT(1)          CONSTANT(B'0'),
     2 BB                              BIT(1)          CONSTANT(B'1'),
     2 BBIU                            BIT(1)          CONSTANT(B'1'),
     2 BC                              BIT(1)          CONSTANT(B'1'),
     2 BETB                            BIT(1)          CONSTANT(B'1'),
     2 BF_LU                           BIT(4)          CONSTANT(B'0110'),
     2 BF_PU                           BIT(4)          CONSTANT(B'0101'),
     2 BF_SESS                         BIT(1)          CONSTANT(B'1'),
     2 BID                             BIT(8)          CONSTANT(X'C8'),
     2 BIND                            BIT(8)          CONSTANT(X'31'),
     2 BINDF                           BIT(8)          CONSTANT(X'85'),
     2 BIS                             BIT(8)          CONSTANT(X'70'),
     2 BIU_AVAILABLE                   BIT(1)          CONSTANT(B'1'),
     2 BLOCKING                        BIT(1)          CONSTANT(B'1'),
     2 BLOCK_CHAINING_WITH_SEED        BIT(3)          CONSTANT(B'000'),
     2 BRACKETS_NOT_USED               BIT(1)          CONSTANT(B'0'),
     2 B1                              BIT(1)          CONSTANT(B'0'),
     2 B2                              BIT(1)          CONSTANT(B'1'),

     2 CANCEL                          BIT(8)          CONSTANT(X'83'),
     2 CANCEL_ONLY                     BIT(2)          CONSTANT(B'11'),
     2 CAPABLE                         BIT(1)          CONSTANT(B'1'),
     2 CD                              BIT(1)          CONSTANT(B'1'),
     2 CDCINIT                         BIT(8)          CONSTANT(X'4B'),
     2 CDINIT                          BIT(8)          CONSTANT(X'41'),
     2 CDSESSEND                       BIT(8)          CONSTANT(X'48'),
     2 CDSESSSF                        BIT(8)          CONSTANT(X'45'),
     2 CDSESSST                        BIT(8)          CONSTANT(X'46'),
     2 CDSESSTF                        BIT(8)          CONSTANT(X'47'),
     2 CDTAKED                         BIT(8)          CONSTANT(X'49'),
     2 CDTAKEDC                        BIT(8)          CONSTANT(X'4A'),
     2 CDTERM                          BIT(8)          CONSTANT(X'43'),
     2 CESLOW                          BIT(8)          CONSTANT(X'0C'),
     2 CEXSLOW                         BIT(8)          CONSTANT(X'0D'),
     2 CHAN370                         BIT(8)          CONSTANT(X'02'),
     2 CHASE                           BIT(8)          CONSTANT(X'84'),
     2 CINIT                           BIT(8)          CONSTANT(X'01'),
     2 CLEANUP                         BIT(8)          CONSTANT(X'29'),
     2 CLEAR                           BIT(8)          CONSTANT(X'A1'),
     2 CMD_SENDER                      BIT(8)          CONSTANT(X'00'),
     2 CODE0                           BIT(1)          CONSTANT(B'0'),
     2 CODE1                           BIT(1)          CONSTANT(B'1'),
     2 COLD                            BIT(4)          CONSTANT(B'0001'),
     2 COMPRESSION                     BIT(1)          CONSTANT(B'1'),
     2 CONDITIONAL                     BIT(1)          CONSTANT(B'1'),
     2 CONFIGURATION_SERVICES          BIT(7)          CONSTANT(B'0000010'),
     2 CONFIGURABLE                    BIT(1)          CONSTANT(B'1'),
     2 CONNOUT                         BIT(8)          CONSTANT(X'0E'),
     2 CONT                            BIT(1)          CONSTANT(B'0'),
     2 CONTACT                         BIT(8)          CONSTANT(X'01'),
     2 CONTACTED                       BIT(8)          CONSTANT(X'80'),
     2 CONTINUE                        BIT(1)          CONSTANT(B'1'),
     2 CONVERT_TO_EXR                  BIT(2)          CONSTANT(B'11'),
     2 CRV                             BIT(8)          CONSTANT(X'C0'),
     2 CTERM                           BIT(8)          CONSTANT(X'02'),
```

```
2 DACTCDRM                      BIT(8)          CONSTANT(X'15'),
2 DACTCONNIN                    BIT(8)          CONSTANT(X'17'),
2 DACTLINK                      BIT(8)          CONSTANT(X'0B'),
2 DACTLU                        BIT(8)          CONSTANT(X'0E'),
2 DACTPU                        BIT(8)          CONSTANT(X'12'),
2 DACTTRACE                     BIT(8)          CONSTANT(X'03'),
2 DEBLOCKING                    BIT(1)          CONSTANT(B'1'),
2 DEC_WS                        BIT(1)          CONSTANT(B'1'),
2 DEC_WS_RPLY                   BIT(1)          CONSTANT(B'1'),
2 DEF_OR_EXCP_RESPONSE          BIT(2)          CONSTANT(B'11'),
2 DEF_RESPONSE                  BIT(2)          CONSTANT(B'10'),
2 DELAYED                       BIT(1)          CONSTANT(B'1'),
2 DELETENR                      BIT(8)          CONSTANT(X'1C'),
2 DELIVER                       BIT(8)          CONSTANT(X'12'),
2 DFC                           BIT(2)          CONSTANT(B'10'),
2 DISCARD_HU                    BIT(2)          CONSTANT(B'01'),
2 DISCONTACT                    BIT(8)          CONSTANT(X'02'),
2 DISPSTOR                      BIT(8)          CONSTANT(X'31'),
2 DO_DISCARD                    BIT(1)          CONSTANT(B'1'),
2 DO_NOT_DISCARD                BIT(1)          CONSTANT(B'0'),
2 DO_NOT_TEST_ER                BIT(8)          CONSTANT(X'04'),
2 DONE                          BIT(1)          CONSTANT(B'1'),
2 DR1                           BIT(1)          CONSTANT(B'1'),
2 DR2                           BIT(1)          CONSTANT(B'1'),
2 DSRLST                        BIT(8)          CONSTANT(X'27'),
2 DUAL_SEQ                      BIT(2)          CONSTANT(B'11'),
2 DUMPFINAL                     BIT(8)          CONSTANT(X'08'),
2 DUMPINIT                      BIT(8)          CONSTANT(X'06'),
2 DUMPTEXT                      BIT(8)          CONSTANT(X'07'),
2 DYNAMIC_DEFINITION            BIT(1)          CONSTANT(B'1'),

2 EB                            BIT(1)          CONSTANT(B'1'),
2 EBIU                          BIT(1)          CONSTANT(B'1'),
2 EC                            BIT(1)          CONSTANT(B'1'),
2 ECHOTEST                      BIT(8)          CONSTANT(X'89'),
2 ED                            BIT(1)          CONSTANT(B'1'),
2 EF                            BIT(1)          CONSTANT(B'1'),
2 END_USER                      BIT(2)          CONSTANT(B'10'),
2 ER                            BIT(1)          CONSTANT(B'1'),
2 ER_INOP                       BIT(8)          CONSTANT(X'1D'),
2 ER_INOP_HDR                   BIT(24)         CONSTANT(X'41021D'),
2 ER_TESTED                     BIT(8)          CONSTANT(X'86'),
2 ER_TESTED_HDR                 BIT(24)         CONSTANT(X'410386'),
2 ER_LENGTH_ERROR               BIT(8)          CONSTANT(X'04'),
2 ER_NOT_DEFINED                BIT(8)          CONSTANT(X'06'),
2 ER_RACE                       BIT(8)          CONSTANT(X'01'),
2 ERP                           BIT(4)          CONSTANT(B'0010'),
2 ERROR                         BIT(8)          CONSTANT(X'03'),
2 ESLOW                         BIT(8)          CONSTANT(X'14'),
2 EXCHANGED_PARMS_INCOMPAT      BIT(4)          CONSTANT(B'1000'),
2 EXCP_RESPONSE                 BIT(2)          CONSTANT(B'01'),
2 EXECTEST                      BIT(8)          CONSTANT(X'01'),
2 EXIST                         BIT(1)          CONSTANT(B'1'),
2 EXP                           BIT(1)          CONSTANT(B'1'),
2 EXPEDITED                     BIT(1)          CONSTANT(B'1'),
2 EXSLOW                        BIT(8)          CONSTANT(X'15'),

2 FALSE                         BIT(1)          CONSTANT(B'0'),
2 FIDF                          BIT(4)          CONSTANT(B'1111'),
2 FID0                          BIT(4)          CONSTANT(B'0000'),
2 FID1                          BIT(4)          CONSTANT(B'0001'),
2 FID2                          BIT(4)          CONSTANT(B'0010'),
2 FID3                          BIT(4)          CONSTANT(B'0011'),
2 FID4                          BIT(4)          CONSTANT(B'0100'),
2 FID4_TH_LENGTH                FIXED BIN(15)   CONSTANT(26),
2 FIFO                          BIT(1)          CONSTANT(B'0'),
2 FM_HEADERS                    BIT(1)          CONSTANT(B'1'),
2 FMD                           BIT(2)          CONSTANT(B'00'),
2 FMH                           BIT(1)          CONSTANT(B'1'),
2 FNA                           BIT(8)          CONSTANT(X'1A'),
2 FORCED                        BIT(8)          CONSTANT(X'02'),
2 FORMAT1                       BIT(8)          CONSTANT(X'01'),
2 FORMAT2                       BIT(8)          CONSTANT(X'02'),
2 FOUND                         BIT(1)          CONSTANT(B'1'),
2 FULL_DUPLEX                   BIT(2)          CONSTANT(B'00'),

2 GOOD                          BIT(2)          CONSTANT(B'00'),

2 H_PRTY                        BIT(2)          CONSTANT(B'10'),
2 HALF_SESS                     BIT(1)          CONSTANT(B'0'),
2 HDX_CONTENTION                BIT(2)          CONSTANT(B'01'),
2 HDX_FLIP_FLOP                 BIT(2)          CONSTANT(B'10'),
```

```
2 IDENTIFIERS                BIT(2)          CONSTANT(B'00'),
2 IMMEDIATE                  BIT(1)          CONSTANT(B'0'),
2 INACTIVE                   BIT(1)          CONSTANT(B'0'),
2 INB                        BIT(1)          CONSTANT(B'0'),
2 INC_WS                     BIT(1)          CONSTANT(B'0'),
2 INC_WS_RPLY                BIT(1)          CONSTANT(B'0'),
2 INCOMPATIBLE_STATIONS      BIT(8)          CONSTANT(X'05'),
2 INCOMPATIBLE_WITH_TG       BIT(8)          CONSTANT(X'08'),
2 INIT_OTHER                 BIT(8)          CONSTANT(X'80'),
2 INIT_OTHER_CD              BIT(8)          CONSTANT(X'40'),
2 INIT_SELF                  BIT(8)          CONSTANT(X'81'),
2 INITPROC                   BIT(8)          CONSTANT(X'35'),
2 INOP                       BIT(8)          CONSTANT(X'81'),
2 IPL_SUCCESSFUL             BIT(8)          CONSTANT(X'00'),
2 IPL_REQUIRED               BIT(4)          CONSTANT(X'3'),
2 IPLFINAL                   BIT(8)          CONSTANT(X'05'),
2 IPLINIT                    BIT(8)          CONSTANT(X'03'),
2 IPLTEXT                    BIT(8)          CONSTANT(X'04'),
2 IPR_DISCARDED              BIT(1)          CONSTANT(B'1'),

2 KEY1                       BIT(8)          CONSTANT(X'01'),
2 KEY2                       BIT(8)          CONSTANT(X'02'),
2 KEY3                       BIT(8)          CONSTANT(X'03'),
2 KEY4                       BIT(8)          CONSTANT(X'04'),
2 KEY5                       BIT(8)          CONSTANT(X'05'),
2 KEY8                       BIT(8)          CONSTANT(X'08'),

2 L_PRTY                     BIT(2)          CONSTANT(B'00'),
2 LCP                        BIT(8)          CONSTANT(X'87'),
2 LDREQD                     BIT(8)          CONSTANT(X'37'),
2 LINK                       BIT(4)          CONSTANT(B'0011'),
2 LINK_FAILURE               BIT(4)          CONSTANT(X'2'),
2 LINK_TRACE_WITH_TG         BIT(8)          CONSTANT(X'81'),
2 LOAD                       BIT(8)          CONSTANT(X'00'),
2 LOAD_REQUIRED              BIT(8)          CONSTANT(X'02'),
2 LOADED                     BIT(8)          CONSTANT(X'01'),
2 LOADED_STA                 BIT(8)          CONSTANT(X'04'),
2 LOSER_RESPONSIBLE          BIT(1)          CONSTANT(B'0'),
2 LOST_DATA                  BIT(1)          CONSTANT(B'1'),
2 LSA                        BIT(8)          CONSTANT(X'05'),
2 LU                         BIT(4)          CONSTANT(B'0001'),
2 LU_LU                      BIT(3)          CONSTANT(B'011'),
2 LUSTAT                     BIT(8)          CONSTANT(X'04'),
2 L1                         BIT(1)          CONSTANT(B'0'),
2 L2                         BIT(1)          CONSTANT(B'1'),

2 M_PRTY                     BIT(2)          CONSTANT(B'01'),
2 MAINTENANCE_SERVICES       BIT(7)          CONSTANT(B'0000011'),
2 MANDATORY                  BIT(2)          CONSTANT(B'11'),
2 MANUAL                     BIT(8)          CONSTANT(X'10'),
2 MAX_ER_NUM                 FIXED BIN(15)   CONSTANT(15),
2 MAY_BE_USED                BIT(1)          CONSTANT(B'1'),
2 MAY_NOT_SEND               BIT(1)          CONSTANT(B'0'),
2 MAY_SEND                   BIT(1)          CONSTANT(B'1'),
2 MULTI_OR_DLC_INCOMPATIBLE  BIT(4)          CONSTANT(B'1100'),
2 MULTI_LINK_TG              BIT(1)          CONSTANT(B'1'),
2 MULTIPLE                   BIT(1)          CONSTANT(B'1'),
```

```
2 N_PRTY                        BIT(1)          CONSTANT(B'1'),
2 NC                            BIT(2)          CONSTANT(B'01'),
2 NC_ACTVR                      BIT(8)          CONSTANT(X'0D'),
2 NC_DACTVR                     BIT(8)          CONSTANT(X'0E'),
2 NC_ER_ACT                     BIT(8)          CONSTANT(X'0B'),
2 NC_ER_ACT_REPLY               BIT(8)          CONSTANT(X'0C'),
2 NC_ER_INOP                    BIT(8)          CONSTANT(X'06'),
2 NC_ER_OP                      BIT(8)          CONSTANT(X'0F'),
2 NC_ER_TEST                    BIT(8)          CONSTANT(X'09'),
2 NC_ER_TEST_REPLY              BIT(8)          CONSTANT(X'0A'),
2 NC_IPL_ABORT                  BIT(8)          CONSTANT(X'46'),
2 NC_IPL_FINAL                  BIT(8)          CONSTANT(X'02'),
2 NC_IPL_INIT                   BIT(8)          CONSTANT(X'03'),
2 NC_IPL_TEXT                   BIT(8)          CONSTANT(X'04'),
2 NEG                           BIT(1)          CONSTANT(B'1'),
2 NEG_RSP                       BIT(2)          CONSTANT(B'10'),
2 NEGATIVE                      BIT(1)          CONSTANT(B'1'),
2 NEGOTIABLE                    BIT(4)          CONSTANT(B'0000'),
2 NETWORK_SERVICES              BIT(8)          CONSTANT(B'00000001'),
2 NG                            BIT(1)          CONSTANT(B'0'),
2 NO                            BIT(1)          CONSTANT(B'0'),
2 NO_ASSEMBLY                   BIT(4)          CONSTANT(X'0'),
2 NO_COMPRESSION                BIT(1)          CONSTANT(B'0'),
2 NO_FM_HEADERS                 BIT(1)          CONSTANT(B'0'),
2 NO_RESPONSE                   BIT(2)          CONSTANT(B'00'),
2 NO_RESTRICTION                BIT(16)         CONSTANT(X'0000'),
2 NO_REVERSE_ERN_DEFINED        BIT(8)          CONSTANT(X'02'),
2 NO_ROUTE                      BIT(8)          CONSTANT(X'07'),
2 NO_SNF                        BIT(2)          CONSTANT(B'10'),
2 NO_TG                         BIT(4)          CONSTANT(B'1010'),
2 NO_TRUNCATE                   BIT(1)          CONSTANT(B'0'),
2 NON_FIFO                      BIT(1)          CONSTANT(B'1'),
2 NON_SNA                       BIT(1)          CONSTANT(B'0'),
2 NON_SPECIFIED                 FIXED BIN(15)   CONSTANT(0),
2 NONE                          BIT(2)          CONSTANT(B'00'),
2 NONNEGOTIABLE                 BIT(4)          CONSTANT(B'0001'),
2 NONSEQ_NONSUP                 BIT(2)          CONSTANT(B'00'),
2 NONSEQ_SUP                    BIT(2)          CONSTANT(B'01'),
2 NONSWITCHED                   BIT(1)          CONSTANT(B'0'),
2 NORM                          BIT(1)          CONSTANT(B'0'),
2 NORMAL                        BIT(1)          CONSTANT(B'0'),
2 NOT_ALLOWED                   BIT(1)          CONSTANT(B'0'),
2 NOT_DONE                      BIT(1)          CONSTANT(B'0'),
2 NOT_FOUND                     BIT(1)          CONSTANT(B'0'),
2 NOT_SENT_OR_RECEIVED          BIT(1)          CONSTANT(B'0'),
2 NOT_SPECIFIED                 BIT(32)         CONSTANT((32)B'0'),
2 NOT_USED                      BIT(1)          CONSTANT(B'0'),
2 NOTIFY                        BIT(8)          CONSTANT(X'20'),
2 NS_IPL_ABORT                  BIT(8)          CONSTANT(X'46'),
2 NS_IPL_FINAL                  BIT(8)          CONSTANT(X'45'),
2 NS_IPL_INIT                   BIT(8)          CONSTANT(X'43'),
2 NS_IPL_TEXT                   BIT(8)          CONSTANT(X'44'),
2 NS_LSA_REQUIRED               BIT(1)          CONSTANT(B'1'),
2 NSEQ_NSUP                     BIT(2)          CONSTANT(B'00'),
2 NSEQ_SUP                      BIT(2)          CONSTANT(B'01'),
2 NSH                           BIT(1)          CONSTANT(B'1'),
2 NS_LSA                        BIT(8)          CONSTANT(X'85'),
2 NSPE                          BIT(8)          CONSTANT(X'04'),

2 OFF                           BIT(1)          CONSTANT(B'0'),
2 OK                            BIT(1)          CONSTANT(B'1'),
2 ON                            BIT(1)          CONSTANT(B'1'),
2 ONE                           FIXED BIN(15)   CONSTANT(1),
2 ONE_STAGE                     BIT(1)          CONSTANT(B'0'),
2 OPERATIVE                     BIT(1)          CONSTANT(B'1'),
2 ORDERLY                       BIT(8)          CONSTANT(X'01'),
```

```
2 PAC                          BIT(1)          CONSTANT(B'1'),
2 PAC_CNT_0                    BIT(1)          CONSTANT(B'1'),
2 PAD_4_BITS                   BIT(4)          CONSTANT(B'0000'),
2 PARTIAL_CHAIN                BIT(2)          CONSTANT(B'00'),
2 PD                           BIT(1)          CONSTANT(B'1'),
2 PERIPHERAL_LU                BIT(4)          CONSTANT(B'0110'),
2 PERIPHERAL_PU                BIT(4)          CONSTANT(B'0101'),
2 POS                          BIT(1)          CONSTANT(B'0'),
2 POSITIVE                     BIT(1)          CONSTANT(B'0'),
2 POSITIVE_REPLY               BIT(8)          CONSTANT(X'00'),
2 PRE_ER_VR                    BIT(1)          CONSTANT(B'1'),
2 PRE_ER_VR_SUPPORT            BIT(8)          CONSTANT(X'03'),
2 PRI                          BIT(1)          CONSTANT(B'1'),
2 PRI_SPEAKS_FIRST             BIT(1)          CONSTANT(B'1'),
2 PRI_TO_SEC_TWO               BIT(1)          CONSTANT(B'0'),
2 PRIMARY                      BIT(4)          CONSTANT(B'0001'),
2 PRIMARY_RESPONSIBLE          BIT(1)          CONSTANT(B'0'),
2 PROCEDURE_FAILURE            BIT(8)          CONSTANT(X'02'),
2 PRTY_1                       BIT(4)          CONSTANT(B'0001'),
2 PRTY_2                       BIT(4)          CONSTANT(B'0010'),
2 PRTY_3                       BIT(4)          CONSTANT(B'0011'),
2 PRTY_4                       BIT(4)          CONSTANT(B'0100'),
2 PRIVATE                      BIT(2)          CONSTANT(B'01'),
2 PROCSTAT                     BIT(8)          CONSTANT(X'36'),
2 PROFILE_0                    BIT(8)          CONSTANT(X'00'),
2 PROFILE_1                    BIT(8)          CONSTANT(X'01'),
2 PROFILE_17                   BIT(8)          CONSTANT(X'11'),
2 PROFILE_18                   BIT(8)          CONSTANT(X'12'),
2 PROFILE_2                    BIT(8)          CONSTANT(X'02'),
2 PROFILE_3                    BIT(8)          CONSTANT(X'03'),
2 PROFILE_4                    BIT(8)          CONSTANT(X'04'),
2 PROFILE_5                    BIT(8)          CONSTANT(X'05'),
2 PROFILE_6                    BIT(8)          CONSTANT(X'06'),
2 PROFILE_7                    BIT(8)          CONSTANT(X'07'),
2 PU                           BIT(4)          CONSTANT(B'0000'),
2 PU_TO_PUCP                   BIT(2)          CONSTANT(B'01'),
2 PU_T1                        BIT(4)          CONSTANT(X'1'),
2 PU_T2                        BIT(4)          CONSTANT(X'2'),
2 PU_T4                        BIT(4)          CONSTANT(X'4'),
2 PU_T5                        BIT(4)          CONSTANT(X'5'),
2 PUCP_PU                      BIT(3)          CONSTANT(B'100'),
2 PUCP_TO_PU                   BIT(2)          CONSTANT(B'10'),
2 P1                           BIT(1)          CONSTANT(B'0'),
2 P2                           BIT(1)          CONSTANT(B'1'),

2 QC                           BIT(8)          CONSTANT(X'81'),
2 QEC                          BIT(8)          CONSTANT(X'80'),
2 QR                           BIT(1)          CONSTANT(B'1'),

2 RCVD                         BIT(1)          CONSTANT(B'1'),
2 RECEIVE                      BIT(1)          CONSTANT(B'1'),
2 RECEIVED                     BIT(1)          CONSTANT(B'1'),
2 RECFMS                       BIT(8)          CONSTANT(X'84'),
2 RECMS                        BIT(8)          CONSTANT(X'81'),
2 RECSTOR                      BIT(8)          CONSTANT(X'34'),
2 RECTD                        BIT(8)          CONSTANT(X'82'),
2 RECTR                        BIT(8)          CONSTANT(X'85'),
2 RECTRD                       BIT(8)          CONSTANT(X'83'),
2 RELQ                         BIT(8)          CONSTANT(X'82'),
2 REQACTLU                     BIT(8)          CONSTANT(X'40'),
2 REQCONT                      BIT(8)          CONSTANT(X'84'),
2 REQDISCONT                   BIT(8)          CONSTANT(X'1B'),
2 REQDISCONT_IMMEDIATE         BIT(8)          CONSTANT(X'08'),
2 REQDISCONT_NORMAL            BIT(8)          CONSTANT(X'00'),
2 REQECHO                      BIT(8)          CONSTANT(X'87'),
2 REQFNA                       BIT(8)          CONSTANT(X'86'),
2 REQMS                        BIT(8)          CONSTANT(X'04'),
2 REQTEST                      BIT(8)          CONSTANT(X'80'),
2 RERN_MUST_BE_ZERO            BIT(8)          CONSTANT(X'00'),
2 RESERVED_ZERO                CHAR(256)       CONSTANT(256(X'00')),
2 RESET_WS                     BIT(1)          CONSTANT(B'1'),
2 RH_LENGTH                    FIXED BIN(15)   CONSTANT(3),
2 RNAA                         BIT(8)          CONSTANT(X'10'),
2 RNAA_BF_LU                   BIT(8)          CONSTANT(X'01'),
2 RNAA_BF_PU                   BIT(8)          CONSTANT(X'00'),
2 RNAA_LU                      BIT(8)          CONSTANT(X'02'),
2 ROUTE_TEST                   BIT(8)          CONSTANT(X'06'),
2 ROUTE_TEST_HDR               BIT(24)         CONSTANT(X'410306'),
2 RPO                          BIT(8)          CONSTANT(X'09'),
2 RQ                           BIT(1)          CONSTANT(B'0'),
2 RQ_NG                        BIT(2)          CONSTANT(B'01'),
2 RQ_OK                        BIT(2)          CONSTANT(B'10'),
2 RQ_WRONG_VR                  BIT(2)          CONSTANT(B'11'),
2 RQR                          BIT(8)          CONSTANT(X'A3'),
2 RSHUTD                       BIT(8)          CONSTANT(X'C2'),
2 RSP                          BIT(1)          CONSTANT(B'1'),
2 RSP_NG                       BIT(2)          CONSTANT(B'01'),
2 RSP_OF_LENGTH_ONE            FIXED BIN(15)   CONSTANT(4),
2 RSP_OF_LENGTH_TWO            FIXED BIN(15)   CONSTANT(5),
2 RSP_OK                       BIT(2)          CONSTANT(B'10'),
2 RSP_SENDER                   BIT(8)          CONSTANT(X'07'),
2 RTR                          BIT(8)          CONSTANT(X'05'),
```

```
2 SBI                                BIT(8)          CONSTANT(X'71'),
2 SC                                 BIT(2)          CONSTANT(B'11'),
2 SD                                 BIT(1)          CONSTANT(B'1'),
2 SDLC                               BIT(8)          CONSTANT(X'01'),
2 SDT                                BIT(8)          CONSTANT(X'A0'),
2 SEC                                BIT(1)          CONSTANT(B'0'),
2 SEC_SPEAKS_FIRST                   BIT(1)          CONSTANT(B'0'),
2 SEC_TO_PRI_TWO                     BIT(1)          CONSTANT(B'1'),
2 SECONDARY                          BIT(4)          CONSTANT(B'0000'),
2 SELECTIVE                          BIT(2)          CONSTANT(B'01'),
2 SEND                               BIT(1)          CONSTANT(B'0'),
2 SEND_FOR_PRI                       BIT(1)          CONSTANT(B'1'),
2 SEND_FOR_SEC                       BIT(1)          CONSTANT(B'0'),
2 SEND_OR_RECEIVE_CHECK              FIXED BIN(15)   CONSTANT(1),
2 SENDER_RESPONSIBLE                 BIT(1)          CONSTANT(B'1'),
2 SENSE_LENGTH                       FIXED BIN(15)   CONSTANT(4),
2 SENT                               BIT(1)          CONSTANT(B'1'),
2 SENT_OR_RECEIVED                   BIT(1)          CONSTANT(B'1'),
2 SEQUENCE_NUMBERS                   BIT(2)          CONSTANT(B'01'),
2 SESSEND                            BIT(8)          CONSTANT(X'88'),
2 SESSION_ASSEMBLY                   BIT(4)          CONSTANT(X'2'),
2 SESSST                             BIT(8)          CONSTANT(X'86'),
2 SET                                BIT(2)          CONSTANT(B'01'),
2 SET_AND_TEST                       BIT(2)          CONSTANT(B'11'),
2 SETCV                              BIT(8)          CONSTANT(X'11'),
2 SHUTC                              BIT(8)          CONSTANT(X'C1'),
2 SHUTD                              BIT(8)          CONSTANT(X'C0'),
2 SIG                                BIT(8)          CONSTANT(X'C9'),
2 SIM                                BIT(1)          CONSTANT(B'1'),
2 SING_SEQ                           BIT(2)          CONSTANT(B'10'),
2 SINGLE                             BIT(1)          CONSTANT(B'0'),
2 SINGLE_SEQ                         BIT(2)          CONSTANT(B'10'),
2 SLU_KEY                            BIT(2)          CONSTANT(B'00'),
2 SNA                                BIT(1)          CONSTANT(B'1'),
2 SNA_LINK                           BIT(4)          CONSTANT(B'0000'),
2 SSCP                               BIT(4)          CONSTANT(B'0010'),
2 SSCP_LU                            BIT(3)          CONSTANT(B'001'),
2 SSCP_PU                            BIT(3)          CONSTANT(B'000'),
2 SSCP_SSCP                          BIT(3)          CONSTANT(B'010'),
2 STATIC_DEFINITION                  BIT(1)          CONSTANT(B'0'),
2 STATIC_ONLY                        BIT(1)          CONSTANT(B'0'),
2 STATION_ASSEMBLY                   BIT(4)          CONSTANT(X'1'),
2 STOP                               BIT(1)          CONSTANT(B'0'),
2 STSN                               BIT(8)          CONSTANT(X'A2'),
2 SUBAREA_LU                         BIT(4)          CONSTANT(B'0001'),
2 SUBAREA_NODE                       BIT(4)          CONSTANT(B'0010'),
2 SUBAREA_PU                         BIT(4)          CONSTANT(B'0010'),
2 SUBSEQUENT_LINK_PARMS_INCOMPAT BIT(4)              CONSTANT(B'1001'),
2 SUPPORTED                          BIT(1)          CONSTANT(B'1'),
2 SWEEP                              BIT(1)          CONSTANT(B'1'),
2 SWITCHED                           BIT(1)          CONSTANT(B'1'),
2 SYMMETRIC                          BIT(1)          CONSTANT(B'1'),
2 S1                                 BIT(1)          CONSTANT(B'0'),
2 S2                                 BIT(1)          CONSTANT(B'1'),

2 TERM_OTHER                         BIT(8)          CONSTANT(X'82'),
2 TERM_OTHER_CD                      BIT(8)          CONSTANT(X'42'),
2 TERM_SELF                          BIT(8)          CONSTANT(X'83'),
2 TEST_DEFINED_ERS                   BIT(8)          CONSTANT(X'03'),
2 TEST_ERS                           BIT(8)          CONSTANT(X'01'),
2 TEST_INOP                          BIT(8)          CONSTANT(X'03'),
2 TEST_NOT_INOP                      BIT(8)          CONSTANT(X'02'),
2 TEST_REGARDLESS                    BIT(8)          CONSTANT(X'01'),
2 TEST_VRS                           BIT(8)          CONSTANT(X'02'),
2 TESTMODE                           BIT(8)          CONSTANT(X'05'),
2 TG_CMD                             BIT(8)          CONSTANT(X'01'),
2 TG_INOPERATIVE                     BIT(8)          CONSTANT(X'05'),
2 TG_SNF_WRAP_ACK                    BIT(8)          CONSTANT(X'01'),
2 TRACE                              BIT(1)          CONSTANT(B'1'),
2 TRUNCATE                           BIT(1)          CONSTANT(B'1'),
2 TRUE                               BIT(1)          CONSTANT(B'1'),
2 TWA                                BIT(2)          CONSTANT(B'00'),
2 TWO_STAGE                          BIT(1)          CONSTANT(B'1'),
2 TWS                                BIT(2)          CONSTANT(B'01'),
2 T1                                 BIT(4)          CONSTANT(B'0001'),
2 T2                                 BIT(4)          CONSTANT(B'0010'),
2 T4                                 BIT(4)          CONSTANT(B'0100'),
2 T5                                 BIT(4)          CONSTANT(B'0101'),

2 UNBIND                             BIT(8)          CONSTANT(X'32'),
2 UNCONDITIONAL                      BIT(1)          CONSTANT(B'0'),

2 VR_INOP                            BIT(8)          CONSTANT(X'23'),
2 VR_INOP_HDR                        BIT(24)         CONSTANT(X'410223'),
2 VR_PAC_RQ                          BIT(1)          CONSTANT(B'1'),
2 VR_PAC_RSP                         BIT(1)          CONSTANT(B'1'),

2 WHOLE_CHAIN_NO_CANCEL              BIT(2)          CONSTANT(B'01'),
2 WHOLE_CHAIN_WITH_CANCEL            BIT(2)          CONSTANT(B'10'),

2 YES                                BIT(1)          CONSTANT(B'1'),

2 ZERO                               FIXED BIN(15)   CONSTANT(0);
```

This page intentionally left blank

APPENDIX B. NODE UTILITY PROCEDURES

This appendix is a collection of utility procedures. They represent functions that deal with the data structures defined in Appendix A. These utility procedures are not an architectural definition. The FAPL procedures are in alphabetical order. Each procedure describes its function and calling parameters.

```
ADD_CP_ENTRY: PROCEDURE(RESOURCE_ADDR,CP_SESS_ID);

                                                                              /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  SEARCHES THE NODE RESOURCE LIST TO FIND A RESOURCE ENTRY THAT │
  │                 CORRESPONDS TO THE RESOURCE_ADDR AND ADDS A CP_INDIRECT ENTRY TO │
  │                 THAT NODE RESOURCE'S ASSOCIATED CP_INDIRECT LIST.  IF THE ENTRY │
  │                 EXISTS ALREADY, IT IS NOT ADDED AGAIN.  IF THE CP_INDIRECT_LIST DOES │
  │                 NOT EXIST, IT IS CREATED.                                │
  │                                                                          │
  │      INPUT:     THE ELEMENT ADDRESS OF THE RESOURCE TO BE ASSOCIATED WITH THE CP-PU │
  │                 SESSION IDENTIFIER                                       │
  │                                                                          │
  │      OUTPUT:    NONE                                                     │
  │                                                                          │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
  │                 FIND_CP_ENTRY                PAGE B-10                   │
  │                 LOCATE_NODE_RESOURCE         PAGE B-14                   │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL RESOURCE_ADDR BIT(16);
  DCL CP_SESS_ID PTR;
  DCL CP_TEMP_LIST PTR;
  DCL P PTR;

  FIND CPCB IN CPCB_LIST WHERE(CPCB.CP_SCB_ID = CP_SESS_ID);

  IF CPCB_PTR ¬= NULL &
       FIND_CP_ENTRY(RESOURCE_ADDR,CP_SESS_ID) = NO THEN        /* PAGE B-10    */
    DO;
    .
    . P = LOCATE_NODE_RESOURCE(RESOURCE_ADDR);                  /* PAGE B-14    */
    .
    . IF P ¬= NULL THEN
    .   DO;
    .   .
    .   . CP_TEMP_LIST = P->NRCB.CP_INDIRECT_LIST;
    .   .
    .   . IF CP_TEMP_LIST = NULL THEN
    .   .   DO;
    .   .   . NEWLIST CP_TEMP_LIST ENTRY_NAME(CP_INDIRECT);
    .   .   . P->NRCB.CP_INDIRECT_LIST = CP_TEMP_LIST;
    .   .   END;
    .   .
    .   . CREATE CP_INDIRECT;
    .   . CP_INDIRECT.CP_ENTRY_PTR = CPCB_PTR;
    .   . INSERT CP_INDIRECT IN CP_TEMP_LIST;
    .   .
    .   END;
    END;

  RETURN;

END ADD_CP_ENTRY;



CHANGE_MU_TO_EXR: PROCEDURE(SENSE_DATA);

                                                                              /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE CONVERTS THE CURRENT MU TO AN EXCEPTION REQUEST BY │
  │                 CHANGING THE SDI, EBIUI, BBIUI BITS AND THE SNC FIELD.  THE RU IS │
  │                 TRUNCATED TO THREE BYTES IF IT IS LONGER THAN THAT AND THE DCF FIELD │
  │                 IS SET TO THE LENGTH OF THE RU (INCLUDING THE 4 BYTES FOR THE SENSE │
  │                 DATA).                                                   │
  │                                                                          │
  │      INPUT:     THE PROCEDURE ASSUMES THAT MU_PTR IS SET TO THE REQUEST TO BE │
  │                 CHANGED.  ONLY REQUESTS CAN BE CHANGED TO EXCEPTION REQUESTS.  THE │
  │                 32-BIT PARAMETER INDICATES THE VALUE TO BE SET IN SNC.   │
  │                                                                          │
  │      OUTPUT:    THE PROCEDURE CHANGES THE CURRENT MU TO AN EXCEPTION REQUEST. │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL SENSE_DATA BIT(32);

  SDI = SD;
  SNC = SENSE_DATA;
  BBIUI = BBIU;
  EBIUI = EBIU;

  IF DCF - RH_LENGTH < 3 THEN
    DCF = DCF + SENSE_LENGTH;
  ELSE
    DCF = RH_LENGTH + SENSE_LENGTH + 3;

END CHANGE_MU_TO_EXR;
```

```
CHANGE_MU_TO_NEG_RSP: PROCEDURE(SENSE_DATA);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE  CONVERTS THE  CURRENT MU TO  A NEGATIVE  RESPONSE BY │
   │                 CHANGING THE RRI,  RTI, SDI BITS AND  THE SNC FIELD.  SDI  IS SET TO │
   │                 INDICATE THAT NO SENSE DATA IS  INCLUDED.  BCI AND ECI INDICATE THAT │
   │                 THE  MU  CONSTITUTES A  SINGLE-ELEMENT  CHAIN  AND BBIUI  AND  EBIUI │
   │                 INDICATE THE RESPONSE  IS A WHOLE BIU.  ALL  OF BYTE 2 OF  THE TH IS │
   │                 SET TO  ZERO.  THE RU  IS TRUNCATED TO THREE  BYTES IF IT  IS LONGER │
   │                 THAN THAT.  THE DCF FIELD IS SET  TO THE LENGTH OF THE RU (INCLUDING │
   │                 THE 4 BYTES FOR THE SENSE DATA).                     │
   │                                                                      │
   │      INPUT:     THE PROCEDURE ASSUMES  THAT MU_PTR IS SET  TO THE MU TO  BE CHANGED. │
   │                 THE 32-BIT PARAMETER INDICATES THE VALUE TO BE SET IN SNC.           │
   │                                                                      │
   │      OUTPUT:    THE PROCEDURE CHANGES THE CURRENT MU TO A NEGATIVE RESPONSE.         │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL SENSE_DATA BIT(32);

  RRI = RSP;
  RTI = NEGATIVE;
  SDI = SD;
  SNC = SENSE_DATA;
  BCI = BC;
  ECI = EC;
  BBIUI = BBIU;
  EBIUI = EBIU;
  BBI =  ¬BB;
  EBI =  ¬EB;
  CDI =  ¬CD;
  CSI = CODE0;
  EDI =  ¬ED;
  PDI =  ¬PD;

  IF DCF - RH_LENGTH < 3 THEN
    DCF = DCF + SENSE_LENGTH;
  ELSE
    DCF = RH_LENGTH + SENSE_LENGTH + 3;

  MUCB.DIRECTION = SEND;

END CHANGE_MU_TO_NEG_RSP;
```

```
CHANGE_MU_TO_POS_RSP: PROCEDURE(TRUNCATION);

                                                                              /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE CONVERTS THE CURRENT MU TO A POSITIVE RESPONSE BY │
   │                 CHANGING THE RRI AND RTI BITS.  SDI IS SET TO INDICATE THAT NO SENSE │
   │                 DATA IS INCLUDED.  BCI AND ECI INDICATE THAT THE MU CONSTITUTES A │
   │                 SINGLE-ELEMENT CHAIN AND BBIUI AND EBIUI INDICATE THE RESPONSE IS A │
   │                 WHOLE BIU.  ALL OF BYTE 2 OF THE TH IS SET TO 0.  IF TRUNCATION IS │
   │                 SPECIFIED, THE DCF FIELD IS UPDATED ACCORDING TO THE ARCHITECTED │
   │                 RULES AS DEFINED IN APPENDIX E.                          │
   │                                                                          │
   │      INPUT:     THE PROCEDURE ASSUMES THAT MU_PTR IS SET TO THE MU TO BE CHANGED AND │
   │                 THE SCB_PTR IS SET TO THE CORRECT HALF-SESSION.  THE BOOLEAN │
   │                 PARAMETER INDICATES IF RU TRUNCATION IS WANTED.  A TRUE VALUE │
   │                 REQUESTS TRUNCATION.                                     │
   │                                                                          │
   │      OUTPUT:    THE PROCEDURE CHANGES THE CURRENT MU TO A POSITIVE RESPONSE. │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL TRUNCATION BIT(1);

  RRI = RSP;
  RTI = POSITIVE;
  SDI =  ¬SD;
  BCI = BC;
  ECI = EC;
  BBIUI = BBIU;
  EBIUI = EBIU;
  BBI =  ¬BB;
  EBI =  ¬EB;
  CDI =  ¬CD;
  CSI = CODE0;
  EDI =  ¬ED;
  PDI =  ¬PD;

  IF TRUNCATION = TRUNCATE THEN
    SELECT ANYORDER;
    .
    . WHEN(RU_CTGY = (NC | SC | DFC))
    .   DCF = RH_LENGTH + 1;
    .
    . WHEN(RU_CTGY = FMD)
    .   IF SCB.TYPE_OF_SESSION ¬= LU_LU &
    .        RQ_CODE = (X'01' | X'41' | X'81') & FI = NSH THEN
    .     DCF = RH_LENGTH + 3;
    .   ELSE
    .     DCF = RH_LENGTH;
    .
    END;

  MUCB.DIRECTION = SEND;

END CHANGE_MU_TO_POS_RSP;



DELETE_ALL_CP_ENTRIES: PROCEDURE(RESOURCE_ADDR);

                                                                              /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  SEARCHES THE NRCB LIST TO FIND A RESOURCE ENTRY THAT CORRESPONDS TO │
   │                 THE RESOURCE_ADDR AND DELETES ALL THE ENTRIES FROM THAT │
   │                 CP_INDIRECT_LIST.                                        │
   │                                                                          │
   │      INPUT:     THE ELEMENT ADDRESS OF THE RESOURCE                      │
   │                                                                          │
   │      OUTPUT:    NONE                                                     │
   │                                                                          │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
   │                 LOCATE_NODE_RESOURCE              PAGE B-14              │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL RESOURCE_ADDR BIT(16);
  DCL P PTR;

  P = LOCATE_NODE_RESOURCE(RESOURCE_ADDR);                  /* PAGE B-14       */

  IF P ¬= NULL THEN
    DESTROY P->NRCB.CP_INDIRECT_LIST;

  RETURN;

END DELETE_ALL_CP_ENTRIES;
```

```
DELETE_ALS_FROM_TGCB: PROCEDURE(ALS_EA);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  SEARCHES THE  ACT_LSCB LIST IN  THE CURRENT  TGCB TO FIND  THE ENTRY
   │                 THAT CORRESPONDS TO THE ADJACENT LINK STATION ADDRESS AND REMOVES IT
   │                 FROM THE ACT_TGCB_LIST FOR THE TGCB.
   │
   │      INPUT:     THE ELEMENT ADDRESS OF THE ADJACENT  LINK STATION TO BE REMOVED FROM
   │                 THE CURRENT TGCB.
   │
   │      OUTPUT:    NONE
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL ALS_EA BIT(16);
  DCL P PTR;

  SCAN TGCB.ASSOC_LSCB_LIST PTR(P);
  .
  . IF P->ASSOC_LSCB_ENTITY.LSCBPTR->LSCB.EA '=' ALS_EA THEN
  .   REMOVE ASSOC_LSCB_ENTITY FROM TGCB.ASSOC_LSCB_LIST DISCARD;
  .
  SCANEND;

  RETURN;

END DELETE_ALS_FROM_TGCB;



DELETE_CP_ENTRY: PROCEDURE(RESOURCE_ADDR,CP_SESS_ID);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  SEARCHES THE NRCB LIST TO FIND  A RESOURCE ENTRY THAT CORRESPONDS TO
   │                 THE RESOURCE_ADDR AND DELETES THE CP_INDIRECT ENTRY CORRESPONDING TO
   │                 THE CP_SESS_ID  FROM THE  CP_INDIRECT LIST  CHAIN.  IF  THE CP_ENTRY
   │                 DOES NOT  EXIST, NOTHING IS DONE.   IF THE CP_INDIRECT  LIST BECOMES
   │                 EMPTY IT IS DESTROYED.
   │
   │      INPUT:     THE  ELEMENT ADDRESS  OF  THE RESOURCE  AND  THE CP-PU  HALF-SESSION
   │                 IDENTIFIER.
   │
   │      OUTPUT:    NONE
   │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):
   │                 FIND_CP_ENTRY                    PAGE B-10
   │                 LOCATE_NODE_RESOURCE             PAGE B-14
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RESOURCE_ADDR BIT(16);
  DCL CP_SESS_ID PTR;
  DCL P PTR;
  DCL TEMP_PTR PTR;

  IF FIND_CP_ENTRY(RESOURCE_ADDR,CP_SESS_ID) = OK THEN          /* PAGE B-10      */
    DO;
    .
    . P = LOCATE_NODE_RESOURCE(RESOURCE_ADDR);                  /* PAGE B-14      */
    .
    . SCAN P->NRCB.CP_INDIRECT_LIST PTR(CP_INDIRECT_PTR);
    . . TEMP_PTR = CP_INDIRECT.CP_ENTRY_PTR;
    . . IF CP_SESS_ID = TEMP_PTR->CPCB.CP_SCB_ID THEN
    . .   REMOVE CP_INDIRECT FROM P->NRCB.CP_INDIRECT_LIST DISCARD;
    . SCANEND;
    .
    . IF EMPTY(P->NRCB.CP_INDIRECT_LIST) THEN
    .   DESTROY P->NRCB.CP_INDIRECT_LIST;
    END;

  RETURN;

END DELETE_CP_ENTRY;
```

```
DEQUEUE_RUS_FROM_RESOURCE: PROCEDURE(RES_EA);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  DEQUEUES ALL RUS THAT  HAVE BEEN QUEUED FOR A LINK  OR ADJACENT LINK  │
   │                 STATION.  THESE ARE  QUEUED WHEN A REQUEST IS RECEIVED  PRIOR TO THE  │
   │                 LINK OR ADJACENT LINK STATION  COMPLETING ITS RESET.  THIS PROCEDURE  │
   │                 IS CALLED WHEN THE LINK OR ADJACENT LINK STATION RESET IS COMPLETE.   │
   │                                                                              │
   │      INPUT:     THE ELEMENT ADDRESS OF THE RESOURCE                          │
   │                                                                              │
   │      OUTPUT:    RUS TO PU.SVC_MGR.NS.RCV;                                    │
   │                                                                              │
   │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
   │                 LOCATE_NODE_RESOURCE              PAGE B-14                  │
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL P PTR;
  DCL P1 PTR;
  DCL RES_EA BIT(16);
  DCL MU_LIST PTR;

  P = LOCATE_NODE_RESOURCE(RES_EA);                    /* PAGE B-14           */

  P1 = MU_PTR;

  MU_LIST = P->NRCB.SAVE_MU_FOR_RETRY_LIST;

  SCAN MU_LIST PTR(MU_PTR);
  . REMOVE MU FROM MU_LIST;
  . SEND MU TO PU.SVC_MGR.NS.RCV;                      /* CHAPTER 11          */
  SCANEND;

  MU_PTR = P1;

  RETURN;

END DEQUEUE_RUS_FROM_RESOURCE;
```

```
DETERMINE_LCP_RESET_OPTION: PROCEDURE(RES_EA) RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  DETERMINES THE RESET OPTION FOR A PU, LINK, ALS, OR BF.PU DETERMINED │
  │                 BY THE OPTIONS SPECIFIED FOR RESOURCES  HIGHER IN THE HIERARCHY WHEN │
  │                 THE SESSION WITH THE CP IS DEACTIVATED.                  │
  │                                                                          │
  │      INPUT:     THE ELEMENT ADDRESS OF THE RESOURCE                      │
  │                                                                          │
  │      OUTPUT:    THE RESET OPTION APPROPRIATE FOR THE RESOURCE            │
  │                                                                          │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
  │                 FIND_ALS_FOR_RESOURCE          PAGE B-9                  │
  │                 FIND_LINK_FOR_RESOURCE         PAGE B-12                 │
  │                 LOCATE_NODE_RESOURCE           PAGE B-14                 │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL P PTR;
  DCL RES_EA BIT(16);
  DCL RESET_OPT BIT(1);

  RESET_OPT = CONTINUE;

  P = LOCATE_NODE_RESOURCE(RES_EA);                          /* PAGE B-14      */

  SELECT ANYORDER(P->NRCB.RESOURCE_CATEGORY);
  .
  . WHEN(PU)
  .   RESET_OPT = P->NRCB.LCP_RESET_OPTION;
  .
  . WHEN(LINK)
  .   DO;
  .   . RESET_OPT = P->NRCB.LCP_RESET_OPTION;
  .   . P = LOCATE_NODE_RESOURCE(NCB.PU_EA);                 /* PAGE B-14      */
  .   . IF P->NRCB.LCP_RESET_OPTION = STOP THEN
  .   .   RESET_OPT = STOP;
  .   END;
  .
  . WHEN(ALS)
  .   DO;
  .   . RESET_OPT = P->NRCB.LCP_RESET_OPTION;
  .   . P = FIND_LINK_FOR_RESOURCE(RES_EA);                 /* PAGE B-12      */
  .   . IF P->NRCB.LCP_RESET_OPTION = STOP THEN
  .   .   RESET_OPT = STOP;
  .   . P = LOCATE_NODE_RESOURCE(NCB.PU_EA);                 /* PAGE B-14      */
  .   . IF P->NRCB.LCP_RESET_OPTION = STOP THEN
  .   .   RESET_OPT = STOP;
  .   END;
  .
  . WHEN(BF.PU)
  .   DO;
  .   . RESET_OPT = P->NRCB.LCP_RESET_OPTION;
  .   . P = FIND_ALS_FOR_RESOURCE(RES_EA);                  /* PAGE B-9       */
  .   . IF P->NRCB.LCP_RESET_OPTION = STOP THEN
  .   .   RESET_OPT = STOP;
  .   . P = FIND_LINK_FOR_RESOURCE(RES_EA);                 /* PAGE B-12      */
  .   . IF P->NRCB.LCP_RESET_OPTION = STOP THEN
  .   .   RESET_OPT = STOP;
  .   . P = LOCATE_NODE_RESOURCE(NCB.PU_EA);                 /* PAGE B-14      */
  .   . IF P->NRCB.LCP_RESET_OPTION = STOP THEN
  .   .   RESET_OPT = STOP;
  .   END;
  .
  END;

  RETURN(RESET_OPT);

END DETERMINE_LCP_RESET_OPTION;
```

```
ENQUEUE_RU_FOR_RESOURCE: PROCEDURE(RES_EA);

                                                                              /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  ENQUEUES A  REQUEST FOR A  LINK OR  ADJACENT LINK STATION.  THIS IS  │
  │                 DONE WHEN A REQUEST  IS RECEIVED PRIOR TO THE LINK  OR ADJACENT LINK │
  │                 STATION COMPLETING ITS RESET.                            │
  │                                                                          │
  │      INPUT:     THE ELEMENT ADDRESS OF THE RESOURCE                      │
  │                                                                          │
  │      OUTPUT:    NONE                                                     │
  │                                                                          │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
  │                 LOCATE_NODE_RESOURCE              PAGE B-14              │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL P PTR;
  DCL RES_EA BIT(16);
  DCL MU_LIST PTR;

  P = LOCATE_NODE_RESOURCE(RES_EA);                    /* PAGE B-14          */

  IF P->NRCB.SAVE_MU_FOR_RETRY_LIST = NULL THEN
    DO;
    . NEWLIST MU_LIST ENTRY_NAME(MU);
    . P->NRCB.SAVE_MU_FOR_RETRY_LIST = MU_LIST;
    END;

  INSERT MU LAST IN P->NRCB.SAVE_MU_FOR_RETRY_LIST;

  RETURN;

END ENQUEUE_RU_FOR_RESOURCE;



FIND_ALS_FOR_DOM_RES: PROCEDURE(RES_NA) RETURNS(POINTER);

                                                                              /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE SEARCHES THE DOMAIN RESOURCE LIST TO FIND AN ADJACENT │
  │                 LINK  STATION CORRESPONDING  TO THE  ADDRESS PASSED  IN RES_NA.  IT  │
  │                 RETURNS TO THE INVOKING PROCEDURE THE POINTER TO THE LINK RESOURCE.  │
  │                                                                          │
  │      INPUT:     THE NETWORK ADDRESS OF THE DOMAIN RESOURCE               │
  │                                                                          │
  │      OUTPUT:    THE  POINTER  TO THE  ADJACENT  LINK  STATION CORRESPONDING  TO  THE  │
  │                 ADDRESS PASSED IF AN ENTRY EXISTS; OTHERWISE, A NULL POINTER │
  │                                                                          │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
  │                 FIND_DOMAIN_RESOURCE              PAGE B-10              │
  │                 FIND_SUBORDINATE_DOM_RES          PAGE B-13              │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL P POINTER;
  DCL RES_NA BIT(48);
  DCL RETURN_PTR POINTER;

  RETURN_PTR = NULL;
  P = FIND_DOMAIN_RESOURCE(RES_NA);                    /* PAGE B-10          */

  IF P ¬= NULL &
       P->DRCB.RESOURCE_CATEGORY = LINK THEN
    P = FIND_SUBORDINATE_DOM_RES(P->DRCB.NETWORK_ADDRESS);
                                                       /* PAGE B-13          */
  IF P ¬= NULL &
       P->DRCB.RESOURCE_CATEGORY = BF.LU THEN
    P = P->DRCB.ASSOCIATED_RES_PTR;

  IF P ¬= NULL &
       P->DRCB.RESOURCE_CATEGORY = BF.PU THEN
    P = P->DRCB.ASSOCIATED_RES_PTR;

  IF P ¬= NULL &
       P->DRCB.RESOURCE_CATEGORY = ALS THEN
    RETURN_PTR = P;

  RETURN(RETURN_PTR);

END FIND_ALS_FOR_DOM_RES;
```

```
FIND_ALS_FOR_RESOURCE: PROCEDURE(RES_EA) RETURNS(POINTER);

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  SEARCHES  THE  NRCB_LIST  TO  FIND  AN  ADJACENT LINK  STATION │
  │                 CORRESPONDING TO RES_EA.  IF ONE IS FOUND, IT RETURNS THE POINTER TO │
  │                 THE  LINK RESOURCE  TO  THE INVOKING  PROCEDURE;  OTHERWISE, A  NULL │
  │                 POINTER IS RETURNED.   THIS PROCEDURE ACCEPTS A  LINK, BF.LU, BF.PU, │
  │                 OR ADJACENT LINK  STATION ELEMENT ADDRESS.  IT RETURNS  A POINTER TO │
  │                 AN ADJACENT LINK STATION IF ONE CAN BE FOUND.                │
  │                                                                      │
  │      INPUT:     THE ELEMENT ADDRESS OF THE RESOURCE                  │
  │                                                                      │
  │      OUTPUT:    THE  POINTER  TO THE  ADJACENT  LINK  STATION CORRESPONDING  TO  THE │
  │                 ADDRESS PASSED IF AN ENTRY EXISTS; OTHERWISE A NULL POINTER  │
  │                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
  │                 DETERMINE_LCP_RESET_OPTION          PAGE B-7         │
  │                                                                      │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                           │
  │                 LOCATE_NODE_RESOURCE                PAGE B-14        │
  │                 LOCATE_SUBORDINATE_RESOURCE         PAGE B-15        │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL P PTR;
  DCL RES_EA BIT(16);
  DCL RETURN_PTR PTR;

  P = LOCATE_NODE_RESOURCE(RES_EA);                          /* PAGE B-14   */

  IF P ¬= NULL &
       P->NRCB.RESOURCE_CATEGORY = LINK THEN
    P = LOCATE_SUBORDINATE_RESOURCE(P->NRCB.ELEMENT_ADDRESS);  /* PAGE B-15   */

  IF P ¬= NULL &
       P->NRCB.RESOURCE_CATEGORY = BF.LU THEN
    P = LOCATE_NODE_RESOURCE(P->NRCB.ASSOCIATED_RESOURCE);     /* PAGE B-14   */

  IF P ¬= NULL &
       P->NRCB.RESOURCE_CATEGORY = BF.PU THEN
    P = LOCATE_NODE_RESOURCE(P->NRCB.ASSOCIATED_RESOURCE);     /* PAGE B-14   */

  IF P ¬= NULL &
       P->NRCB.RESOURCE_CATEGORY = ALS THEN
    RETURN_PTR = P;

  ELSE
    RETURN_PTR = NULL;

  RETURN(RETURN_PTR);

END FIND_ALS_FOR_RESOURCE;
```

```
FIND_CP_ENTRY: PROCEDURE(RESOURCE_ADDR,CP_SESS_ID) RETURNS(BIT(1));

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  SEARCHES THE NRCB LIST TO FIND A RESOURCE ENTRY THAT CORRESPONDS TO
  │                 THE RESOURCE_ADDR AND CHECKS THE RESOURCE'S CP_INDIRECT LIST TO SEE
  │                 IF AN ENTRY EXISTS THAT CORRESPONDS TO THE CP_SESS_ID.
  │
  │      INPUT:     THE ELEMENT ADDRESS OF THE RESOURCE AND THE CP_PU HALF-SESSION
  │                 IDENTIFIER
  │
  │      OUTPUT:    NG IF NOT ON THE LIST, OR OK IF FOUND
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 ADD_CP_ENTRY                    PAGE B-2
  │                 DELETE_CP_ENTRY                 PAGE B-5
  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):
  │                 LOCATE_NODE_RESOURCE            PAGE B-14
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RESOURCE_ADDR BIT(16);
  DCL CP_SESS_ID PTR;
  DCL P PTR;
  DCL P1_PTR PTR;
  DCL TEMP_PTR PTR;

  P = LOCATE_NODE_RESOURCE(RESOURCE_ADDR);                  /* PAGE B-14   */

  FIND CPCB IN CPCB_LIST WHERE(CPCB.CP_SCB_ID = CP_SESS_ID);

  IF P ¬= NULL & CPCB_PTR ¬= NULL &
       P->NRCB.CP_INDIRECT_LIST ¬= NULL THEN
    SCAN P->NRCB.CP_INDIRECT_LIST PTR(CP_INDIRECT_PTR);
    . TEMP_PTR = CP_INDIRECT.CP_ENTRY_PTR;
    . IF CP_SESS_ID = TEMP_PTR->CPCB.CP_SCB_ID THEN
    .   RETURN(OK);
    SCANEND;

  RETURN(NG);

END FIND_CP_ENTRY;



FIND_DOMAIN_RESOURCE: PROCEDURE(RES_ADDR) RETURNS(POINTER);

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE SEARCHES THE DOMAIN RESOURCE LIST TO FIND THE ENTRY
  │                 CORRESPONDING TO THE ADDRESS PASSED IN RES_ADDR. IT RETURNS TO THE
  │                 INVOKING PROCEDURE THE POINTER TO THE DOMAIN RESOURCE.
  │
  │      INPUT:     THE NETWORK ADDRESS OF THE DOMAIN RESOURCE
  │
  │      OUTPUT:    THE POINTER TO THE DOMAIN RESOURCE CORRESPONDING TO THE ADDRESS
  │                 PASSED, IF AN ENTRY EXISTS; OTHERWISE, A NULL POINTER
  │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):
  │                 FIND_ALS_FOR_DOM_RES            PAGE B-8
  │                 FIND_LINK_FOR_DOM_RES           PAGE B-11
  │                 FIND_PU_FOR_DOM_RES             PAGE B-12
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RES_ADDR BIT(48);
  DCL P POINTER;
  DCL RETURN_PTR POINTER;

  RETURN_PTR = NULL;

  SCAN DRCB_LIST PTR(P) WHILE(RETURN_PTR = NULL);
  .
  . IF P->DRCB.NETWORK_ADDRESS = RES_ADDR THEN
  .   RETURN_PTR = P;
  .
  SCANEND;

  RETURN(RETURN_PTR);

END FIND_DOMAIN_RESOURCE;
```

```
FIND_ERCB: PROCEDURE(SUBAREA_ADDRESS,ER_NUMB) RETURNS(POINTER);

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE LOCATES THE ERCB THAT MATCHES THE INPUT PARAMETERS.
  │
  │      INPUT:     THE (SUBAREA ADDRESS,  ER NUMBER) PAIR THAT  IDENTIFIES THE SPECIFIC
  │                 ERCB
  │
  │      OUTPUT:    RETURNS A  POINTER TO THE ERCB  IF FOUND; OTHERWISE, RETURNS  A NULL
  │                 POINTER
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL SUBAREA_ADDRESS BIT(32);
  DCL ER_NUMB BIT(4);
  DCL P PTR;

  SCAN ERCB_LIST PTR(P);
  . IF P->ERCB.PARTNER_SA = SUBAREA_ADDRESS &
  .       P->ERCB.ER_NUM = ER_NUMB.THEN
  .   RETURN(P);
  SCANEND;

  RETURN(NULL);

END FIND_ERCB;



FIND_LINK_FOR_DOM_RES: PROCEDURE(RES_NA) RETURNS(POINTER);

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE  SEARCHES THE  DOMAIN RESOURCE  LIST TO  FIND A  LINK
  │                 CORRESPONDING TO  THE ADDRESS PASSED IN  RES_NA.  IT RETURNS  TO THE
  │                 INVOKING PROCEDURE THE POINTER TO THE LINK RESOURCE
  │
  │      INPUT:     THE NETWORK ADDRESS OF THE DOMAIN RESOURCE.
  │
  │      OUTPUT:    THE POINTER  TO THE LINK CORRESPONDING  TO THE ADDRESS PASSED  IF AN
  │                 ENTRY EXISTS; OTHERWISE, A NULL POINTER
  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):
  │                 FIND_DOMAIN_RESOURCE              PAGE B-10
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL P POINTER;
  DCL RES_NA BIT(48);
  DCL RETURN_PTR POINTER;

  RETURN_PTR = NULL;
  P = FIND_DOMAIN_RESOURCE(RES_NA);                    /* PAGE B-10      */

  IF P ¬= NULL &
       P->DRCB.RESOURCE_CATEGORY = BF.LU THEN
    P = P->DRCB.ASSOCIATED_RES_PTR;

  IF P ¬= NULL &
       P->DRCB.RESOURCE_CATEGORY = BF.PU THEN
    P = P->DRCB.ASSOCIATED_RES_PTR;

  IF P ¬= NULL &
       P->DRCB.RESOURCE_CATEGORY = ALS THEN
    P = P->DRCB.ASSOCIATED_RES_PTR;

  IF P ¬= NULL &
       P->DRCB.RESOURCE_CATEGORY = LINK THEN
    RETURN_PTR = P;

  RETURN(RETURN_PTR);

END FIND_LINK_FOR_DOM_RES;
```

```
FIND_LINK_FOR_RESOURCE: PROCEDURE(RES_EA) RETURNS(POINTER);

                                                                              /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  SEARCHES THE NRCB_LIST  TO FIND A LINK CORRESPONDING  TO RES_EA.  IF │
  │                 ONE IS  FOUND, IT RETURNS  THE POINTER TO  THE LINK RESOURCE  TO THE │
  │                 INVOKING PROCEDURE; OTHERWISE,  A NULL POINTER IS  RETURNED.  RES_EA │
  │                 MAY BE A BF.LU,  A BF.PU, AN ADJACENT LINK STATION,  OR A LINK.  THE │
  │                 PROCEDURE SEARCHES  THE HIERARCHY TO  FIND THE LINK  ASSOCIATED WITH │
  │                 THAT RESOURCE.                                           │
  │                                                                          │
  │      INPUT:     THE ELEMENT ADDRESS OF THE RESOURCE                      │
  │                                                                          │
  │      OUTPUT:    THE POINTER  TO THE LINK CORRESPONDING  TO THE ADDRESS PASSED  IF AN │
  │                 ENTRY EXISTS; OTHERWISE, A NULL POINTER                  │
  │                                                                          │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
  │                 DETERMINE_LCP_RESET_OPTION          PAGE B-7             │
  │                                                                          │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
  │                 LOCATE_NODE_RESOURCE                PAGE B-14            │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL P PTR;
  DCL RES_EA BIT(16);
  DCL RETURN_PTR PTR;

  P = LOCATE_NODE_RESOURCE(RES_EA);                          /* PAGE B-14      */

  IF P ¬= NULL &
       P->NRCB.RESOURCE_CATEGORY = BF.LU THEN
    P = LOCATE_NODE_RESOURCE(P->NRCB.ASSOCIATED_RESOURCE);   /* PAGE B-14      */

  IF P ¬= NULL &
       P->NRCB.RESOURCE_CATEGORY = BF.PU THEN
    P = LOCATE_NODE_RESOURCE(P->NRCB.ASSOCIATED_RESOURCE);   /* PAGE B-14      */

  IF P ¬= NULL &
       P->NRCB.RESOURCE_CATEGORY = ALS THEN
    P = LOCATE_NODE_RESOURCE(P->NRCB.ASSOCIATED_RESOURCE);   /* PAGE B-14      */

  IF P ¬= NULL &
       P->NRCB.RESOURCE_CATEGORY = LINK THEN
    RETURN_PTR = P;

  ELSE
    RETURN_PTR = NULL;

  RETURN(RETURN_PTR);

END FIND_LINK_FOR_RESOURCE;



FIND_PU_FOR_DOM_RES: PROCEDURE(RES_NA) RETURNS(POINTER);

                                                                              /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS  PROCEDURE SEARCHES  THE  DOMAIN RESOURCE  LIST  TO  FIND A  PU │
  │                 CORRESPONDING TO  THE ADDRESS PASSED IN  RES_NA.  IT RETURNS  TO THE │
  │                 CALLING PROCEDURE THE POINTER TO THE PU RESOURCE.        │
  │                                                                          │
  │      INPUT:     THE NETWORK ADDRESS OF THE DOMAIN RESOURCE               │
  │                                                                          │
  │      OUTPUT:    A  POINTER TO  THE SUBAREA  PU  ENTRY CORRESPONDING  TO THE  ADDRESS │
  │                 PASSED, IF AN ENTRY EXISTS; OTHERWISE, A NULL POINTER    │
  │                                                                          │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
  │                 FIND_DOMAIN_RESOURCE                PAGE B-10            │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL RES_NA BIT(48);
  DCL PU_SA BIT(32);
  DCL PU_EA BIT(16);
  DCL P POINTER;

  PU_SA = RES_NA(0:31);

  PU_EA = X'0000';

  P = FIND_DOMAIN_RESOURCE(PU_SA||PU_EA);

  RETURN(P);

END FIND_PU_FOR_DOM_RES;
```

```
FIND_SUBORDINATE_DOM_RES: PROCEDURE(RES_ADDR) RETURNS(POINTER);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE  SEARCHES THE DOMAIN RESOURCE  LIST TO FIND  AN ENTRY │
   │                 SUBORDINATE TO THE ADDRESS PASSED  IN RES_ADDR.  IT RETURNS DRCB_PTR │
   │                 TO THE INVOKING PROCEDURE.                           │
   │                                                                      │
   │      INPUT:     THE NETWORK ADDRESS OF THE DOMAIN RESOURCE            │
   │                                                                      │
   │      OUTPUT:    THE POINTER  TO THE FIRST DOMAIN  RESOURCE ENTRY SUBORDINATE  TO THE │
   │                 ADDRESS PASSED  IF AN  ENTRY EXISTS;  OTHERWISE, A  NULL POINTER  IS │
   │                 RETURNED                                             │
   │                                                                      │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
   │                 FIND_ALS_FOR_DOM_RES                 PAGE B-8        │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RES_ADDR BIT(48);
  DCL P POINTER;
  DCL P2 POINTER;
  DCL RETURN_PTR POINTER;

  RETURN_PTR = NULL;

  SCAN DRCB_LIST PTR(P) WHILE(RETURN_PTR = NULL);
  .
  . P2 = DRCB.ASSOCIATED_RES_PTR;
  .
  . IF P2->DRCB.NETWORK_ADDRESS = RES_ADDR THEN
  .
  .   RETURN_PTR = P;
  .
  SCANEND;

  RETURN(RETURN_PTR);

END FIND_SUBORDINATE_DOM_RES;



FIND_TGCB: PROCEDURE(DEST_SA,ERN) RETURNS(PTR);

                                                                        /*
   ┌──────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE  RETURNS A POINTER TO  THE TGCB FOR  THE TRANSMISSION │
   │                 GROUP OVER WHICH THE SPECIFIED EXPLICIT  ROUTE LEAVES THE NODE.  THE │
   │                 PROCEDURE FAILS TO FIND A TGCB  IF NO SUBAREA_ROUTING EXISTS FOR THE │
   │                 DESTINATION SUBAREA OR  IF NO TGCB EXISTS  FOR THE SUBAREA_ROUTING'S │
   │                 SPECIFIED TRANSMISSION GROUP TO THE NEXT SUBAREA.    │
   │                                                                      │
   │      INPUT:     DESTINATION SUBAREA AND ER NUMBER                    │
   │                                                                      │
   │      OUTPUT:    POINTER TO  THE TGCB FOR THE  FIRST TRANSMISSION GROUP OF  THE FIRST │
   │                 EXPLICIT ROUTE OR A NULL POINTER IF A TGCB CANNOT BE FOUND │
   └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL DEST_SA BIT(32);
  DCL ERN BIT(4);

  DCL TGCB_PTR PTR;

  FIND SUBAREA_ROUTING IN SUBAREA_ROUTING_LIST
       WHERE(SUBAREA_ROUTING.DEST_SA = DEST_SA);

  IF SUBAREA_ROUTING_PTR ¬= NULL THEN
    DO;
    . FIND TGCB IN TGCB_LIST WHERE
    .       (TGCB.TGN = SUBAREA_ROUTING.TGN(ERN) &
    .       TGCB.ADJ_SA = SUBAREA_ROUTING.ADJ_SA(ERN));
    . RETURN(TGCB_PTR);
    END;
  ELSE
    RETURN(NULL);

END FIND_TGCB;
```

```
FIND_TGCB_FOR_ALS_EA: PROCEDURE(RES_EA) RETURNS(PTR);
                                                                              /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  SEARCHES THE  TGCB LIST FOR A  TRANSMISSION GROUP THAT  CONTAINS THE │
  │                 ADJACENT LINK STATION SPECIFIED BY RES_EA.                │
  │                                                                          │
  │      INPUT:     THE ADJACENT LINK STATION ELEMENT ADDRESS                 │
  │                                                                          │
  │      OUTPUT:    TGCB_PTR IS SET TO NULL OR TO THE ADDRESS OF THE TGCB IF FOUND. │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL RES_EA BIT(16);
  DCL INDEX FIXED BINARY(15);
  DCL P PTR;
  DCL P1 PTR;

  SCAN TGCB_LIST PTR(P) WHILE(TGCB_PTR = NULL);
  .
  . SCAN P->TGCB.ASSOC_LSCB_LIST PTR(P1);
  . . IF P1->ASSOC_LSCB_ENTITY.LSCBPTR->LSCB.EA = RES_EA THEN
  . .   RETURN(P);
  . SCANEND;
  .
  SCANEND;

  RETURN(NULL);

END FIND_TGCB_FOR_ALS_EA;



LOCATE_NODE_RESOURCE: PROCEDURE(RESOURCE_ADDR) RETURNS(POINTER);
                                                                              /*
  ┌──────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  SEARCHES  THE  NRCB_LIST TO  FIND  THE  ENTRY CORRESPONDING  TO  THE │
  │                 ADDRESS PASSED  IN 'RESOURCE_ADDR'.  IT  RETURNS THE POINTER  TO THE │
  │                 RESOURCE TO THE INVOKING PROCEDURE.                      │
  │                                                                          │
  │      INPUT:     THE ELEMENT ADDRESS OF THE RESOURCE                      │
  │                                                                          │
  │      OUTPUT:    THE POINTER  TO THE NRCB CORRESPONDING  TO THE ADDRESS PASSED  IF AN │
  │                 ENTRY EXISTS; OTHERWISE, A NULL POINTER                  │
  │                                                                          │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
  │                 ADD_CP_ENTRY                    PAGE B-2                 │
  │                 DELETE_ALL_CP_ENTRIES           PAGE B-4                 │
  │                 DELETE_CP_ENTRY                 PAGE B-5                 │
  │                 DEQUEUE_RUS_FROM_RESOURCE       PAGE B-6                 │
  │                 DETERMINE_LCP_RESET_OPTION      PAGE B-7                 │
  │                 ENQUEUE_RU_FOR_RESOURCE         PAGE B-8                 │
  │                 FIND_ALS_FOR_RESOURCE           PAGE B-9                 │
  │                 FIND_CP_ENTRY                   PAGE B-10                │
  │                 FIND_LINK_FOR_RESOURCE          PAGE B-12                │
  │                 PURGE_RUS_FROM_RESOURCE         PAGE B-20                │
  │                 RESOURCE_TOTAL_SHARE_CNT        PAGE B-21                │
  └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL RESOURCE_ADDR BIT(16);
  DCL P PTR;

  SCAN NRCB_LIST PTR(P);
  .
  . IF P->NRCB.ELEMENT_ADDRESS = (RESOURCE_ADDR & NCB.NODE_ELEMENT_MASK) THEN
  .   RETURN(P);
  .
  SCANEND;

  RETURN(NULL);

END LOCATE_NODE_RESOURCE;
```

```
LOCATE_SUBORDINATE_RESOURCE: PROCEDURE(RESOURCE_ADDR) RETURNS(POINTER);

                                                                        /*
  ┌──────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  SEARCHES THE NRCB_LIST  TO FIND AN ENTRY SUBORDINATE  TO THE ADDRESS │
  │                 PASSED IN  RESOURCE_ADDR.  A SUBORDINATE  RESOURCE IS ONE  WHICH HAS │
  │                 ITS ASSOCIATED RESOURCE EQUAL TO THE RESOURCE_ADDR.  │
  │                                                                      │
  │      INPUT:     THE ELEMENT ADDRESS OF THE RESOURCE                  │
  │                                                                      │
  │      OUTPUT:    THE POINTER  TO THE  FIRST NRCB  ENTRY ASSOCIATED  WITH THE  ADDRESS │
  │                 PASSED IF AN ENTRY EXISTS; OTHERWISE, A NULL POINTER │
  │                                                                      │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                       │
  │                 FIND_ALS_FOR_RESOURCE              PAGE B-9          │
  └──────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL RESOURCE_ADDR BIT(16);
  DCL P PTR;

  SCAN NRCB_LIST PTR(P);
  .
  . IF P->NRCB.ASSOCIATED_RESOURCE = (RESOURCE_ADDR & NCB.NODE_ELEMENT_MASK) THEN
  .   RETURN(P);
  .
  SCANEND;

  RETURN(NULL);

END LOCATE_SUBORDINATE_RESOURCE;
```

```
MAP_FROM_CANONICAL: PROCEDURE(PIU_LENGTH);
                                                                              /*
 ┌──────────────────────────────────────────────────────────────────────────┐
 │      FUNCTION:  TO TRANSLATE AN OUTGOING PIU FROM CANONICAL FORMAT TO LINK FORMAT.
 │
 │      INPUT:     CANONICAL PIU, POINTED TO  BY MU_PTR.                  HE
 │                 LINK FORMAT BTU TO
 │
 │      OUTPUT:    PIU IN LINK FORMAT, POINTED TO BY  MU_PTR.  FOR FID2 AND FID3 PIU'S,
 │                 PIU_LENGTH IS SET EQUAL TO THE LENGTH OF THE PIU.
 │
 │      REFERS TO THE FOLLOWING PROCEDURE(S):
 │                 PTR_ADD                          PAGE B-20
 └──────────────────────────────────────────────────────────────────────────┘
                                                                              */
  DCL CANONICAL_PIU_PTR PTR;
  DCL PIU_LENGTH FIXED BIN;

  DCL RUN BASED(RUN_ADDR) CHAR(256);
  DCL RUN_SNC BIT(32) BASED(RUN_ADDR);

  DCL THN_ADDR PTR;
  DCL RHN_ADDR PTR;
  DCL RUN_ADDR PTR;

  CANONICAL_PIU_PTR = MU_PTR;
  CREATE MU;

  THN_ADDR = MU_PTR;
                                                                              /*
                  ┌──────────────────────────────────────────┐
                  │  SELECT FID TYPE AND MOVE TH FROM CANONICAL MU  │
                  └──────────────────────────────────────────┘
                                                                              */
  SELECT ANYORDER(CANONICAL_PIU_PTR->MU.FID);
  .
  . WHEN(FID0)
  .   DO;
  .   . RHN_ADDR = PTR_ADD(THN_ADDR,10);                  /* PAGE B-20       */
  .   . FID1_TH = CANONICAL_PIU_PTR->TH,BY NAME;
  .   END;
  .
  . WHEN(FID1)
  .   DO;
  .   . RHN_ADDR = PTR_ADD(THN_ADDR,10);                  /* PAGE B-20       */
  .   . FID1_TH = CANONICAL_PIU_PTR->TH,BY NAME;
  .   END;
  .
  . WHEN(FID2)
  .   DO;
  .   . RHN_ADDR = PTR_ADD(THN_ADDR,6);                   /* PAGE B-20       */
  .   . FID2_TH = CANONICAL_PIU_PTR->TH,BY NAME;
  .   . PIU_LENGTH = CANONICAL_PIU_PTR->MU.DCF + 6;
  .   END;
  .
  . WHEN(FID3)
  .   DO;
  .   . RHN_ADDR = PTR_ADD(THN_ADDR,2);                   /* PAGE B-20       */
  .   . FID3_TH = CANONICAL_PIU_PTR->TH,BY NAME;
  .   . PIU_LENGTH = CANONICAL_PIU_PTR->MU.DCF + 2;
  .   END;
  .
  . WHEN(FID4)
  .   DO;
  .   . RHN_ADDR = PTR_ADD(THN_ADDR,26);                  /* PAGE B-20       */
  .   . FID4_TH = CANONICAL_PIU_PTR->TH,BY NAME;
  .   END;
  .
  . WHEN(FIDF)
  .   DO;
  .   . FIDF_TH = CANONICAL_PIU_PTR->TH,BY NAME;
  .   . DISCARD CANONICAL_PIU_PTR->MU;
  .   . RETURN;
  .   END;
  .
  END;
```

```
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │        MOVE RH TO LINK FORMAT MU     │
                    └──────────────────────────────────────┘
                                                                        */
  RHN_ADDR->RH = CANONICAL_PIU_PTR->RH,BY NAME;
  RUN_ADDR = PTR_ADD(RHN_ADDR,3);                    /* PAGE B-20      */
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │ IF SENSE DATA IS INCLUDED, MOVE SNC FROM
                    │ CANONICAL MU.                        │
                    └──────────────────────────────────────┘
                                                                        */
  IF CANONICAL_PIU_PTR->SDI = SD THEN
    DO;
    . RUN_SNC = CANONICAL_PIU_PTR->SNC;
    . RUN_ADDR = PTR_ADD(RUN_ADDR,4);                /* PAGE B-20      */
    END;
                                                                        /*
                    ┌──────────────────────────────────────┐
                    │        MOVE RU FROM CANONICAL MU     │
                    └──────────────────────────────────────┘
                                                                        */
  IF CANONICAL_PIU_PTR->SDI = SD THEN
    RUN = CANONICAL_PIU_PTR->RU(0:(CANONICAL_PIU_PTR->MU.DCF) - 8); /* 8 ALLOWS FOR ZERO ORIGING */
  ELSE
    RUN = CANONICAL_PIU_PTR->RU(0:(CANONICAL_PIU_PTR->MU.DCF) - 4); /* 4 ALLOWS FOR ZERO ORIGING */

  DISCARD CANONICAL_PIU_PTR->MU;

  RETURN;
END MAP_FROM_CANONICAL;
```

```
MAP_TO_CANONICAL: PROCEDURE(NON_CAN_PTR,CANONICAL_PTR,PIULNTH);

                                                                        /*
┌──────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  TO  TRANSLATE AN  INCOMING MESSAGE  UNIT  (MU) FROM LINK FORMAT  TO │
│                 CANONICAL FORMAT.   THE CANONICAL FORMAT  MU CONTAINS ALL  TH FIELDS │
│                 THAT MAY APPEAR IN ANY FID TYPE, BUT ONLY THOSE FIELDS APPLICABLE TO │
│                 THE MU BEING PROCESSED ARE FILLED IN.                    │
│                                                                          │
│                 ONE EXCEPTION TO THIS IS THE DATA  COUNT FIELD (DCF).  DCF IS FILLED │
│                 IN FOR ALL FID TYPES EVEN THOUGH IT  DOES NOT EXIST IN THE LINK FORM │
│                 OF  FID2  AND FID3.   THIS  ALLOWS  REFERENCE  TO  THE DCF  IN  NODE │
│                 PROCEDURES WITHOUT CONCERN FOR THE FID TYPE.  THE DCF FIELD VALUE IS │
│                 DERIVED FROM THE BTU LENGTH PARAMETER (PIULNTH) FOR FID2 AND FID3.   │
│                                                                          │
│                 WHEN A SENSE CODE  IS PRESENT IN THE RU OF A LINK  FORMAT BTU, IT IS │
│                 MOVED INTO  THE SNC FIELD  IN THE  MUCB DURING THE  MAPPING PROCESS. │
│                 THE SENSE CODE IS STILL CONSIDERED A  PART OF THE RU FOR PURPOSES OF │
│                 THE LENGTH IN THE DCF.                                   │
│                                                                          │
│                 THIS ROUTINE  HANDLES THE  MESSAGE UNIT  WHEN AN  RH IS  NOT PRESENT │
│                 (¬BBIU).                                                 │
│                                                                          │
│      INPUT:     NON_CAN_PTR, CANONICAL_PTR, AND PIULNTH.   NON_CAN_PTR POINTS TO THE │
│                 LINK FORMAT PIU.  CANONICAL_PTR POINTS TO THE CANONICAL FORMAT MU TO │
│                 RECEIVE THE CONVERTED  PIU.  FOR FID2 AND FID3  PIULNTH CONTAINS THE │
│                 TOTAL LENGTH OF THE BTU.                                 │
│                                                                          │
│      OUTPUT:    MU IN CANONICAL FORMAT.                                  │
│                                                                          │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                               │
│                 PTR_ADD                          PAGE B-20               │
└──────────────────────────────────────────────────────────────────────────┘
                                                                        */
  DCL NON_CAN_PTR PTR;
  DCL CANONICAL_PTR PTR;
  DCL PIULNTH FIXED BIN;

  DCL RUN BASED(RUN_ADDR) CHAR(256);
  DCL RUN_SNC BIT(32) BASED(RUN_ADDR);

  DCL THN_ADDR PTR;
  DCL RHN_ADDR PTR;
  DCL RUN_ADDR PTR;

  THN_ADDR = NON_CAN_PTR;

  CANONICAL_PTR->MUCB.DIRECTION = RCV;                  /* SET MUCB DIRECTION     */

                                                                        /*
                  ┌──────────────────────────────────────────┐
                  │  SELECT FID TYPE AND MOVE TH TO CANONICAL MU  │
                  └──────────────────────────────────────────┘
                                                                        */
  SELECT ANYORDER(FID1_TH.FID);
  .
  . WHEN(FID0)
  .   DO;
  .   . RHN_ADDR = PTR_ADD(THN_ADDR,10);                /* PAGE B-20              */
  .   . CANONICAL_PTR->TH = FID1_TH,BY NAME;
  .   END;
  .
  . WHEN(FID1)
  .   DO;
  .   . RHN_ADDR = PTR_ADD(THN_ADDR,10);                /* PAGE B-20              */
  .   . CANONICAL_PTR->TH = FID1_TH,BY NAME;
  .   END;
  .
  . WHEN(FID2)
  .   DO;
  .   . RHN_ADDR = PTR_ADD(THN_ADDR,6);                 /* PAGE B-20              */
  .   . CANONICAL_PTR->TH = FID2_TH,BY NAME;
  .   . CANONICAL_PTR->MU.DCF = PIULNTH - 6;
  .   END;
  .
  . WHEN(FID3)
  .   DO;
  .   . RHN_ADDR = PTR_ADD(THN_ADDR,2);                 /* PAGE B-20              */
  .   . CANONICAL_PTR->TH = FID3_TH,BY NAME;
  .   . CANONICAL_PTR->MU.DCF = PIULNTH - 2;
  .   END;
  .
  . WHEN(FID4)
  .   DO;
  .   . RHN_ADDR = PTR_ADD(THN_ADDR,26);                /* PAGE B-20              */
  .   . CANONICAL_PTR->TH = FID4_TH,BY NAME;
  .   END;
  .
  . WHEN(FIDF)
  .   DO;
  .   . CANONICAL_PTR->TH = FIDF_TH,BY NAME;
  .   . RETURN;
  .   END;
  .
  END;
```

```
  IF FID1_TH.BBIUI = BBIU THEN            /* FIRST SEGMENT OR WHOLE BIU                    */
    DO;
    .
    . CANONICAL_PTR->RH = RHN_ADDR->RH,BY NAME;                 /* MOVE RH TO CANONICAL MU  */
    . RUN_ADDR = PTR_ADD(RHN_ADDR,3);                           /* PAGE B-20                */
    .
    . IF CANONICAL_PTR->SDI = SD THEN                           /* IF SNC PRESENT           */
    .   DO;
    .   . CANONICAL_PTR->SNC = RUN_SNC;                         /* MOVE SNC TO MUCB         */
    .   . RUN_ADDR = PTR_ADD(RUN_ADDR,4);                       /* PAGE B-20                */
    .   . CANONICAL_PTR->RU = RUN(0:(CANONICAL_PTR->MU.DCF) - 8);
    .   .                                                       /* 8 ALLOWS FOR ZERO ORIGIN */
    .   END;
    . ELSE
    .   CANONICAL_PTR->RU = RUN(0:(CANONICAL_PTR->MU.DCF) - 4);
    .                                                           /* 4 ALLOWS FOR ZERO ORIGIN */
    .
    END;
  ELSE
    DO;                                                         /* MIDDLE OR LAST SEGMENT   */
    . RUN_ADDR = RHN_ADDR;
    .
    . CANONICAL_PTR->RU = RUN(0:(CANONICAL_PTR->MU.DCF) - 1);
    END;                                    /* MOVE RU TO MU (ZERO ORIGIN)                  */

  RETURN;
END MAP_TO_CANONICAL;
```

```
MODULO: PROCEDURE(VALUE,MODULUS) RETURNS(FIXED BINARY(15));

                                                                                          /*
  ┌──────────────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE RETURNS  THE VALUE OF THE FIRST  PARAMETER MODULO THE │
  │                 SECOND.  THAT IS,  IT RETURNS THE REMAINDER PRODUCED  WHEN THE FIRST │
  │                 PARAMETER IS DIVIDED BY THE SECOND.                                  │
  │                                                                                      │
  │      INPUT:     A  NON-NEGATIVE NUMBER  THAT  IS  TO HAVE  A  MODULO  TAKEN AND  THE │
  │                 POSITIVE MODULUS THAT IS TO BE USED.                                 │
  │                                                                                      │
  │      OUTPUT:    THE MODULUS RESULT                                                   │
  └──────────────────────────────────────────────────────────────────────────────────────┘
                                                                                          */
  DCL VALUE FIXED(15) BIN;
  DCL MODULUS FIXED(15) BIN;

  RETURN(VALUE - ((VALUE / MODULUS) * MODULUS));
END MODULO;
```

```
NAU_SESSION_COUNT: PROCEDURE(NAU_EA) RETURNS(FIXED BINARY(15));

                                                                                          /*
  ┌──────────────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  COUNTS THE NUMBER OF SESSIONS WITH THE SPECIFIED NAU.                │
  │                                                                                      │
  │      INPUT:     THE ELEMENT ADDRESS OF THE SPECIFIED NAU                             │
  │                                                                                      │
  │      OUTPUT:    THE NUMBER OF NON-RESET SESSIONS WITH THE NAU                        │
  └──────────────────────────────────────────────────────────────────────────────────────┘
                                                                                          */
  DCL NAU_EA BIT(16);
  DCL P PTR;
  DCL RETURN_VALUE BIT(16);

  RETURN_VALUE = 0;

  SCAN SCB_LIST PTR(P);
  .
  . IF P->SCB.THIS_EA = NAU_EA &
  .      P->SCB.THIS_SA = NCB.NODE_SUBAREA_ADDRESS THEN
  .   RETURN_VALUE = RETURN_VALUE + 1;
  .
  SCANEND;

  RETURN(RETURN_VALUE);

END NAU_SESSION_COUNT;
```

```
PTR_ADD: PROCEDURE(OLDADDR,INCR) RETURNS(PTR);

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS  FUNCTION  PERFORMS THE  ADDITION  OF  AN  INTEGER VALUE  TO  A  │
  │                 POINTER.                                                     │
  │                                                                              │
  │      INPUT:     THE POINTER AND INTEGER VALUE TO BE ADDED                    │
  │                                                                              │
  │      OUTPUT:    THE FUNCTION RETURNS THE UPDATED POINTER VALUE.              │
  │                                                                              │
  │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                               │
  │                 MAP_FROM_CANONICAL              PAGE B-16                    │
  │                 MAP_TO_CANONICAL                PAGE B-18                    │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL OLDADDR PTR;
  DCL INCR FIXED BIN(31);

  DCL OLDADDR_ALIAS FIXED BIN(31) BASED(ADDR(OLDADDR));
  DCL NEWADDR PTR;
  DCL NEWADDR_ALIAS FIXED BIN(31) BASED(ADDR(NEWADDR));

  NEWADDR_ALIAS = OLDADDR_ALIAS + INCR;

  RETURN(NEWADDR);
END PTR_ADD;



PURGE_RUS_FROM_RESOURCE: PROCEDURE(RES_EA);

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  DEQUEUES AND DISCARDS  ALL RUS THAT HAVE  BEEN QUEUED FOR A  LINK OR  │
  │                 ADJACENT LINK STATION.  THESE ARE QUEUED  WHEN A REQUEST IS RECEIVED  │
  │                 PRIOR TO  THE LINK  OR ADJACENT LINK  STATION COMPLETING  ITS RESET.  │
  │                 THIS PROCEDURE IS CALLED  WHEN AN INOP OCCURS DURING THE  RESET OF A  │
  │                 LINK OR ADJACENT LINK STATION.                               │
  │                                                                              │
  │      INPUT:     THE ELEMENT ADDRESS OF THE RESOURCE                          │
  │                                                                              │
  │      OUTPUT:    RUS TO PU.SVC_MGR.NS.RCV;                                    │
  │                                                                              │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
  │                 LOCATE_NODE_RESOURCE            PAGE B-14                    │
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL P PTR;
  DCL RES_EA BIT(16);
  DCL MU_LIST PTR;

  P = LOCATE_NODE_RESOURCE(RES_EA);                    /* PAGE B-14            */

  DESTROY P->NRCB.SAVE_MU_FOR_RETRY_LIST;

  RETURN;

END PURGE_RUS_FROM_RESOURCE;
```

```
RESOURCE_TOTAL_SHARE_CNT: PROCEDURE(RES_EA) RETURNS(FIXED BINARY(15));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  DETERMINES THE NUMBER OF CONTROL POINTS  FOR WHICH THERE ARE ENTRIES
  │                 ON THE CPCB_LIST FOR THIS RESOURCE.
  │
  │      INPUT:     THE ELEMENT ADDRESS OF THE RESOURCE
  │
  │      OUTPUT:    A COUNT OF THE CONTROL POINTS THAT CONTROL THE RESOURCE
  │
  │      REFERS TO THE FOLLOWING PROCEDURE(S):
  │                 LOCATE_NODE_RESOURCE                PAGE B-14
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RES_EA BIT(16);
  DCL P PTR;
  DCL CP_CNT FIXED BINARY(15);

  P = LOCATE_NODE_RESOURCE(RES_EA);                          /* PAGE B-14       */

  CP_CNT = 0;

  IF P ¬= NULL THEN
    SCAN P->NRCB.CP_INDIRECT_LIST PTR(CP_INDIRECT_PTR);
    .
    . CP_CNT = CP_CNT + 1;
    .
    SCANEND;

  RETURN(CP_CNT);

END RESOURCE_TOTAL_SHARE_CNT;



RQD: PROCEDURE RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE DETERMINES  WHETHER THE MESSAGE IDENTIFIED  BY MU_PTR
  │                 REQUESTS THAT A DEFINITE RESPONSE BE MADE.
  │
  │      INPUT:     THE CURRENT MESSAGE UNIT.
  │
  │      OUTPUT:    RETURNS YES, IF AN RQD REQUEST; NO, IF NOT AN RQD REQUEST.
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);

  IF RRI = RQ &
       (DR1I = DR1 | DR2I = DR2) &
       ERI =  ¬ER THEN
    RC = YES;
  ELSE
    RC = NO;

  RETURN(RC);

END RQD;



RQE: PROCEDURE RETURNS(BIT(1));

                                                                                /*
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │      FUNCTION:  THIS PROCEDURE DETERMINES  WHETHER THE MESSAGE IDENTIFIED  BY MU_PTR
  │                 IS AN EXCEPTION REQUEST.
  │
  │      INPUT:     THE CURRENT MESSAGE UNIT.
  │
  │      OUTPUT:    RETURNS YES, IF AN RQE REQUEST; NO, IF NOT AN RQE REQUEST.
  └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);

  IF RRI = RQ &
       ERI = ER THEN
    RC = YES;
  ELSE
    RC = NO;

  RETURN(RC);

END RQE;
```

```
RQN: PROCEDURE RETURNS(BIT(1));

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS PROCEDURE DETERMINES  WHETHER THE MESSAGE IDENTIFIED  BY MU_PTR
   │                 REQUESTS THAT NO RESPONSE BE MADE.                           │
   │                                                                              │
   │      INPUT:     THE CURRENT MESSAGE UNIT.                                    │
   │                                                                              │
   │      OUTPUT:    RETURNS YES, IF AN RQN REQUEST; NO, IF NOT AN RQN REQUEST.   │
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RC BIT(1);

  IF RRI = RQ &
       DR1I =  ¬DR1 &
       DR2I =  ¬DR2 &
       ERI =  ¬ER THEN
    RC = YES;
  ELSE
    RC = NO;

  RETURN(RC);

END RQN;



UPM_CREATE_RQ: PROCEDURE(RQ_NAME) RETURNS(POINTER);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM  CREATES AN RU  IN THE FORM DEFINED  IN APPENDIX E  FOR THE
   │                 SPECIFIC REQUEST NAME.                                       │
   │                                                                              │
   │      INPUT:     THE REQUEST NAME.                                            │
   │                                                                              │
   │      OUTPUT:    THE POINTER TO THE NEW MESSAGE UNIT.                         │
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RQ_NAME CHAR(31) VARYING;
  DCL M_PTR POINTER;

  CREATE M_PTR->MU;
                                                  /* UNDEFINED PROCEDURE       */

  RETURN(M_PTR);
END UPM_CREATE_RQ;



UPM_CREATE_RSP: PROCEDURE(RSP_NAME) RETURNS(POINTER);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  CREATES A  RSP IN THE  FORM DEFINED IN  APPENDIX E FOR  THE SPECIFIC
   │                 REQUEST NAME.                                                │
   │                                                                              │
   │      INPUT:     THE RESPONSE NAME                                            │
   │                                                                              │
   │      OUTPUT:    THE POINTER TO THE NEW MESSAGE UNIT.                         │
   └──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL RSP_NAME CHAR(31) VARYING;
  DCL M_PTR POINTER;

  CREATE M_PTR->MU;
                                                  /* SEE FUNCTION              */

  RETURN(M_PTR);
END UPM_CREATE_RSP;
```

```
UPM_LOG: PROCEDURE(MESSAGE);

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS UPM MAKES AN ENTRY IN A LOG.                        │
   │                                                                          │
   │      INPUT:     THE INVALID REQUEST OR RESPONSE.                         │
   │                                                                          │
   │      OUTPUT:    AN ENTRY IS MADE IN THE LOG.                             │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL MESSAGE CHAR(*) VARYING;

                                                  /* UNDEFINED PROCEDURE      */

  RETURN;
END UPM_LOG;
```

This page intentionally left blank

SNA is an architecture defined by a meta-implementation. The node meta-implementation uses data structures and specific algorithms that are not part of the architecture. The function performed is part of the architecture, but the particular method used is not. One of the important aspects of the meta-implementation is the execution model. The execution model defines the data structures and procedures needed for the meta-implementation to execute by describing how, when, and in what environment procedures are invoked in an SNA meta-implementation node.

NODE META-IMPLEMENTATION

The SNA node meta-implementation consists of the FAPL procedures and multiple processes connected by shared storage. A process refers to an independent processing unit (and its storage), running asynchronously to all other processes in the node. Figure C-1 shows the processes of a node. There is one higher-level process and multiple data link control (DLC) level processes. The number of DLC-level processes may differ from node to node, but is fixed at system-definition time for each node; one DLC-level process exists for each link attached to the node, whether or not the link is active. Subarea nodes may have any number of DLC-level processes; peripheral nodes have only one DLC-level process.

The processes communicate with one another by means of shared storage. Information in shared storage data structures is placed there by one process and accessed by another. There are two types of data structures in shared storage: control blocks and lists. These data structures contain all information used by multiple processes.

When entities or lists in shared storage are simply read by different processes, any number of processes may read simultaneously. There are, however, lists and entities over which a process requires exclusive control in order to update contained information or to access data liable to be updated. To control access to such objects, the FAPL statements LOCK (of a list) and UNLOCK are used. A procedure executing a LOCK statement is delayed until it is granted use of the requested resource. The LOCK and UNLOCK statements are used for both reading and writing when exclusive control is needed.

In order to avoid deadlock possibilities and to simplify the locking mechanism, only one list is locked in a LOCK statement, and LOCK statements are not nested.

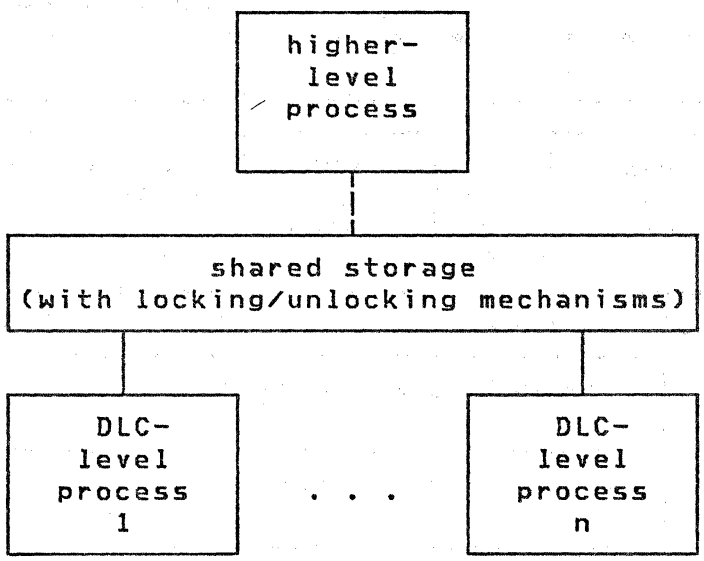


Figure C-1. Processes in an SNA node

Figure C-2 shows the relationship of SNA layers of a subarea node to processes. Only transmission group control (TGC) spans processes. Each DLC element is contained in its own process and all other layers are in the higher-level process. For a peripheral node, there is a single DLC-level process and path control (PC) spans the two processes.

There are three types of communication that require objects to be in shared storage: DLC element to PC, DLC element to PU.SVC_MGR.DLC_MGR, and an end user or node operator to the SNA node. Only the first of these is discussed in this book.

For a subarea node, the objects that are in shared storage are those required for communication between TGC and the DLC elements. Figure C-3 shows the FAPL objects in shared storage:

- The transmission group control blocks, TGCBs
- The queue used to transmit PIUs from DLC to TGC, Q_BTU_RCV
- The list used to transmit PIUs from TGC to DLC, PRTY_SEND_PIU_LIST
- The link station control block, LSCB

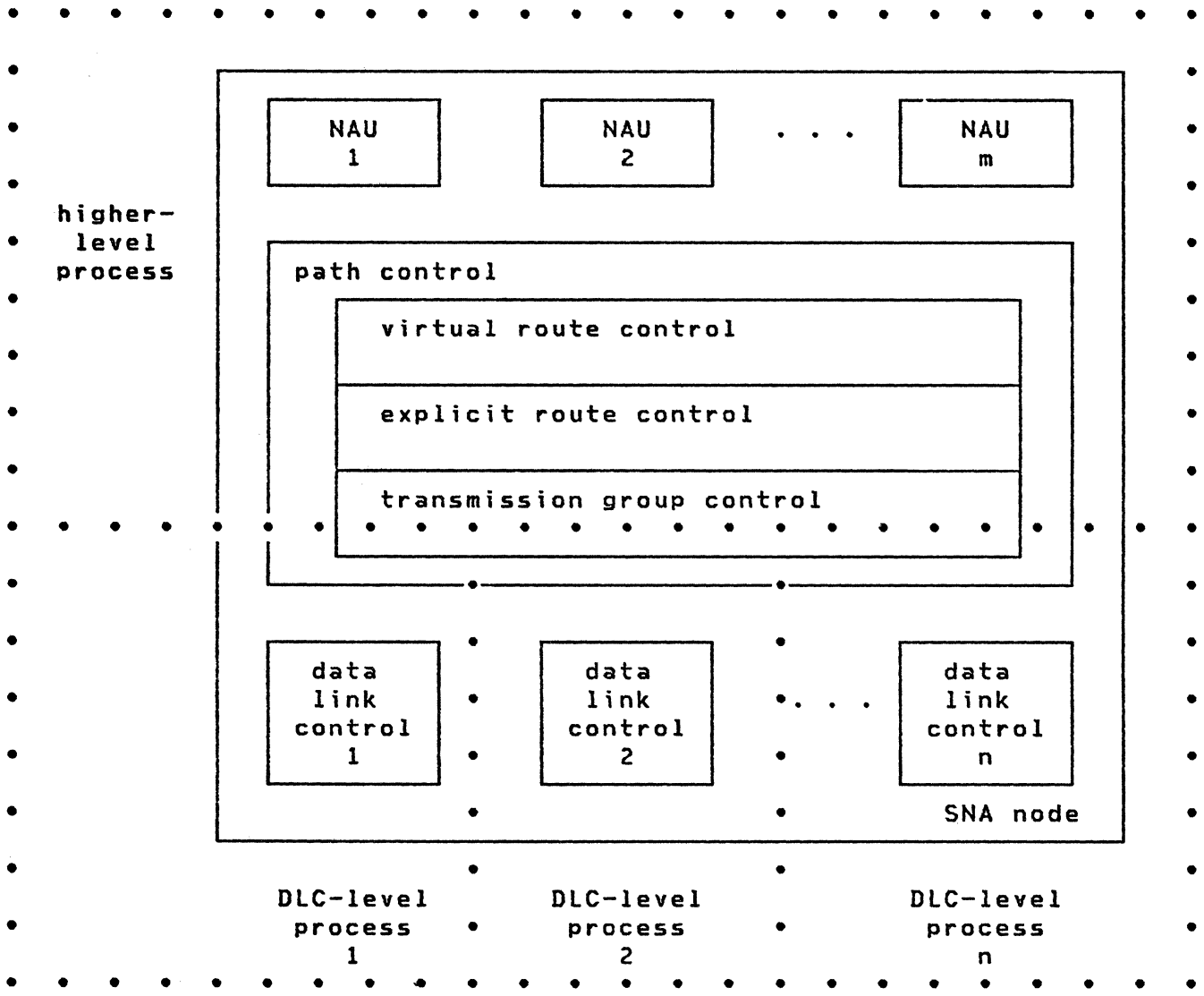


Figure C-2. Mapping of Processes onto SNA Subarea Node Layers

These two lists and the fields of the two control blocks contain the information that is shared.

For a peripheral node, inter-process communication involves the lists used to pass PIUs between the two processes and some information about the characteristics of the link station. This information is contained in the path control control block (PCCB) and the LSCB, both of which are located in shared storage.

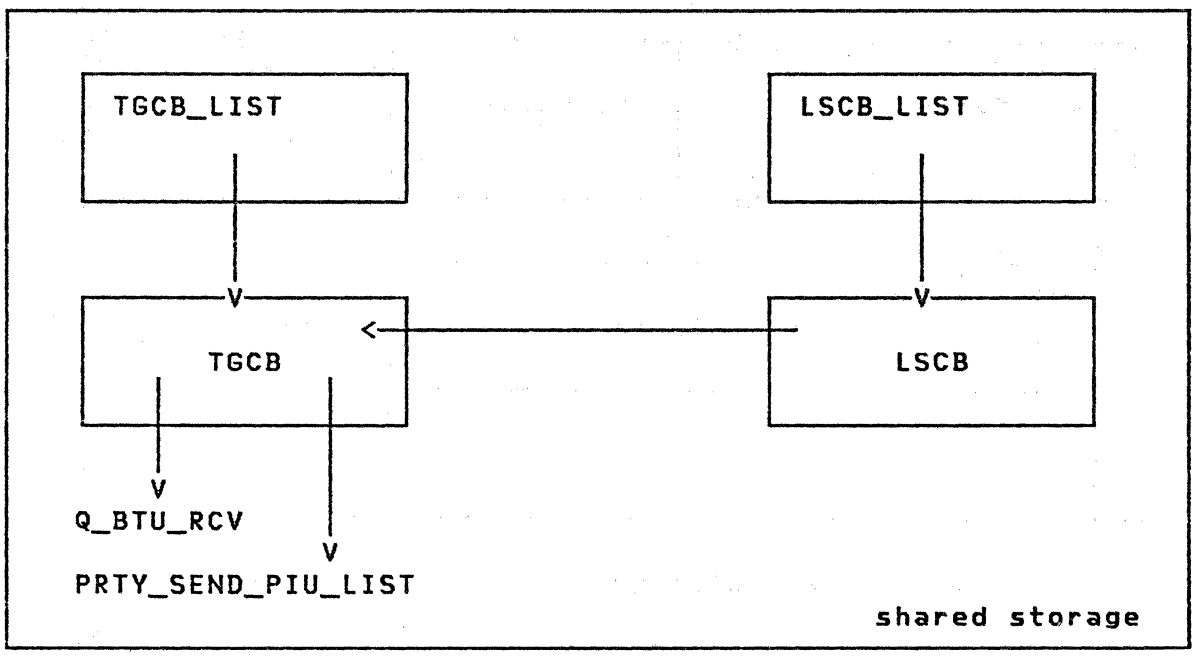


Figure C-3. Contents of Shared Storage for a Subarea Node

PARTS OF A PROCESS

This section describes the structure of a process and the specific details of a subarea node higher-level process. The peripheral node higher-level process is similar; only its differences are described. Because DLC is not described in this book, the DLC-level process is not discussed.

A process consists of a scheduler, a dispatcher, the dispatched procedures, and local storage for, and shared storage used by, the procedures invoked by that dispatcher. Figure C-4 shows the relations between these components and shared storage.

The scheduler is the root procedure in a process. The scheduler's primary function is to select the scheduled data queues that are used by its associated procedures and initiate the reading. A scheduled data queue is a list declared with the QUEUE option on the NEWLIST statement. This option identifies the list as one that is known by the scheduler; the scheduler schedules the dispatching of the procedures that dequeue from these lists.

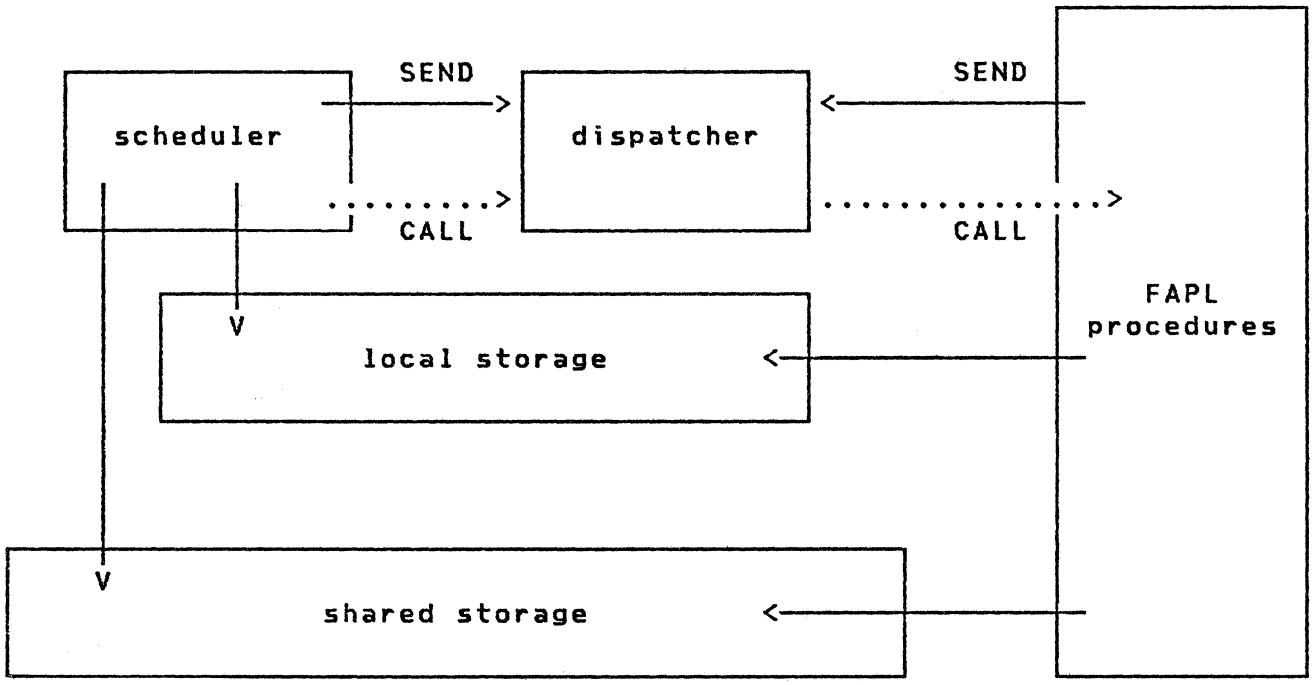


Figure C-4. Process Components and Interactions

The term queue is used throughout the book to refer to a scheduled data queue. This appendix uses the longer name to distinguish these queues from the dispatching queues (see below) and to emphasize their role in the execution model. A scheduled data queue is used to carry information between components of the meta-implementation; a dispatching queue is part of the execution model used to control the execution order of subthreads (see below). A scheduled data queue may contain any type of entity; a dispatching queue always contains dispatching queue entries (DQEs, page C-18). A scheduled data queue is always anchored in a control block.

The scheduler services each scheduled data queue in an order that can vary according to implementation options. When it selects a scheduled data queue to service, it sends an OPEN_QUEUE signal to the dispatcher, and then calls the dispatcher. The scheduled data queue that is selected may be located in either local or shared storage.

The dispatcher services all SENDs in the process, both those from the scheduler and those from subsequently invoked FAPL procedures. The dispatcher processes the single SEND from the scheduler and any number of SENDs from the procedures before returning to the scheduler. When a SEND occurs, the information is placed on the dispatching queue. The

dispatching queue is a FIFO list that is not serviced by the scheduler. When the dispatcher gains control, it removes the next item from its dispatching queue and calls the procedure that is specified in the DQE. The dispatcher also maintains a set of variables comprising the current environment and accessible to dispatched procedures; the FAPL procedures assume that all these variables are properly established when they are invoked.

The processing that occurs from the time that the scheduler calls the dispatcher to the time that the dispatcher returns control to the scheduler is referred to as a thread. The processing from the time that the dispatcher calls a FAPL procedure to the time it regains control is a subthread.

THE SCHEDULER

The scheduler requires knowledge of the data structures in order to locate the scheduled data queues and scheduler-initiated procedures to be serviced, and to pass the dispatcher the information needed to establish the current environment. The current environment needs to be established because single procedures work with multiple control blocks, and addressability to the proper control blocks must be provided.

As the scheduler searches through data structures for the scheduled data queues and scheduler-initiated procedures, it sets the pointer to the last control block. When a thread is begun, the only control block pointer that is established is the one to the control block that contains the scheduled data queue being serviced or that triggers the scheduler-initiated procedure being invoked. Figure C-5 shows the data structures and scheduled data queues of interest to the scheduler of a subarea node higher-level process. For a peripheral node, the scheduler of the higher-level process handles session control blocks (SCBs) for half-sessions and the PCCB.

The scheduler traverses the data structures selecting the next scheduled data queue or procedures to service and establishing the pointer that is relevant. The scheduler of a subarea node higher-level process might establish SCB_PTR, TCCB_PTR, VRCB_PTR, TGCB_PTR, or PCCB_PTR. The node control block (NCB) is always available to the FAPL procedures. Procedure HIGHER_LEVEL_SCHEDULER (page C-14) shows the meta-implementation scheduler for a subarea node higher-level process. The servicing of a scheduled data queue consists of verifying that the scheduled data queue is not empty, sending an OPEN_QUEUE signal to the associated dequeuing procedure, and calling the dispatcher. The servicing of a scheduler-initiated procedure consists of sending a signal to the procedure and calling the dispatcher.

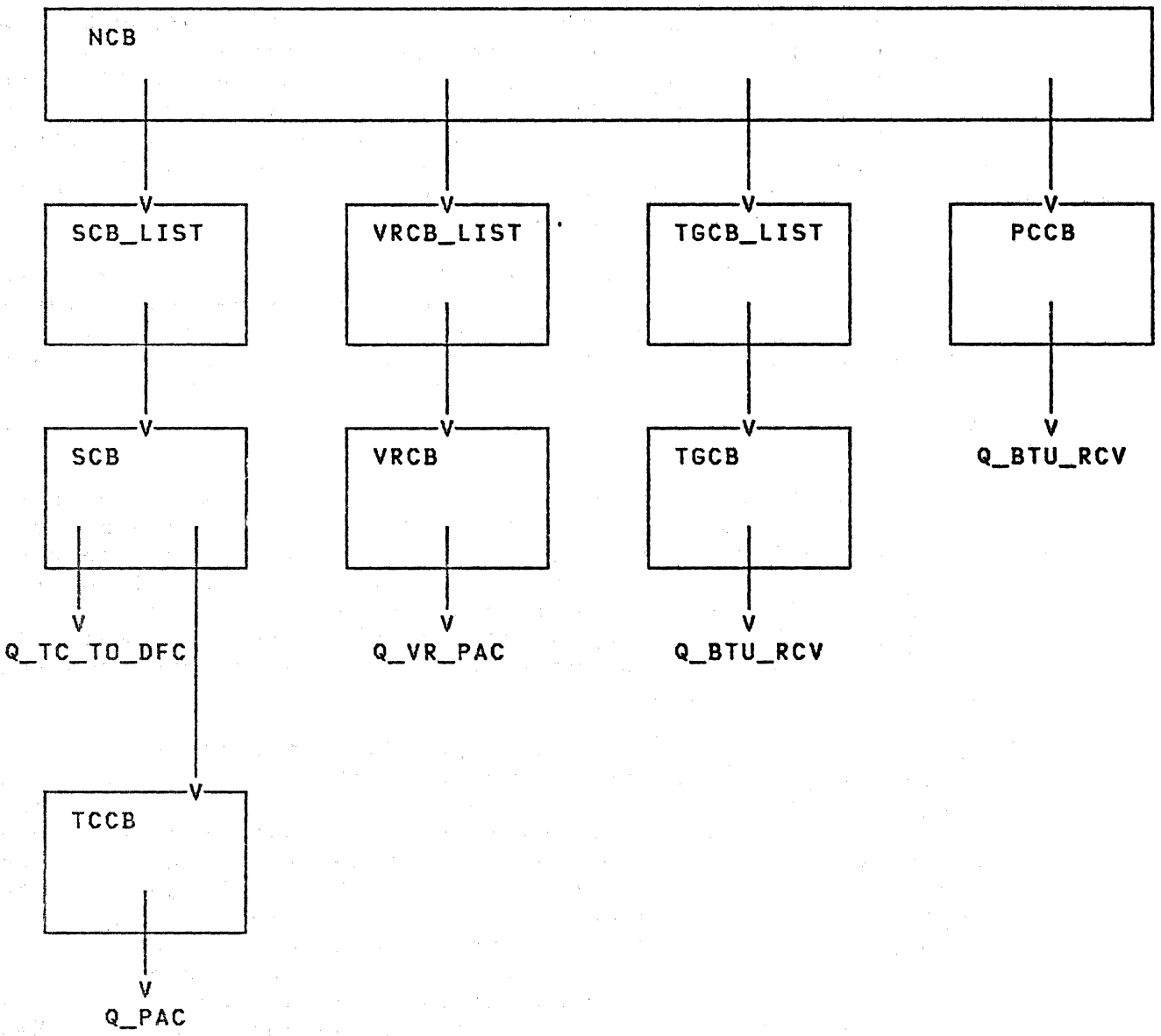


Figure C-5. Node Data Structures Used by the Scheduler of a Subarea Node Higher-Level Process

A dequeuing procedure is a FAPL procedure having the function to check that it is in a state to receive messages. If it is in a valid state, the procedure dequeues an entity (using REMOVE), and processes it. In this book, these procedures are named DEQUEUE.queue-name (or a variation on that form).

There are two types of scheduled data queues that a scheduler services: ones that pass message units between architected components and ones that pass message units into or out of the node. Only the first type is discussed in this book.

The scheduled data queues serviced by a subarea node higher-level process are Q_PAC, Q_TC_TO_DFC, Q_VR_PAC, and Q_BTU_RCV. For each of these scheduled data queues, an architected procedure inserts an entity on the scheduled data queue, and an architected dequeuing procedure removes it. These scheduled data queues contain one of two types of entities: message units (MUs, page C-16) or basic transmission units (BTUs, page C-18).

There are three scheduler-initiated procedures in a subarea node higher-level process: TC_OR_BF_TC.IPR_SEND, PC_SA.VRC.VRPRS_SEND, and PU.SVC_MGR.PC_ROUTE_MGR.RCV. TC_OR_BF_TC.IPR_SEND is used to initiate isolated pacing responses for a half-session or boundary function to its pacing partner (boundary function or half-session). PC_SA.VRC.VRPRS_SEND is used to initiate virtual route pacing responses. PU.SVC_MGR.PC_ROUTE_MGR.RCV is used to initiate an NC_DACTVR(Forced). In all these cases, the decision to send the request or response is implementation-dependent. In the meta-implementation, a UPM is called within the sent-to procedure that makes the implementation-dependent part of the decision about sending the message, and the architected procedure determines if it is architecturally valid to send the message.

If a scheduler is in control, and all its scheduled data queues are empty, a new thread is started by one of the scheduler-initiated procedures or by the appearance of data on a scheduled data queue. This data is placed there by a different process or by something outside the SNA node (e.g., end users, node operators).

THE DISPATCHER

The dispatcher has the responsibility of processing SENDs. Unlike CALL, SEND does not imply return of control to the issuing procedure when the invoked procedure completes. A SEND implies only that execution will occur at some later unspecified time, not immediately, as with a CALL.

One of the differences between using SEND to pass a message unit and enqueuing it (via INSERT) on a scheduled data queue is in the invocation of the receiving procedure. SENDs are processed in the order in which they are executed; message units placed on different scheduled data queues are processed in the order in which the scheduled data queues are serviced, though they are handled FIFO within a scheduled data queue.

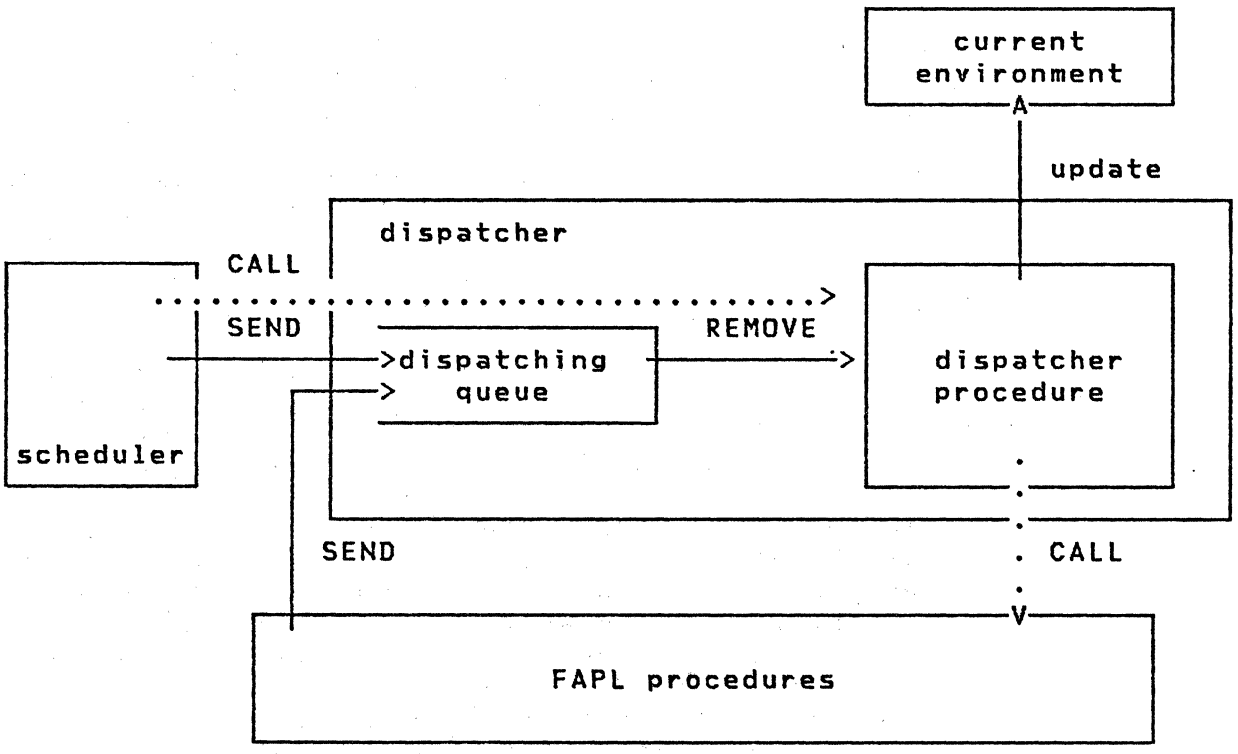


Figure C-6. Details of a Dispatcher

Whenever a SEND is issued, a DQE is added to the dispatching queue. When the dispatcher gains control, it removes the next entry from the dispatching queue, establishes the current environment, and calls the procedure that the SEND specified. Figure C-6 shows the structure of the dispatcher and its interactions with the scheduler and FAPL procedures.

The DQE contains the information used to establish the current environment for the procedure that the SEND specified.

For the dispatcher of a higher-level process, the current environment consists of two sets of variables, one kept in the environment vector (EV) and one kept in the node control block (NCB). (The environment vector is a structure containing meta-implementation variables.) The variables included in the current environment are:

- The destination procedure
- The sending procedure
- The input signal that was sent, if any
- The parameter pointer (PARM_PTR) that was sent, if any
- The canonical message unit (MU) that was sent, if any
- The pointers to all current control blocks, other than the NCB, as defined in the CONTROL_BLOCK_DEFINITION statement

The control block pointers are the part of the current environment kept in the NCB. The set of control blocks used in a subarea node includes the LSCB, TGCB, ERCB, VRCB, and SCB; in a peripheral node, the set includes the LSCB, PCCB and SCB. Some of the control block pointers may not have meaningful values, because they are not applicable to the current processing. (For example, a half-session does not have any knowledge of link stations, and so it has no LSCB.) The invoked procedure uses only the pointers that are properly established.

The first five variables listed above are kept in the EV. The canonical message unit requires a few additional remarks; the first four fields are described in the context of the SEND statement.

The canonical message unit is a meta-implementation device to allow the code of the meta-implementation to be FID-type independent, and to allow all layers to use a generic structure. Depending on what fields are filled in, the MU may represent a PIU, a BIU, or an RU. The valid fields are determined by the procedure using the MU. (A BTU is represented by a different entity and an explicit conversion is made from BTU to PIU.)

Procedure MAP_TO_CANONICAL (Appendix B) converts the link-form PIU into the canonical form as soon as it is received in PC. All processing then assumes the canonical form. Similarly, MAP_FROM_CANONICAL (Appendix B) converts the canonical MU into a link-form PIU just before it is passed to a DLC element.

The message unit control block (MUCB) is a portion of the MU used to keep information that is related to the MU, but not part of it. It is used primarily for passing parameters about the MU between layers. For example, the Lost Data indicator (LDI) is a parameter passed from DLC to PC, indicating whether or not any data received on a link was lost because of truncation.

The SEND statement creates a DQE, fills in the appropriate values, and adds an entity to the dispatching queue. The destination procedure is specified in the SEND statement itself, as are the MU, the input signal, and PARM_PTR, if they are applicable. The sending procedure is the current destination procedure, i.e., the field specifies the procedure that started the current subthread. For the control block pointers, the current values are copied, unless an override is explicitly given in the USING clause.

There are two places that SENDs occur: the scheduler and dispatched FAPL procedures. For a SEND in the scheduler of a higher-level process, the destination procedure is the dequeuing procedure, the sending procedure is the scheduler, the input signal is OPEN_QUEUE, and no value for PARM_PTR is specified. A message unit is not sent; if a thread is using an existing message unit, the message unit is on the scheduled data queue and the dequeuing procedure establishes the pointer. The scheduler establishes only the pointer to the control block that contains the scheduled data queue being serviced.

For SENDs occurring in dispatched FAPL procedures, the destination procedure is the procedure specified in the SEND, the sending procedure is the current destination procedure, and the input signal and PARM_PTR are specified in the SEND statement itself. The SEND statement causes the current value of the pointers to all control blocks other than the NCB, unless they are overridden by a USING clause, to be copied in the DQE. The procedures that communicate via SENDs are responsible for validly establishing the appropriate pointers. For FAPL procedures in the higher-level process, the value of the pointer to the specified MU is also copied; if a signal is being sent, a null pointer is placed in the DQE.

When the dispatcher gains control and removes a DQE, it copies the values of the entry into the variables of the current environment, and calls the specified procedure. A dispatcher procedure for a subarea node higher-level process is shown on page C-13.

A procedure called by the dispatcher of a higher-level process has different levels of access to the different fields of the current environment. The procedure can use or change any of the pointers to the MU or control blocks by

referring to the appropriate pointers. (All these entities are defined to use the current environment pointers as their default pointers, so an unqualified reference to any of these entities refers to the one defined by the current environment.) As for the other variables, the destination procedure is not accessible, the sending procedure and input signal can be checked by the built-in functions, DISPATCHED_BY and INPUT, respectively, and the parameter can be accessed by referring to PARM_PTR. While values may be assigned to PARM_PTR, the value is passed on by a SEND only if PARM_PTR is included in the USING clause.

```
HIGHER_LEVEL_DISPATCHER: PROCEDURE;

                                                                                /*
   ┌──────────────────────────────────────────────────────────────────────────┐
   │      FUNCTION:  THIS IS A  DISPATCHER FOR A SUBAREA NODE  HIGHER-LEVEL PROCESS.  THE │
   │                 DISPATCHER DEQUEUES THE  FIRST ENTRY FROM THE  DISPATCHING QUEUE (IF │
   │                 IT IS  NOT EMPTY), SETS  UP THE  CURRENT ENVIRONMENT, AND  CALLS THE │
   │                 NAMED  PROCEDURE.  IF  THE DISPATCHING  QUEUE IS  EMPTY, CONTROL  IS │
   │                 RETURNED TO THE SCHEDULER.                               │
   │                                                                          │
   │      INPUT:     NO INPUT AT TIME OF CALL.  INPUT TO THE DISPATCHER IS IN THE FORM OF │
   │                 ENTITIES ADDED TO THE DISPATCHING QUEUE BY SEND STATEMENTS.  │
   │                                                                          │
   │      OUTPUT:    A CALL TO THE NEXT SENT-TO PROCEDURE OR A RETURN TO THE SCHEDULER │
   │                                                                          │
   │      NOTES:     1. THE NECESSARY DECLARATION AND INITIALIZATION  LOGIC FOR THE ENTRY │
   │                    VARIABLE ARRAY,  PROCNAME, TO INCLUDE ALL  DESTINATION PROCEDURES │
   │                    REPLACES THE "DCL PROCNAME;" STATEMENT.               │
   │                                                                          │
   │                 2. DISPQ, THE DISPATCHING QUEUE, IS DEFINED BY THE FOLLOWING NEWLIST │
   │                    STATEMENT:                                            │
   │                                                                          │
   │                                  NEWLIST DISPQ ENTRY_NAME(DQE) FIFO;     │
   │                                                                          │
   │                    DQE IS DECLARED ON PAGE C-18.                         │
   │                                                                          │
   │      REFERENCED BY THE FOLLOWING PROCEDURE(S):                           │
   │                 HIGHER_LEVEL_SCHEDULER              PAGE C-14            │
   └──────────────────────────────────────────────────────────────────────────┘
                                                                                */
  DCL PROCNAME;                          /* SEE NOTE 1                         */

  DO WHILE(¬EMPTY(DISPQ));               /* DISPATCHER CONTINUES TO CALL PROCEDURES */
  .                                      /* UNTIL ITS QUEUE IS EMPTY.  SEE NOTE 2. */
  .
  . REMOVE DQE FROM DISPQ;
  .
                                                                                /*
                  ┌──────────────────────────────────────────┐
                  │  THE   FOLLOWING  STATEMENTS   SET  DATA   AND  │
                  │  POINTERS  INTO   THE  EV  FROM  THE   DQE  TO  │
                  │  INITIALIZE  THE CURRENT  ENVIRONMENT FOR  THE  │
                  │  PROCEDURE  TO   BE  CALLED.    THE  DQE   WAS  │
                  │  GENERATED BY EXECUTION OF A SEND STATEMENT.    │
                  └──────────────────────────────────────────┘
                                                                                */
  .
  . EV.DEST_PROC     = DQE.DEST_PROC;    /* DESTINATION PROCEDURE NAME         */
  . EV.SEND_PROC     = DQE.SEND_PROC;    /* SENDING PROCEDURE NAME             */
  . EV.INPUT_SIGNAL  = DQE.INPUT_SIGNAL; /* INPUT SIGNAL                       */
  . NCB.LSCB_PTR     = DQE.LSCBPTR;      /* LINK STATION CONTROL BLOCK         */
  . NCB.SCB_PTR      = DQE.SCBPTR;       /* SESSION CONTROL BLOCK              */
  . NCB.TGCB_PTR     = DQE.TGCBPTR;      /* TRANSMISSION GROUP CONTROL BLOCK   */
  . NCB.ERCB_PTR     = DQE.ERCBPTR;      /* EXPLICIT ROUTE CONTROL BLOCK       */
  . NCB.VRCB_PTR     = DQE.VRCBPTR;      /* VIRTUAL ROUTE CONTROL BLOCK        */
  . NCB.NRCB_PTR     = DQE.NRCBPTR;      /* NODE RESOURCE CONTROL BLOCK        */
  . NCB.DRCB_PTR     = DQE.DRCBPTR;      /* DOMAIN RESOURCE CONTROL BLOCK      */
  . NCB.TCCB_PTR     = DQE.TCCBPTR;      /* TRANSMISSION CONTROL CONTROL BLOCK */
  . EV.PARM_PTR      = DQE.PARMPTR;      /* PARAMETER LIST POINTER             */
  . EV.MU_PTR        = DQE.MUPTR;        /* CANONICAL MESSAGE UNIT POINTER     */
  .
  . DISCARD DQE;
  .
  . CALL PROCNAME(EV.DEST_PROC);
  END;

  RETURN;

END HIGHER_LEVEL_DISPATCHER;
```

```
HIGHER_LEVEL_SCHEDULER: PROCEDURE;

                                                                                /*
┌──────────────────────────────────────────────────────────────────────────────┐
│      FUNCTION:  THIS PROCEDURE SERVICES ALL SCHEDULED  DATA QUEUES OF A HIGHER-LEVEL │
│                 PROCESS  IN A  SUBAREA NODE.   IF THE  SCHEDULED DATA  QUEUE IS  NOT │
│                 EMPTY, THE  PROCEDURE SENDS  A SIGNAL  TO THE  APPROPRIATE DEQUEUING │
│                 PROCEDURE  AND CALLS  THE  DISPATCHER.  THIS  IS  ONLY  ONE OF  MANY │
│                 POSSIBLE IMPLEMENTATIONS OF A HIGHER-LEVEL SCHEDULER.        │
│                                                                              │
│      INPUT:     NONE                                                         │
│                                                                              │
│      OUTPUT:    SENDS  AN  OPEN_QUEUE  SIGNAL  TO THE  DEQUEUING  PROCEDURE  OF  THE │
│                 SELECTED SCHEDULED DATA QUEUE.  EACH SCAN ESTABLISHES THE POINTER TO │
│                 THE CURRENT CONTROL BLOCK BEING USED.                        │
│                                                                              │
│      REFERS TO THE FOLLOWING PROCEDURE(S):                                   │
│                 HIGHER_LEVEL_DISPATCHER          PAGE C-13                   │
└──────────────────────────────────────────────────────────────────────────────┘
                                                                                */
                                                                                /*
                ┌──────────────────────────────────────────┐
                │ ESTABLISH THE  SCHEDULER AS THE START  OF THE │
                │ CURRENT THREAD                           │
                └──────────────────────────────────────────┘
                                                                                */
DQE.SEND_PROC = SCHEDULER_INDEX;                        /* VALUE SET BY PROCESSOR */

DO WHILE(B'1');                                         /* ONCE GIVEN CONTROL,    */
.                                                       /* SCHEDULER CONTINUES    */
.                                                       /* RUNNING INDEFINITELY   */
                                                                                /*
                ┌──────────────────────────────────────────┐
                │ SERVICE PCCB SCHEDULED DATA QUEUE        │
                └──────────────────────────────────────────┘
                                                                                */
. IF ¬EMPTY(PCCB.Q_BTU_RCV) THEN
.   DO;
.   . SEND 'OPEN_QUEUE' TO PC.DEQ_Q_BTU_RCV;            /* CHAPTER 3              */
.   . CALL HIGHER_LEVEL_DISPATCHER;                     /* PAGE C-13              */
.   END;
                                                                                /*
                ┌──────────────────────────────────────────┐
                │ SERVICE TGCB SCHEDULED DATA QUEUES       │
                └──────────────────────────────────────────┘
                                                                                */
. SCAN TGCB_LIST PTR(TGCB_PTR);
. . IF ¬EMPTY(TGCB.Q_BTU_RCV) THEN
. .   DO;
. .   . SEND 'OPEN_QUEUE' TO PC_SA.TGC.DEQ_Q_BTU_RCV;   /* CHAPTER 3              */
. .   . CALL HIGHER_LEVEL_DISPATCHER;                   /* PAGE C-13              */
. .   END;
. SCANEND;
                                                                                /*
                ┌──────────────────────────────────────────┐
                │ SERVICE  VRCB   SCHEDULED  DATA   QUEUES  AND │
                │ SCHEDULER-INITIATED PROCEDURES           │
                └──────────────────────────────────────────┘
                                                                                */
. SCAN VRCB_LIST PTR(VRCB_PTR);
. . IF ¬EMPTY(VRCB.Q_VR_PAC) THEN
. .   DO;
. .   . SEND 'OPEN_QUEUE' TO PC_SA.VRC.DEQ.Q_VR_PAC;    /* CHAPTER 3              */
. .   . CALL HIGHER_LEVEL_DISPATCHER;                   /* PAGE C-13              */
. .   END;
. .
. . SEND 'SEND_VRPRS' TO PC_SA.VRC.VRPRS_SEND;          /* CHAPTER 3              */
. . CALL HIGHER_LEVEL_DISPATCHER;                       /* PAGE C-13              */
. .
. . SEND 'SEND_DACTVR_F' TO PU.SVC_MGR.PC_ROUTE_MGR.RCV; /* CHAPTER 12            */
. . CALL HIGHER_LEVEL_DISPATCHER;                       /* PAGE C-13              */
. .
. SCANEND;
```

```
                                                                        /*
                    ┌──────────────────────────────────────────┐
                    │ SERVICE  SCB  SCHEDULED  DATA  QUEUES  AND │
                    │ SCHEDULER-INITIATED PROCEDURES             │
                    └──────────────────────────────────────────┘
                                                                        */
  . SCAN SCB_LIST PTR(SCB_PTR);
  . . TCCB_PTR = SCB.TC_CB_PTR;
  . . IF  ¬EMPTY(TCCB.Q_PAC) THEN
  . .   DO;
  . .   . SEND 'OPEN_QUEUE' TO TC_OR_BF_TC.DEQUEUE.Q_PAC;      /* CHAPTER 4   */
  . .   . CALL HIGHER_LEVEL_DISPATCHER;                        /* PAGE C-13   */
  . .   END;
  . .
  . . SEND 'SEND_IPR' TO TC_OR_BF_TC.IPR_SEND;                 /* CHAPTER 4   */
  . . CALL HIGHER_LEVEL_DISPATCHER;                            /* PAGE C-13   */
  . .
  . . IF SCB.SCB_TYPE = HALF_SESS THEN
  . .   DO;
  . .   . IF  ¬EMPTY(SCB.Q_TC_TC_DFC) THEN
  . .   .   DO;
  . .   .   . SEND 'OPEN_QUEUE' TO DEQUEUE.Q_TC_TO_DFC;       /* CHAPTER 5   */
  . .   .   . CALL HIGHER_LEVEL_DISPATCHER;                    /* PAGE C-13   */
  . .   .   END;
  . .   END;
  . . ELSE                                                     /* SCB.SCB_TYPE = BF_SESS */
  . .   DO;
  . .   . TCCB_PTR = SCB.SEC_TO_BF_TC_CB_PTR;
  . .   . IF  ¬EMPTY(TCCB.Q_PAC) THEN
  . .   .   DO;
  . .   .   . SEND 'OPEN_QUEUE' TO TC_OR_BF_TC.DEQUEUE.Q_PAC;  /* CHAPTER 4   */
  . .   .   . CALL HIGHER_LEVEL_DISPATCHER;                    /* PAGE C-13   */
  . .   .   END;
  . .   .
  . .   . SEND 'SEND_IPR' TO TC_OR_BF_TC.IPR_SEND;            /* CHAPTER 4   */
  . .   . CALL HIGHER_LEVEL_DISPATCHER;                        /* PAGE C-13   */
  . .   .
  . .   END;
  . SCANEND;
  .
  END;

END HIGHER_LEVEL_SCHEDULER;
```

```
/*
┌──────────────────────────────────────────────────────────────────────────┐
│                    CANONICAL MESSAGE UNIT (MU) DEFINITION                │
│                                                                          │
│   FUNCTION:  THE CANONICAL  MESSAGE UNIT  IS THE  STRUCTURE USED  TO ADDRESS  ALL  │
│              MU-RELATED  FIELDS THROUGHOUT  THE  ARCHITECTURE.   IT COMBINES  ALL  │
│              FIELDS OF ALL FID TYPES AND HAS ALL  THE FIELDS OF A PIU.  IN LAYERS  │
│              THAT WORK WITH BIU'S OR RU'S, ONLY THE APPROPRIATE FIELDS ARE FILLED  │
│              IN.                                                         │
└──────────────────────────────────────────────────────────────────────────┘
*/

   ENTITY(MU),

/*
          ┌──────────────────────────────────────────┐
          │ THE MESSAGE UNIT CONTROL BLOCK (MUCB) IS USED │
          │ TO CONTAIN  CONTROL INFORMATION RELATED  TO A │
          │ MESSAGE UNIT AS IT FLOWS THROUGH A NODE.  THE │
          │ MUCB IS UNIQUE AS A CONTROL BLOCK SINCE IT IS │
          │ A  PART OF  THE MESSAGE  UNIT  ITSELF AND  IS │
          │ CREATED AND DISCARDED WITH THE MESSAGE UNIT.  │
          └──────────────────────────────────────────┘
*/

 2 MUCB,
   3 SEND_CHECK_SENSE    BIT(32),       /* SENSE FIELDS ARE USED TO COMMUNICATE TO        */
   3 RECEIVE_CHECK_SENSE BIT(32),       /* THE END USER IN THIS NODE WHAT ERROR OCCURRED  */
   3 TG_SEND_PRTY        BIT(4),        /* WORKING PRIORITY                               */
   3 DIRECTION           BIT(1),        /* B'0' = SEND, B'1' = RECEIVE                    */
   3 SEND_CHECK          BIT(1),        /* INDICATES THAT SEND_CHECK_SENSE IS SET         */
   3 LDI                 BIT(1),        /* LOST DATA INDICATOR                            */
   3 XID                 BIT(1),        /* RU IS AN XID.  TH & RH NOT MEANINGFUL          */
   3 XID_LENGTH          FIXED(16) BIN, /* LENGTH OF XID. ONLY SET IF XID BIT ON          */
   3 PUCP_BASED_SESSION BIT(2),         /* B'00' = NOT, B'01' = PUCP_TO_PU                */
                                        /* B'10' = PU_TO_PUCP, B'11' = RESERVED           */
 2 TH,
   3 SNF               FIXED(16) BIN,   /* SEQUENCE NUMBER FIELD                          */
   3 DCF               FIXED(16) BIN,   /* DATA COUNT FIELD. FOR RECEIVED FID'S 2 AND     */
                                        /* 3, LENGTH IS SET BY MAP_TO_CANONICAL FROM      */
                                        /* A VALUE IN THE BTUCB                           */
   3 DSAF              BIT(32),         /* DESTINATION SUBAREA FIELD                      */
   3 OSAF              BIT(32),         /* ORIGIN SUBAREA FIELD                           */
   3 DEF               BIT(16),         /* DESTINATION ELEMENT FIELD                      */
   3 OEF               BIT(16),         /* ORIGIN ELEMENT FIELD                           */
   3 DAF               BIT(16),         /* DESTINATION ADDRESS FIELD                      */
   3 OAF               BIT(16),         /* ORIGIN ADDRESS FIELD                           */
   3 DAFPRIME          BIT(8),          /* DESTINATION ADDRESS FIELD                      */
   3 OAFPRIME          BIT(8),          /* ORIGIN ADDRESS FIELD                           */
   3 LSID,
     4 LU_PU_IND       BIT(1),          /* B'0' = PU, B'1' = LU                           */
     4 LU_SSCP_IND     BIT(1),          /* B'0' = SSCP, B'1' = LU                         */
     4 LOCAL_ADDRESS   BIT(6),          /* LOCAL ADDRESS                                  */
   3 FID               BIT(4),          /* FORMAT ID                                      */
   3 BBIUI             BIT(1),          /* BEGIN BIU SEGMENTING FLAG                      */
   3 EBIUI             BIT(1),          /* END BIU SEGMENTING FLAG                        */
   3 EFI               BIT(1),          /* EXPEDITED FLOW INDICATOR                       */
   3 TG_SWEEP          BIT(1),          /* SWEEP = THIS PIU DOES NOT OVERTAKE ANY OTHER   */
   3 ER_VR_SUPP_IND    BIT(1),          /* PRE_ER_VR = NODE ON ROUTE DOES NOT SUPPORT     */
                                        /* ER'S & VR'S                                    */
   3 VR_PAC_CNT_IND    BIT(1),          /* PAC_CNT_0 = VR PACING COUNT HAS REACHED 0      */
   3 NTWK_PRTY         BIT(1),          /* N_PRTY = PIU FLOWS AT NETWORK PRIORITY         */
   3 IERN              BIT(4),          /* INITIAL ERN                                    */
   3 ERN               BIT(4),          /* EXPLICIT ROUTE NUMBER                          */
   3 VRID,
     4 VRN             BIT(4),          /* VIRTUAL ROUTE NUMBER                           */
     4 RESERVED        BIT(2),          /* RESERVED BITS                                  */
     4 TPF             BIT(2),          /* TRANSMISSION PRIORITY FIELD                    */
                                        /* 00 = LOW, 01 = MEDIUM, 10 = HIGH               */
   3 VR_CWI            BIT(1),          /* 1 = DECR RQ ON, 0 = OTHERWISE                  */
   3 TG_NONFIFO_IND    BIT(1),          /* 1 = ¬TG REFIFO, 0 = TG REFIFO                  */
   3 VR_SQTI           BIT(2),          /* NSEQ_NSUP, NSEQ_SUP, OR SING_SUP               */
   3 TG_SNF            BIT(12),         /* USED BY VR, TG_NONFIFO_IND = 1                 */
   3 VRPRQ             BIT(1),          /* 1 = VR PACING RQ ON, 0 = OFF                   */
   3 VRPRS             BIT(1),          /* 1 = VR PACING RSP ON, 0 = OFF                  */
   3 VR_CWRI           BIT(1),          /* 1 = DECR WINDOW BY 1, 0 = INCR WINDOW BY 1     */
   3 VR_RWI            BIT(1),          /* 1 = RESET WINDOW TO 1, 0 = OTHERWISE           */
   3 VR_SNF_SEND       BIT(12),         /* SEND SEQ NUMBER                                */
   3 SNAI              BIT(1),          /* 0 = PRE_SNA, 1 = OTHERWISE                     */
   3 CMD_FORMAT        BIT(8),          /* FIDF, X'01' = TG_CMD                           */
   3 CMD_TYPE          BIT(8),          /* FIDF, X'01' = TG_SNF_WRAP_ACK                  */
   3 CMD_SEQ_NUM       FIXED(16) BIN,   /* FIDF, COMMAND SEQ NUMBER                       */
```

```
  2 RH,
    3 RRI               BIT(1),       /* RQ = REQUEST, RSP = RESPONSE                */
    3 RU_CTGY           BIT(2),       /* FMD, NC, DFC, OR SC                         */
    3 FI                BIT(1),       /* 1 = FMH OR NSH                              */
    3 SDI               BIT(1),       /* SD = SENSE DATA INCLUDED                    */
    3 BCI               BIT(1),       /* BC = FIRST IN CHAIN                         */
    3 ECI               BIT(1),       /* EC = LAST IN CHAIN                          */
    3 DR1I              BIT(1),       /* FOR A REQUEST, THE ENCODINGS ARE            */
    3 DR2I              BIT(1),       /*  RQN = DR1I=¬DR1 & DR2I=¬DR2 & ERI=¬ER      */
    3 ERI               BIT(1),       /*  RQE = (DR1I=DR1 | DR2I=DR2) & ERI=ER       */
                                      /*  RQD = (DR1I=DR1 | DR2I=DR2) & ERI=¬ER      */
    3 QRI               BIT(1),       /* QR = ENQUEUE RESPONSE IN TC QUEUE           */
    3 PI                BIT(1),       /* PAC = PACING RESPONSE                       */
    3 BBI               BIT(1),       /* BB = BEGIN BRACKET                          */
    3 EBI               BIT(1),       /* EB = END BRACKET                            */
    3 CDI               BIT(1),       /* CD = CHANGE DIRECTION                       */
    3 CSI               BIT(1),       /* ENCODING USED, CODE0 OR CODE1               */
    3 EDI               BIT(1),       /* ED = RU IS ENCIPHERED                       */
    3 PDI               BIT(1),       /* PD = RU IS PADDED                           */
  2 SNC                 BIT(32),      /* SENSE DATA.  ONLY FILLED IN IF SDI = SD     */
  2 RU                  CHAR(1024);   /* LENGTH OF RU IS DCF - 3(LENGTH OF RH)       */

DCL RTI BIT(1) BASED(ADDR(ERI));      /* FOR A RESPONSE, B'0' = POS & B'1' = NEG     */
                                                                                    /*
```

```
| REQUEST-RESPONSE UNIT (RU) DEFINITIONS |
```

```
                                                                                    */
DCL 1 NS_REQUEST UNALIGNED BASED(ADDR(RU)),          /* FOR NS REQUESTS            */
    2 SERVICE_TYPE BIT(8),                           /* PU OR LU OR EITHER         */
    2 NS_CATEGORY BIT(8),                            /* SAME OR CROSS DOMAIN AND   */
                                                     /* TYPE OF SERVICE            */
    2 NS_RQ_CODE BIT(8);
DCL RQ_CODE BIT(8) BASED(ADDR(RU));
DCL 1 NSC_RQ UNALIGNED BASED(ADDR(RU)),              /* ACCESS TO TARGET ADDRESS   */
    2 NS_HEADER BIT(24),
    2 TARGET_ADDRESS BIT(16);
```

```
/*
   ┌──────────────────────────────────────────────────────────────────────┐
   │                    DISPATCHING QUEUE ENTRY (DQE)                     │
   │                                                                      │
   │      FUNCTION:  THIS  ENTITY IS  CREATED BY  THE SEND  STATEMENT AND  PLACED ON  THE  │
   │                 DISPATCHING QUEUE (DISPQ).   IT IS REMOVED FROM  DISPQ AND PROCESSED │
   │                 IN PROCEDURE DISPATCHER (PAGE C-13).                 │
   └──────────────────────────────────────────────────────────────────────┘
*/
ENTITY(DQE),                          /* DISPATCHING QUEUE ENTRY          */
    2 MUPTR PTR,                      /* MESSAGE UNIT PTR                 */
    2 LSCBPTR PTR,                    /* LINK STATION CONTROL BLOCK PTR   */
    2 SCBPTR PTR,                     /* SESSION CONTROL BLOCK PTR        */
    2 TGCBPTR PTR,                    /* TRANSMISSION GROUP CONTROL BLOCK PTR */
    2 ERCBPTR PTR,                    /* EXPLICIT ROUTE CONTROL BLOCK PTR */
    2 VRCBPTR PTR,                    /* VIRTUAL ROUTE CONTROL BLOCK PTR  */
    2 NRCBPTR PTR,                    /* NODE RESOURCE CONTROL BLOCK PTR  */
    2 DRCBPTR PTR,                    /* DOMAIN RESOURCE CONTROL BLOCK PTR */
    2 TCCBPTR PTR,                    /* TRANSMISSION CONTROL CONTROL BLOCK PTR */
    2 PARMPTR PTR,                    /* PARAMETER ENTITY PTR             */
    2 INPUT_SIGNAL FIXED(8) BIN,      /* INPUT SIGNAL NUMBER              */
    2 DEST_PROC FIXED(8) BIN,         /* DESTINATION PROCEDURE NUMBER     */
    2 SEND_PROC FIXED(8) BIN;         /* SENDING PROCEDURE NUMBER         */

/*
   ┌──────────────────────────────────────────────────────────────────────┐
   │                  BASIC TRANSMISSION UNIT (BTU)                       │
   │                                                                      │
   │      FUNCTION:  THE BTU IS THE  UNIT OF SNA REQUESTS AND RESPONSES  TRANSMITTED IN A │
   │                 SINGLE BASIC  LINK UNIT (BLU).   IT CONSISTS  OF ONE OR  MORE PIU'S. │
   │                 THIS ENTITY  CAPTURES THE DATA  OF THE BTU  PLUS ITS LENGTH  AND THE │
   │                 LINK STATION AT WHICH IT WAS RECEIVED.   THE BTU IS USED IN THE DATA │
   │                 LINK CONTOL LAYER AND IN THE  TRANSMISSION GROUP CONTROL SUBLAYER OF │
   │                 PATH CONTROL.                                        │
   └──────────────────────────────────────────────────────────────────────┘
*/

ENTITY (BTU),
  2 BTUCB,
    3 BTU_LENGTH FIXED(16) BIN,       /* LENGTH OF BTU                    */
    3 LSCBPTR PTR,                    /* LINK STATION FROM WHICH THE BTU ARRIVED */
    3 LDI BIT(1),                     /* LOST DATA INDICATOR              */

  2 BTU_DATA CHAR(*);                 /* ONE OR MORE PIUS. ARBITRARY MAX LENGTH */
```

TH FORMATS

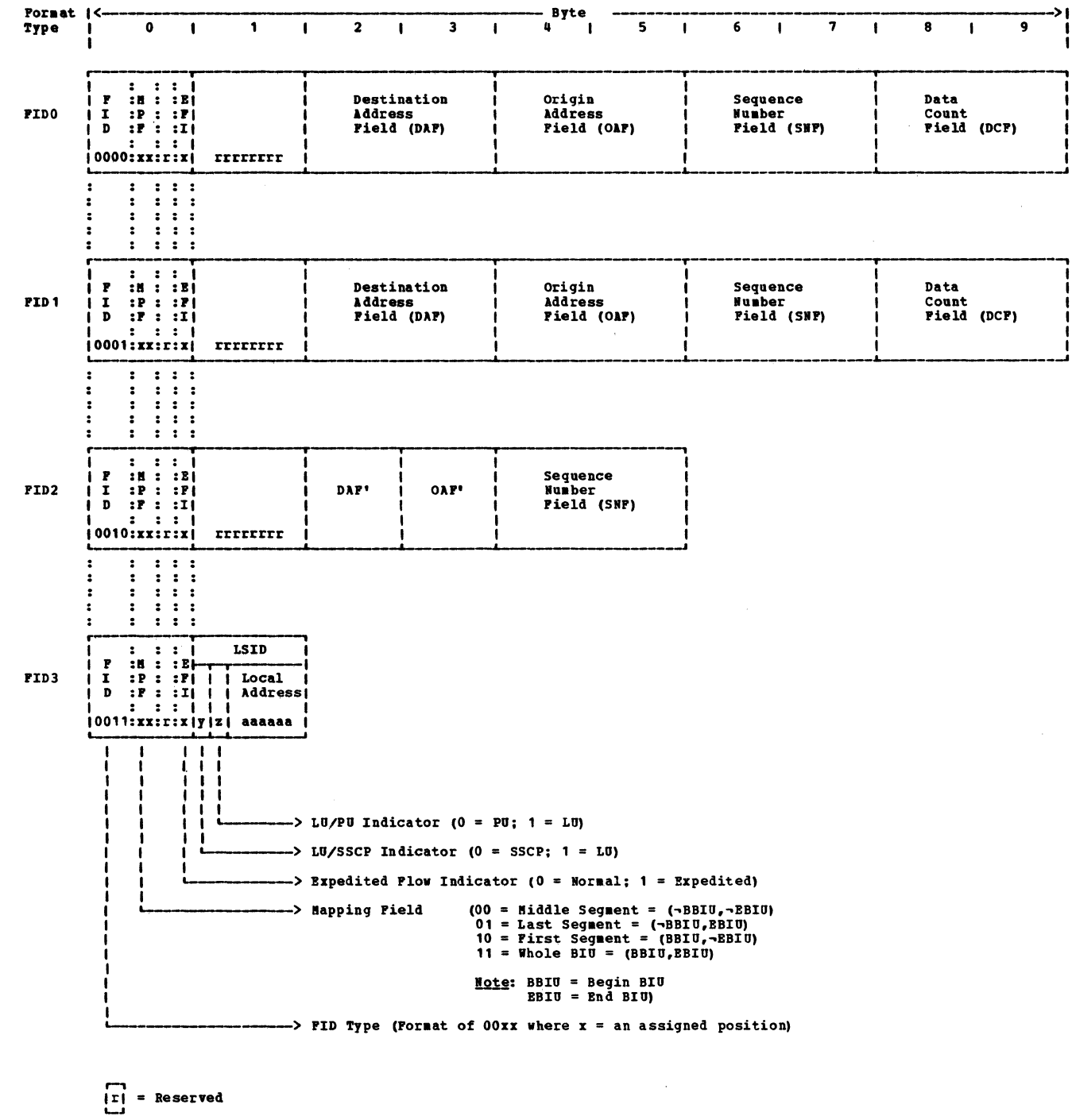


Figure D-1. TH Formats: FID0-FID3 (Part 1 of 4)

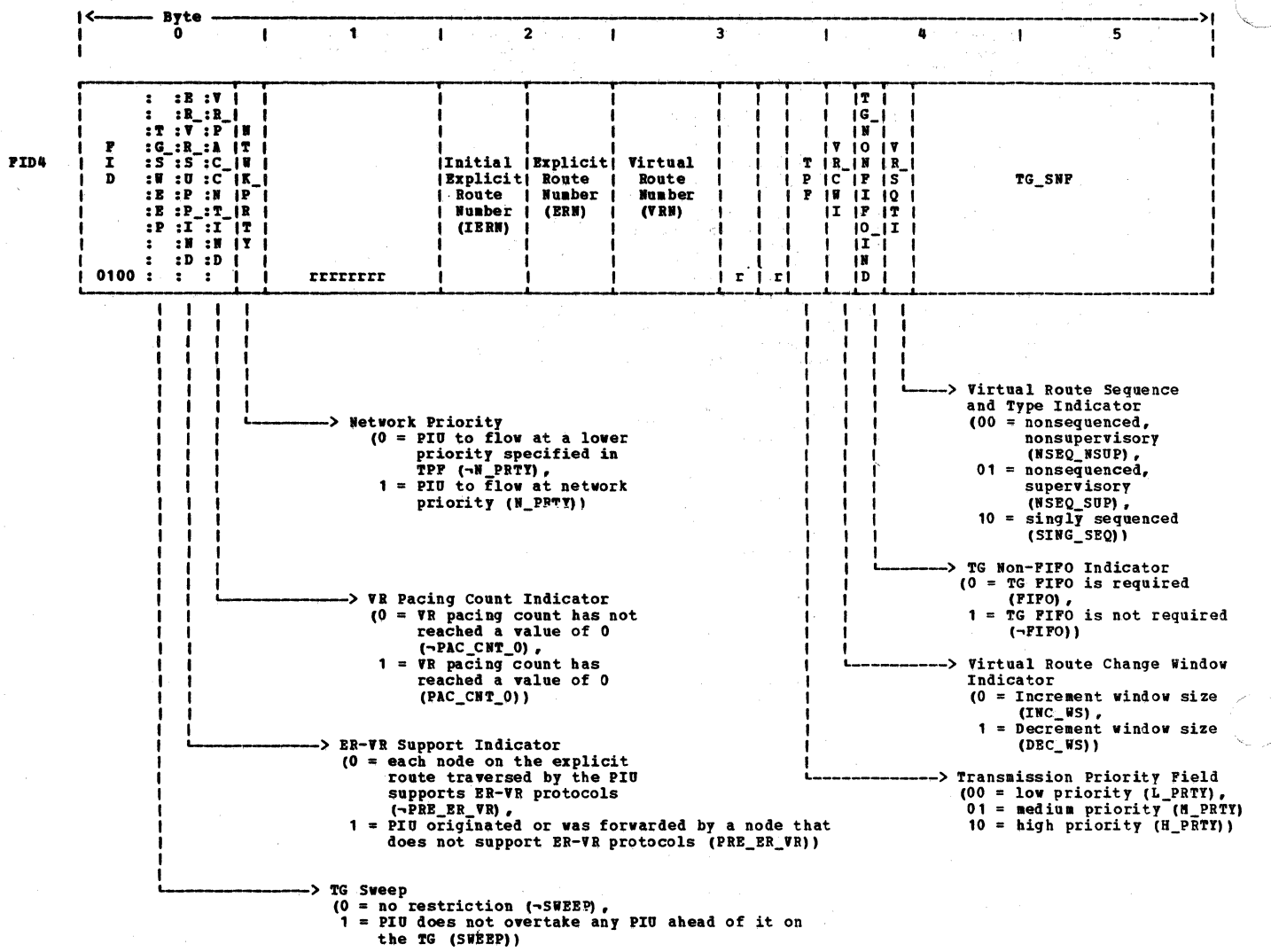


Figure D-2. TH Formats: FID4 (Part 2 of 4)

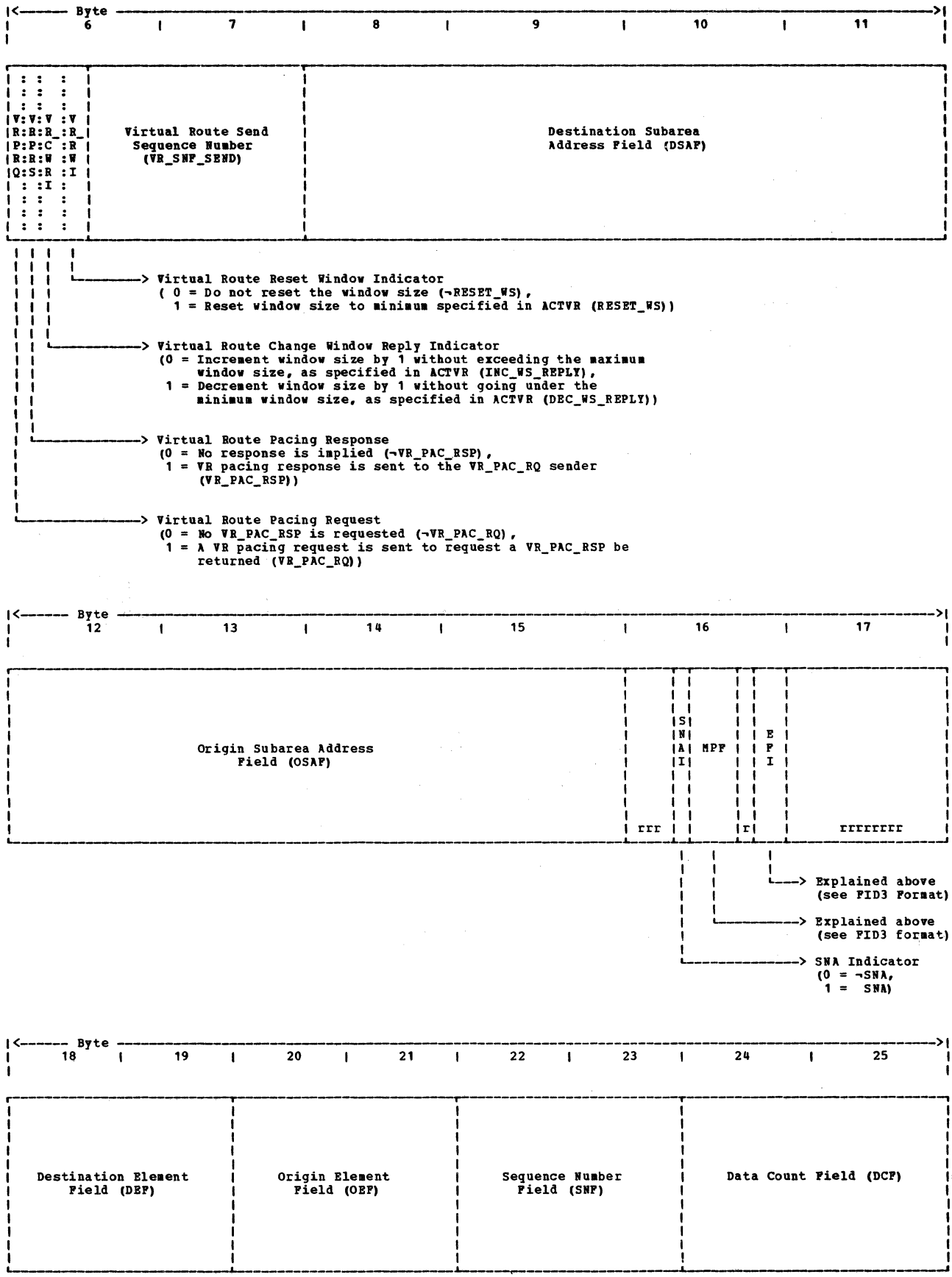


Figure D-3. TH Formats: FID4 Continued (Part 3 of 4)

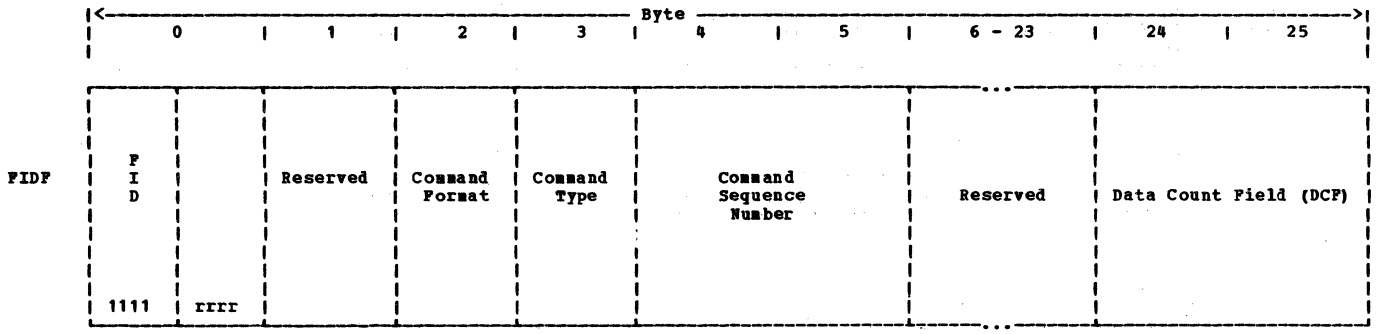
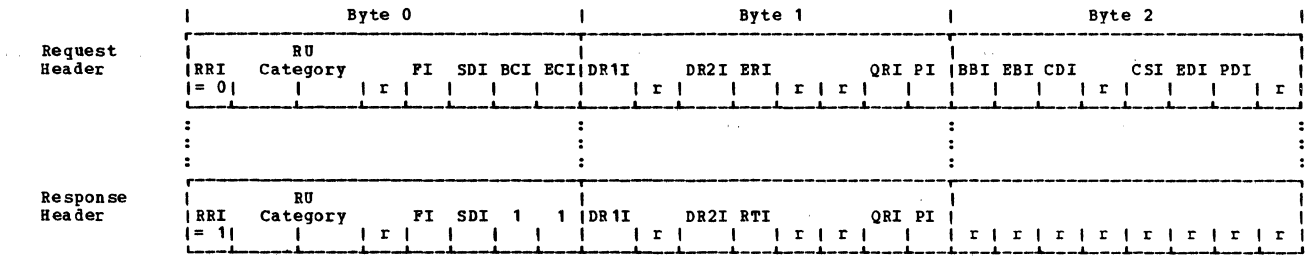


Figure D-4. TH Formats: FIDF (Part 4 of 4)

RH FORMATS



Field	Description	Explanation/Usage
RRI	Request-Response indicator	0 = request (RQ); 1 = response (RSP)
RU Category	Request-Response Unit Category	00 = FM Data (FMD) 01 = Network Control (NC) 10 = Data Flow Control (DFC) 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; 1 = FM header (FMH) follows, for LU-LU sessions; or field-formatted with an NS header (NSH), for network services
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: DR1I, DR2I, ERI = 000 means no-response requested = 100|010|110 means definite-response requested = 101|011|111 means exception-response requested (001 is reserved)
RTI	Response Type indicator	0 = positive (+); 1 = negative (-)
QRI	Queued Response indicator	0 = response bypasses TC queues (¬QR); 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
CDI	Change Direction indicator	0 = do not change direction (¬CD); 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)

[r] = Reserved

Figure D-5. RH Formats

This page intentionally left blank

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 control lists, which are used in multiple RUs, and the (DLC) XID command and response information-field formats.

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
NC	network control
FMD NS(c)	function management data, network services, configuration services
FMD NS(ma)	function management data, network services, maintenance services
FMD NS(me)	function management data, network services, measurement services
FMD NS(mn)	function management data, network services, management services
FMD NS(no)	function management data, network services, network operator 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 SNA LU-LU Session Types.

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, a "symbolic name in EBCDIC characters" is defined in general accordance with the System/360 or System/370 Assembler Language definition of an "ordinary symbol": the name must begin with any one of the EBCDIC letters--A through Z, $, #, or @--and be followed by zero or more EBCDIC letters or numerics (0-9).

SUMMARY OF REQUEST RU'S BY CATEGORY

NC			
+LSA	NC_ER_ACT_REPLY	NC_ER_TEST	NC_IPL_FINAL
NC_ACTVR	NC_ER_INOP	NC_ER_TEST_REPLY	NC_IPL_INIT
NC_DACTVR	NC_ER_OP	NC_IPL_ABORT	NC_IPL_TEXT
NC_ER_ACT			
SC			
*ACTCDRM	CLEAR	DACTLU	*STSN
*ACTLU	CRV	DACTPU	SDT
*ACTPU	DACTCDRM	RQR	UNBIND
*BIND			
DFC			
BID	LUSTAT	RSHUTD	SHUTC
BIS	QC	RTR	SHUTD
CANCEL	QEC	SBI	SIG
CHASE	RELQ		
FMD NS(c)			
ABCONN	DACTLINK	INOP	+NS_LSA
ABCONNOUT	DISCONTACT	IPLFINAL	PROCSTAT
ACTCONNIN	DELETENR	IPLINIT	REQACTLU
ACTLINK	DUMPFINAL	IPLTEXT	REQCONT
*ADDLINK	*DUMPINIT	LCP	REQDISCONT
*ADDLINKSTA	*DUMPTEXT	LDREQD	REQFNA
+ANA	ER_INOP	NS_IPL_ABORT	*RNAA
CONNOUT	ESLOW	NS_IPL_FINAL	RPO
CONTACT	EXSLOW	NS_IPL_INIT	SETCV
CONTACTED	FNA	NS_IPL_TEXT	VR_INOP
DACTCONNIN	INITPROC	NS_LD_REQD	
FMD NS(ma)			
ACTTRACE	EXECTEST	RECTR	REQTEST
DACTTRACE	RECFMS	RECTRD	*ROUTE_TEST
DISPSTOR	RECMS	REQECHO	SETCV
ECHOTEST	RECSTOR	REQMS	TESTMODE
ER_TESTED	RECTD		
FMD NS(mn)			
DELIVER	FORWARD		
FMD NS(s)			
BINDF	CDTAKED	*DSRLST	SESSEND
CDCINIT	CDTAKEDC	INIT-OTHER	SESSST
*CDINIT	*CDTERM	*INIT-OTHER-CD	TERM-OTHER
*CDSESSEND	*CINIT	INIT-SELF	TERM-OTHER-CD
CDSESSSF	CLEANUP	NOTIFY	TERM-SELF
CDSESSST	CTERM	NSPE	UNBINDF
CDSESSTF			

* 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 RUs" and "Positive Response RUs with Extended Formats."

\+ These RUs are supported only for subarea nodes that are not at the current level of SNA.

INDEX OF RU'S BY NS HEADERS AND REQUEST CODES

Within DFC, NC, 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 (e.g., X'05') can represent different requests in separate categories (e.g., DFC, NC, and configuration services). DSRLST, NOTIFY, and SETCV are exceptions: these three requests have request codes--X'27', X'20', and X'11', respectively--that are unique across all the FMD NS categories.

FMD NS Headers (Third byte is the request code)

X'010201'	CONTACT
X'010202'	DISCONTACT
X'010203'	IPLINIT
X'010204'	IPLTEXT
X'010205'	IPLFINAL
X'010206'	DUMPINIT
X'010207'	DUMPTEXT
X'010208'	DUMPFINAL
X'010209'	RPO
X'01020A'	ACTLINK
X'01020B'	DACTLINK
X'01020E'	CONNOUT
X'01020F'	ABCONN
X'010211'	SETCV (FMD NS(c))
X'010214'	ESLOW
X'010215'	EXSLOW
X'010216'	ACTCONNIN
X'010217'	DACTCONNIN
X'010218'	ABCONNOUT
X'010219'	ANA
X'01021A'	FNA
X'01021B'	REQDISCONT
X'010280'	CONTACTED
X'010281'	INOP
X'010284'	REQCONT
X'010285	NS_LSA
X'010301'	EXECTEST
X'010302'	ACTTRACE
X'010303'	DACTTRACE
X'010311'	SETCV (FMD NS(ma))
X'010331'	DISPSTOR
X'010334'	RECSTOR
X'010380'	REQTEST
X'010381'	RECMS
X'010382'	RECTD
X'010383'	RECTRD
X'010604'	NSPE
X'010681'	INIT-SELF (Format 0)
X'010683'	TERM-SELF (Format 0)
X'410210'	RNAA

X'41021C'	DELETENR
X'41021D'	ER_INOP
X'41021E'	ADDLINK
X'410221'	ADDLINKSTA
X'410223'	VR_INOP
X'410235'	INITPROC
X'410236'	PROCSTAT
X'410237'	NS_LD_REQD
X'410240'	REQACTLU
X'410243'	NS_IPL_INIT
X'410244'	NS_IPL_TEXT
X'410245'	NS_IPL_FINAL
X'410246'	NS_IPL_ABORT
X'410286'	REQFNA
X'410287'	LCP
X'410304'	REQMS
X'410305'	TESTMODE
X'410306'	ROUTE_TEST
X'410384'	RECFMS
X'410385'	RECTR
X'410386'	ER_TESTED
X'810387'	REQECHO
X'810389'	ECHOTEST
X'810601'	CINIT
X'810602'	CTERM
X'810620'	NOTIFY (SSCP-->LU)
X'810629'	CLEANUP
X'810680'	INIT-OTHER
X'810681'	INIT-SELF (Format 1)
X'810682'	TERM-OTHER
X'810683'	TERM-SELF (Format 1)
X'810685'	BINDF
X'810686'	SESSST
X'810687'	UNBINDF
X'810688'	SESSEND
X'810810'	FORWARD
X'810812'	DELIVER
X'818620'	NOTIFY (SSCP-->SSCP)
X'818627'	DSRLST
X'818640'	INIT-OTHER-CD
X'818641'	CDINIT
X'818642'	TERM-OTHER-CD
X'818643'	CDTERM
X'818645'	CDSESSSF
X'818646'	CDSESSST
X'818647'	CDSESSTF
X'818648'	CDSESSEND
X'818649'	CDTAKED
X'81864A'	CDTAKEDC
X'81864B'	CDCINIT

DFC, NC, and SC Request Codes

X'02'	NC_IPL_FINAL
X'03'	NC_IPL_INIT
X'04'	NC_IPL_TEXT (NC)
X'04'	LUSTAT (DFC)
X'05'	RTR (DFC)
X'05'	LSA (NC)
X'06'	NC_ER_INOP
X'07'	ANSC
X'09'	NC_ER_TEST
X'0A'	NC_ER_TEST_REPLY
X'0B'	NC_ER_ACT
X'0C'	NC_ER_ACT_REPLY
X'0D'	ACTLU (SC)
X'0D'	NC_ACTVR (NC)
X'0E'	DACTLU (SC)
X'0E'	NC_DACTVR (NC)
X'0F'	NC_ER_OP
X'11'	ACTPU
X'12'	DACTPU
X'14'	ACTCDRM
X'15'	DACTCDRM
X'31'	BIND
X'32'	UNBIND
X'46'	NC_IPL_ABORT
X'70'	BIS
X'71'	SBI
X'80'	QEC
X'81'	QC
X'82'	RELQ
X'83'	CANCEL
X'84'	CHASE
X'A0'	SDT
X'A1'	CLEAR
X'A2'	STSN
X'A3'	RQR
X'C0'	CRV (SC)
X'C0'	SHUTD (DFC)
X'C1'	SHUTC
X'C2'	RSHUTD
X'C8'	BID
X'C9'	SIG

REQUEST RU FORMATS

ABCONN; SSCP-->PU_T4|5, PUCP-->PU, Norm; FMD NS(c) (ABANDON CONNECTION)

```
DCL 1 ABCONN_RQ                 BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                         BIT(24), /*  0-2   */
      2 LINK_ADDRESS                      BIT(16); /*  3-4   */
```

0-2 X'01020F' NS header
3-4 Network address of link

ABCONNOUT; SSCP-->PU_T4|5, PUCP-->PU, Norm; FMD NS(c) (ABANDON CONNECT OUT)

```
DCL 1 ABCONNOUT_RQ              BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                         BIT(24), /*  0-2   */
      2 LINK_ADDRESS                      BIT(16); /*  3-4   */
```

0-2 X'010218' NS header
3-4 Network address of link

ACTCDRM; SSCP-->SSCP, Exp; SC (ACTIVATE CROSS-DOMAIN RESOURCE MANAGER)

```
DCL 1 ACTCDRM_RQ                BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                           BIT(8), /*   0    */
      2 FORMAT                            BIT(4), /*   1    */
      2 TYPE_ACTIVATION                   BIT(4),
      2 FM_PROFILE                        BIT(8), /*   2    */
      2 TS_PROFILE                        BIT(8), /*   3    */
      2 CONTENTS_ID                      CHAR(8), /*  4-11  */
      2 SSCP_ID                          CHAR(6), /* 12-17  */
      2 RESERVED                          BIT(2), /*   18   */
      2 PRI_RCV_PAC_CNT                   BIT(6),
      2 CONTROL_VECTORS                 CHAR(*); /*  19-n  */
```

0 X'14' request code
1 bits 0-3, format: X'0' (only value defined)
bits 4-7, type activation requested:
X'1' cold
X'2' ERP
2 FM profile (see Appendix F)
3 TS profile (see Appendix F)
4-11 Contents ID: eight-character EBCDIC symbolic name that represents implementation and installation dependent information about the SSCP issuing the ACTCDRM; eight space (X'40') characters is the value used if no information is to be conveyed (This field could be used to provide a check for a functional and configurational match between the SSCPs.)
12-17 SSCP ID: a six-byte field that includes the ID of the SSCP issuing the ACTCDRM; the first four bits specify the format for the remaining bits:

```
                bits 0-3, format 0000 (only value defined)
                bits 4-7, physical unit type (see Appendix F) of
                          the node containing the SSCP
                bits 8-47, implementation and installation
                           dependent binary identification
        18      TS Usage
                bits 0-1, reserved
                bits 2-7, primary CPMGR receive pacing count (zero
                          means no pacing of requests flowing to
                          the primary)
        19-n    One or more control vectors, as described in the
                section "Control Vectors and Control Lists," later
                in this appendix
                Note: The following vector keys may be used in
                ACTCDRM:
                 X'06'  CDRM control vector
                 X'09'  activation request/response sequence
                        identifier control vector

ACTCONNIN; SSCP-->PU_T4|5, PUCP-->PU, Norm; FMD NS(c) (ACTIVATE
        CONNECT IN)

        DCL 1 ACTCONNIN_RQ            BASED(ADDR(RU)), /* Byte(s)*/
              2 NS_HEADER                     BIT(24), /*  0-2   */
              2 LINK_ADDRESS                  BIT(16), /*  3-4   */
              2 TYPE                           BIT(1), /*   5    */
              2 RESERVED                       BIT(7);

        0-2       X'010216' NS header
        3-4       Network address of link
        5         bit 0, type: 0 (only value defined)
                  bits 1-7, reserved

ACTLINK; SSCP-->PU_T4|5, PUCP-->PU, Norm; FMD NS(c) (ACTIVATE LINK)

        DCL 1 ACTLINK_RQ              BASED(ADDR(RU)), /* Byte(s)*/
              2 NS_HEADER                     BIT(24), /*  0-2   */
              2 LINK_ADDRESS                  BIT(16); /*  3-4   */

        0-2       X'01020A' NS header
        3-4       Network address of link

ACTLU; SSCP-->LU, Exp; SC (ACTIVATE LOGICAL UNIT)

        DCL 1 ACTLU_RQ                BASED(ADDR(RU)), /* Byte(s)*/
              2 RQ_CODE                       BIT(8), /*   0    */
              2 TYPE_ACTIVATION               BIT(8), /*   1    */
              2 FM_PROFILE                    BIT(4), /*   2    */
              2 TS_PROFILE                    BIT(4);

        0         X'0D' request code
        1         Type activation requested:
                   X'01'  cold
                   X'02'  ERP
```

```
2          bits 0-3, FM profile (see Appendix F)
           bits 4-7, TS profile (see Appendix F)
```

ACTPU; SSCP|PUCP-->PU, Exp; SC (ACTIVATE PHYSICAL UNIT)

```
DCL 1 ACTPU_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                          BIT(8), /*   0    */
      2 FORMAT                           BIT(4), /*   1    */
      2 TYPE_ACTIVATION                  BIT(4),
      2 FM_PROFILE                       BIT(4), /*   2    */
      2 TS_PROFILE                       BIT(4),
      2 SSCP_ID                        CHAR(6), /*  3-8   */
      2 CONTROL_VECTORS                CHAR(*); /*  9-n   */
```

```
0          X'11' request code
1          bits 0-3, format:
                     X'0'  Format 0
                     X'3'  Format 3; same as Format 0,
                           except that it includes one or
                           more control vectors in bytes 9-n
                           (sent only to PU_T4|5s that
                           support ERs and VRs)
           bits 4-7, type activation requested:
                     X'1'  cold
                     X'2'  ERP
2          bits 0-3, FM profile (see Appendix F)
           bits 4-7, TS profile (see Appendix F)
3-8        A six-byte field that specifies the ID of the SSCP
           issuing ACTPU; the first four bits specify the
           format for the remaining bits:
           bits 0-3, format: 0000 (only value defined)
           bits 4-7, PU type (see Appendix F) of the node
                     containing the SSCP
           bits 8-47, implementation and installation
                     dependent binary identification
Note:  End of Format 0; Format 3 continues below
9-n        One or more control vectors, as described in the
           section "Control Vectors and Control Lists," later
           in this appendix
           Note: The following vector keys may be used in
           ACTPU:
            X'09'  activation request/response sequence
                   identifier control vector
            X'0B'  SSCP-PU session capabilities control
                   vector
```

ACTTRACE; SSCP-->PU_T4|5, Norm; FMD NS(ma) (ACTIVATE TRACE)

```
DCL 1 ACTTRACE_RQ               BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 LINK_ADDRESS                     BIT(16), /*  3-4   */
      2 TRACE_TYPE                        BIT(8), /*   5    */
      2 TRACE_DATA                     CHAR(*); /*  6-n   */
```

```
          0-2       X'010302' NS header
          3-4       Network address of the resource to be traced
          5         Selected trace:
                    bit 0, transmission group trace
                    bits 1-6, reserved
                    bit 7, link trace
          6-n       Data to support trace
```

ADDLINK; SSCP-->PU_T4|5, Norm; FMD NS(c) (ADD LINK)

```
          DCL 1 ADDLINK_RQ                BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                       BIT(24), /*  0-2   */
                2 PU_ADDRESS                      BIT(16), /*  3-4   */
                2 RESERVED                        BIT(16), /*  5-6   */
                2 LOCAL_LINK_ID_LENGTH          FIXED(8), /*   7    */
                2 LOCAL_LINK_ID
                      CHAR(REFER(LOCAL_LINK_ID_LENGTH)); /*  8-n   */

          0-2       X'41021E' NS header
          3-4       Network address of target PU
          5-6       Reserved
          7         Length of local link identifier
          8-n       Local link identifier
```

ADDLINKSTA; SSCP-->PU_T4|5, Norm; FMD NS(c) (ADD LINK STATION)

```
          DCL 1 ADDLINKSTA_RQ             BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                       BIT(24), /*  0-2   */
                2 TARGET_ADDRESS                  BIT(16), /*  3-4   */
                2 FID_TYPES                        BIT(8), /*   5    */
                2 RESERVED                         BIT(8), /*   6    */
                2 LINK_STA_ID_LENGTH            FIXED(8), /*   7    */
                2 LINK_STA_ID
                        CHAR(REFER(LINK_STA_ID_LENGTH)); /*  8-n   */

          0-2       X'410221' NS header
          3-4       Network address of target PU or link
          5         FID types supported:
                    bit 0,  1  FID0 support
                    bit 1,  1  FID1 support
                    bit 2,  1  FID2 support
                    bit 3,  1  FID3 support
                    bit 4,  1  FID4 support
                    bits 5-7, Reserved
          6         Reserved
          7         Length of link station identifier
                    Note: When assigning an address for a link
                    station on a point to point link, this field can
                    be 0, the link station identifier is omitted, and
                    the target network address in bytes 3 and 4
                    indicates the link to which the link station
                    belongs.
          8-n       Link station identifier
```

ANA; SSCP-->PU_T4|5, Norm; FMD NS(c) (ASSIGN NETWORK ADDRESSES)

```
          DCL 1 ANA_RQ                      BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                         BIT(24), /*  0-2   */
                2 PU_ADDRESS                        BIT(16), /*  3-4   */
                2 NUM_ADDRESSES                      BIT(8), /*   5    */
                2 TYPE                               BIT(8), /*   6    */
                2 NETWORK_ADDRESS(1:REFER(NUM_ADDRESSES))
                                                    BIT(16); /*  7-n   */
```

0-2	X'010219' NS header
3-4	Network address of PU associated with the node to which LU network addresses are to be assigned
5	Number of network addresses to be assigned
6	Type: X'80' noncontiguous (only value defined)
7-8	First network address
9-n	Any additional network addresses (two-byte multiples)

BID; LU-->LU, Norm; DFC (BID)

```
          DCL 1 BID_RQ                      BASED(ADDR(RU)), /* Byte(s)*/
                2 RQ_CODE                           BIT(8); /*   0    */
```

0	X'C8' request code

BIND; PLU-->SLU, Exp; SC (BIND SESSION)

```
          DCL 1 BIND_RQ                     BASED(ADDR(RU)), /* Byte(s)*/
                2 RQ_CODE                           BIT(8), /*   0    */
                2 FORMAT                            BIT(4), /*   1    */
                2 TYPE                              BIT(4),
                2 FM_PROFILE                        BIT(8), /*   2    */
                2 TS_PROFILE                        BIT(8), /*   3    */
                2 PRI_CHAIN_USE                     BIT(1), /*   4    */
                2 PRI_RQ_MODE                       BIT(1),
                2 PRI_CHAIN_RSP                     BIT(2),
                2 PRI_TWO_PHASE_COMMIT              BIT(1),
                2 RESERVED                          BIT(1),
                2 PRI_COMPRESSION_IND               BIT(1),
                2 PRI_EB_IND                        BIT(1),
                2 SEC_CHAIN_USE                     BIT(1), /*   5    */
                2 SEC_RQ_MODE                       BIT(1),
                2 SEC_CHAIN_RSP                     BIT(2),
                2 SEC_TWO_PHASE_COMMIT              BIT(1),
                2 RESERVED                          BIT(1),
                2 SEC_COMPRESSION_IND               BIT(1),
                2 SEC_EB_IND                        BIT(1),
                2 RESERVED                          BIT(1), /*   6    */
                2 FM_HEADER_USAGE                   BIT(1),
                2 BRACKETS_USAGE                    BIT(1),
                2 BRACKET_TERM_RULE                 BIT(1),
                2 ALTERNATE_CODE                    BIT(1),
                2 SQN_AVAILABILITY                  BIT(1),
```

```
          2 BIS_SENT                          BIT(1),
          2 RESERVED                          BIT(1),
          2 SEND_RCV_MODE                     BIT(2), /*    7     */
          2 RECOVERY_RESPONSIBILITY           BIT(1),
          2 CONT_WINNER_LOSER                 BIT(1),
          2 RESERVED                          BIT(3),
          2 HDX_FF_RESET_STATE                BIT(1),
          2 SEC_TO_PRI_STAGING_IND            BIT(1), /*    8     */
          2 RESERVED                          BIT(1),
          2 SEC_SEND_PACING_CNT               BIT(6),
          2 RESERVED                          BIT(2), /*    9     */
          2 SEC_RCV_PACING_CNT                BIT(6),
          2 SEC_SEND_MAX_RU_SIZE              BIT(8), /*    10    */
          2 PRI_SEND_MAX_RU_SIZE              BIT(8), /*    11    */
          2 PRI_TO_SEC_STAGING_IND            BIT(1), /*    12    */
          2 RESERVED                          BIT(1),
          2 PRI_SEND_PACING_CNT               BIT(6),
          2 RESERVED                          BIT(2), /*    13    */
          2 PRI_RCV_PACING_CNT                BIT(6),
          2 PS_PROFILE,
            3 PS_USAGE_FMT                    BIT(1), /*    14    */
            3 LU_LU_SESSION_TYPE              BIT(7),
          2 PS_USAGE                        CHAR(11), /* 15-25    */
          2 CRYPTOGRAPHY_PRIVATE              BIT(2), /*    26    */
          2 CRYPTOGRAPHY_SESSION_LEVEL        BIT(2),
          2 CRYPTOGRAPHY_LENGTH               BIT(4),
          2 CRYPTOGRAPHY_KEY_ENCIPH_METHOD    BIT(2), /*    27    */
          2 RESERVED                          BIT(3),
          2 CRYPTOGRAPHY_CIPHER_METHOD        BIT(3),
          2 SESS_CRYPTOGRAPHY_KEY
                 CHAR(REFER(CRYPTOGRAPHY_LENGTH)), /*  28-k    */
          2 PLU_NTWK_NAME_LENGTH              BIT(8), /*  k+1     */
          2 PLU_NTWK_NAME
                CHAR(REFER(PLU_NTWK_NAME_LENGTH)), /* k+2-m    */
          2 USER_DATA_LENGTH                  BIT(8), /*  m+1     */
          2 USER_DATA
                   CHAR(REFER(USER_DATA_LENGTH)), /* m+2-n    */
          2 URC_LENGTH                        BIT(8), /*  n+1     */
          2 URC              CHAR(REFER(URC_LENGTH)), /* n+2-p    */
          2 SLU_NTWK_NAME_LENGTH              BIT(8), /*  p+1     */
          2 SLU_NTWK_NAME
                CHAR(REFER(SLU_NTWK_NAME_LENGTH)); /* p+2-r    */

0         X'31' request code
1         bits 0-3, format: 0000 (only value defined)
          bits 4-7, type:
                     0000  negotiable
                     0001  nonnegotiable
2         FM profile (see Appendix F)
3         TS profile (see Appendix F)
          FM Usage--Primary LU Protocols for FM Data
4         bit 0, chaining use selection:
                   0  only  single-RU  chains   allowed  from
                      primary LU half-session
```

1 multiple-RU chains allowed from primary LU half-session

bit 1, request control mode selection:
0 immediate request mode
1 delayed request mode

bits 2-3, chain response protocol used by primary LU half-session for FMD requests; chains from primary will ask for:
00 no response
01 exception response
10 definite response
11 definite or exception response

bit 4, 2-phase commit for sync point (reserved if sync point protocol not used, i.e., a TS profile other than 4 is used):
0 2-phase commit not supported
1 2-phase commit supported

bit 5, reserved

bit 6, compression indicator:
0 compression will not be used on requests from primary
1 compression may be used

bit 7, send End Bracket indicator
0 primary will not send EB
1 primary may send EB

FM Usage--Secondary LU Protocols for FM Data

5 bit 0, chaining use selection:
0 only single-RU chains allowed from secondary LU half-session
1 multiple-RU chains allowed from secondary LU half-session

bit 1, request control mode selection:
0 immediate request mode
1 delayed request mode

bits 2-3, chain response protocol used by secondary LU half-session for FMD requests; chains from secondary will ask for:
00 no response
01 exception response
10 definite response
11 definite or exception response

bit 4, 2-phase commit for sync point (reserved if sync point protocol not used, i.e., a TS profile other than 4 is used):
0 2-phase commit not supported
1 2-phase commit supported

bit 5, reserved

bit 6, compression indicator:
0 compression will not be used on requests from secondary
1 compression may be used

bit 7, send End Bracket indicator
0 secondary will not send EB

1 secondary may send EB

FM Usage--Common LU Protocols

6 bit 0, reserved

bit 1, FM header usage:
- 0 FM headers not allowed
- 1 FM headers allowed

bit 2, brackets usage and reset state:
- 0 brackets not used if neither primary nor secondary will send EB, i.e., if byte 4, bit 7 = 0 and byte 5, bit 7 = 0; brackets are used and bracket state managers' reset states are INB if either primary or secondary, or both, may send EB, i.e., if byte 4, bit 7 = 1 or byte 5, bit 7 = 1
- 1 brackets are used and bracket state managers' reset states are BETB

bit 3, bracket termination rule selection (reserved if brackets not used, i.e., if byte 6, bit 2 = 0, byte 4, bit 7 = 0, and byte 5, bit 7 = 0):
- 0 Rule 2 (unconditional termination) will be used during this session
- 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

bit 5, sequence number availability for sync point resynchronization (reserved if sync point protocol not used, i.e., a TS profile other than 4 is used):
- 0 sequence numbers not available
- 1 sequence numbers available

Note: Sequence numbers are transaction processing program sequence numbers from the previous activation of the session with the same session name; they are associated with the last acknowledged requests and any pending requests to commit a unit of work. If there was no previous activation, the numbers are 0, and this bit is set to 0.

bit 6, BIS sent (reserved if sync point protocol not used, i.e., a TS profile other than 4 is used):
- 0 BIS not sent
- 1 BIS sent

bit 7, reserved

7 bits 0-1, normal-flow send/receive mode selection:
- 00 full-duplex
- 01 half-duplex contention
- 10 half-duplex flip-flop
- 11 reserved

bit 2, recovery responsibility (reserved if normal

flow send/receive mode is FDX, i.e., if byte 7, bits 0-1 = 00):
0 contention loser responsible for recovery (see byte 7, bit 3 for specification of which half-session is the contention loser)
1 symmetric responsibility for recovery

bit 3, contention winner/loser (reserved if normal flow send/receive mode is FDX, i.e., if byte 7, bits 0-1 = 00; or if the normal flow send/receive mode is HDX-FF, brackets are not used, and symmetric responsibility for recovery is used, i.e., if byte 7, bits 0-1 = 10, byte 4, bit 7 = 0, byte 5, bit 7 = 0, byte 6, bit 2 = 0, and byte 7, bit 2 = 1):
0 secondary is contention winner and primary is contention loser
1 primary is contention winner and secondary is contention loser

Note: Contention winner is also brackets first speaker if brackets are used.

bits 4-6, reserved

bit 7, half-duplex flip-flop reset states (reserved unless (1) normal-flow send/receive mode is half-duplex flip-flop (byte 7, bits 0-1 = 10) and (2) brackets are not used or bracket state manager's reset state is INB (byte 6, bit 2 = 0)):
0 HDX-FF reset state is RECEIVE for the primary and SEND for the secondary (e.g., the secondary sends normal-flow requests first after session activation)
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)

TS Usage

8 bit 0, staging indicator for secondary CPMGR to primary CPMGR normal flow:
0 pacing in this direction occurs in one stage
1 pacing in this direction occurs in two stages

Note: The meanings of 0 and 1 are reversed from the staging indicator for primary CPMGR to secondary CPMGR.

bit 1, reserved

bits 2-7, secondary CPMGR's send pacing count: 0 means no pacing of requests flowing from the secondary

9 bits 0-1, reserved

bits 2-7, secondary CPMGR's receive pacing count: 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: 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, 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 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 CPMGR to secondary CPMGR normal flow:

- 1 pacing in this direction occurs in one stage
- 0 pacing in this direction occurs in two stages

Note: The meanings of 0 and 1 are reversed from the staging indicator for secondary to primary CPMGR.

bit 1, reserved

bits 2-7, primary CPMGR's send pacing count: 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 will indicate the same value as, the secondary CPMGR's receive pacing count--see byte 9, bits 2-7, above.)

13 bits 0-1, reserved

bits 2-7, primary CPMGR's receive pacing count: 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 will indicate the same value as, the secondary CPMGR's send pacing count--see

byte 8, bits 2-7, above.)

PS Profile

14 bit 0, PS Usage field format:
- 0 basic format
- 1 reserved

bits 1-7, LU-LU session type

PS Usage

15-25 PS characteristics

Note: For information on PS usage, see SNA LU-LU Session Types.

End of PS Usage Field

26-k Cryptography Options

26 bits 0-1, private cryptography options:
- 00 no private cryptography supported
- 01 private cryptography supported: the session cryptography key and cryptography protocols are privately supplied by the end user

bits 2-3, session-level cryptography options:
- 00 no session-level cryptography supported
- 01 session-level selective cryptography supported; all cryptography key management is supported by SSCP.SVC_MGR and LU.SVC_MGR; exchange (via +RSP(BIND)) and verification (via CRV) of the cryptography session-seed value is supported by the LU.SVC_MGRs for the session; all FMD requests carrying ED are enciphered/deciphered by the CPMGRs
- 10 reserved
- 11 session-level mandatory cryptography supported; same as session-level selective cryptography except all FMD requests are enciphered/deciphered by the CPMGRs

bits 4-7, session-level cryptography options field length:
- X'0' no session-level cryptography specified; following 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 (only value defined)

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 (only value defined)

2 ;-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--see Note, below, concerning the BIND RU length

k+2-m Primary LU network name or, if the secondary LU issued the INITIATE(-SELF or -OTHER), the uninterpreted name as carried in that RU (and also in CDINIT for a cross-domain session)

m+1 Length of user data (X'00' = no user data field present)--see Note, below, concerning the BIND RU length

m+2-n User data

m+2 User data key
X'00' structured subfields follow
-X'00' first byte of unstructured user data
Note: Individual structured subfields may be omitted entirely. When present, they appear in ascending field number order.

- For unstructured user data

m+3-n Remainder of unstructured user data

- For structured user data

m+3-n Structured subfields (For detailed definitions, see the structured user data section on page E-129.)

n+1 Length of user request correlation (URC) field
Note: X'00' = no URC present

n+2-p URC: end user defined identifier (present only if carried in INIT from SLU)

p+1 Length of secondary LU network name--see Note, below, concerning the BIND RU length
Note: X'00' = no secondary LU name present

p+2-r Secondary LU network name (present only in negotiable BIND)

Note: The length of the BIND RU cannot exceed 256 bytes, lest a negative response be returned.

	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)

```
DCL 1 BINDF_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2    */
      2 SENSE_DATA                      BIT(32), /*  3-6    */
      2 REASON                           BIT(8), /*   7     */
      2 SESSION_KEY                      BIT(8), /*   9     */
                                        /* See page E-127 */
      2 SESSION_KEY_CONTENT            CHAR(*); /*  10-m   */
```

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	Session key X'06' uninterpreted name pair X'07' network address pair
9-m	Session Key Content
	• For session key X'06': uninterpreted name pair
9	Type: X'F3' logical unit
10	Length, in binary, of symbolic name of PLU
11-k	Symbolic name in EBCDIC characters
k+1	Type: X'F3' logical unit
k+2	Length, in binary, of symbolic name of SLU
k+3-m	Symbolic name, in EBCDIC characters
	• For session key X'07': network address pair
9-10	Network address of PLU
11-12(=m)	Network address of SLU

BIS; LU-->LU, Norm; DFC (BRACKET INITIATION STOPPED)

```
DCL 1 BIS_RQ                    BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                         BIT(8); /*   0    */
```

0	X'70' request code

CANCEL; LU-->LU, Norm; DFC (CANCEL)

```
DCL 1 CANCEL_RQ                 BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                         BIT(8); /*   0    */
```

0	X'83' request code

CDCINIT; SSCP-->SSCP, Norm; FMD NS(s) (CROSS-DOMAIN CONTROL INITIATE)

```
DCL 1 CDCINIT_RQ                    BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                          BIT(24), /*  0-2   */
      2 FORMAT                              BIT(8), /*   3    */
      2 RESERVED                            BIT(8), /*   4    */
      2 PCID                               CHAR(8), /*  5-12  */
      2 PLU_NETWORK_ADDRESS                BIT(16), /* 13-14  */
      2 SLU_NETWORK_ADDRESS                BIT(16), /* 15-16  */
      2 BIND_IMAGE_LENGTH                  BIT(16), /* 17-18  */
      2 BIND_IMAGE
              CHAR(REFER(BIND_IMAGE_LENGTH)), /*  19-n  */
      2 SNA_DEV_CHAR_LENGTH                BIT(16), /*n+1-n+2 */
      2 SNA_DEV_CHAR
            CHAR(REFER(SNA_DEV_CHAR_LENGTH)), /* n+3-p  */
      2 CRYPTO_SESS_KEY_LENGTH              BIT(8), /*  p+1   */
      2 CRYPTO_SESS_KEY
          CHAR(REFER(CRYPTO_SESS_KEY_LENGTH)); /* p+2-q  */
```

0-2	X'81864B' NS header
3	Format bits 0-3, 0000 Format 0 (only value defined) bits 4-7, reserved
4	Reserved
5-12	PCID
5-6	The network address of SSCP(ILU)
7-12	A unique 6-byte value, generated by the SSCP(ILU), that is retained and used in all cross-domain requests dealing with the same procedure until it is completed. The SSCP(ILU) maintains correlation between PCID and the URC, if one has been provided by the INIT-SELF or INIT-OTHER request.
13-14	Network address of PLU
15-16	Network address of SLU
17-18	Length, in binary, of BIND image
19-n	BIND image: bytes 1-p of the BIND RU (see BIND format description), i.e., through the URC field Notes on BIND image: • If the length of the URC field is zero, then the length field itself is excluded from the BIND image. • For SLUs not in the sending SSCP's PU_T5 node, the session cryptography key is enciphered under the SLU master cryptography key; for SLUs in the PU_T5 node, the sending SSCP enciphers the session cryptography key under a dummy SLU master cryptography key.
n+1-n+2	Length, in binary, of LU or non-SNA device characteristics field and format--i.e., bytes n+3 - p (X'00' = no characteristics/format field)
n+3	LU or non-SNA device characteristics format: X'01' Format 1: access method unique device characteristics (only value defined)
n+4-p	LU or non-SNA device specifications (See CINIT for

the format of this field.)

p+1 Length, in binary, of session cryptography key
Note: X'00' = no Session Cryptography Key field is present

p+2-q Session cryptography key for primary: the session cryptography key, enciphered under the cross-domain cryptography key defined for the SSCP(SLU) to SSCP(PLU) direction (a different cross-domain cryptography key is defined for the opposite direction) and using a seed value of 0

CDINIT; SSCP-->SSCP, Norm; FMD NS(s) (CROSS-DOMAIN INITIATE)

```
DCL 1 CDINIT_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 FORMAT                            BIT(8), /*   3    */
      2 FORMAT_DATA                     CHAR(*); /* 4-End  */


DCL 1 CDINIT_RQ_FMT0_2
            BASED(ADDR(CDINIT_RQ.FORMAT_DATA)), /* Byte(s)*/
      2 TYPE                              BIT(2), /*   4    */
      2 RESERVED                          BIT(4),
      2 DLU_PRI_OR_SEC                    BIT(1),
      2 RESERVED                          BIT(1),
      2 DLU_QUEUING_CONDITIONS            BIT(8), /*   5    */
      2 OLU_STATUS                        BIT(8), /*   6    */
      2 PCID                            CHAR(8), /*  7-14  */
      2 OLU_ADDRESS                      BIT(16), /* 15-16  */
      2 RESERVED                         BIT(16), /* 17-18  */
      2 INITIATE_ORIGIN                   BIT(8), /*   19   */
      2 NOTIFY_SPECIFICATIONS             BIT(8), /*   20   */
      2 MODE_NAME                       CHAR(8), /* 21-28  */
      2 DLU_TYPE                          BIT(8), /*   29   */
      2 DLU_NTWK_NAME_LENGTH              BIT(8), /*   30   */
      2 DLU_NTWK_NAME
            CHAR(REFER(DLU_NTWK_NAME_LENGTH)), /*  31-m  */
      2 REQUESTER_ID_LENGTH               BIT(8), /*  m+1   */
      2 REQUESTER_ID
             CHAR(REFER(REQUESTER_ID_LENGTH)), /* m+2-n  */
      2 PASSWORD_LENGTH                   BIT(8), /*  n+1   */
      2 PASSWORD  CHAR(REFER(PASSWORD_LENGTH)), /* n+2-p  */
      2 USER_DATA_LENGTH                  BIT(8), /*  p+1   */
      2 USER_DATA
                CHAR(REFER(USER_DATA_LENGTH)), /* p+2-q  */
      2 OLU_TYPE                          BIT(8), /*  q+1   */
      2 OLU_NTWK_NAME_LENGTH              BIT(8), /*  q+2   */
      2 OLU_NTWK_NAME
            CHAR(REFER(OLU_NTWK_NAME_LENGTH)), /* q+3-r  */
      2 DLU_UNINTRP_NAME_TYPE             BIT(8), /*  r+1   */
      2 DLU_UNINTRP_NAME_LENGTH           BIT(8), /*  r+2   */
      2 DLU_UNINTRP_NAME
          CHAR(REFER(DLU_UNINTRP_NAME_LENGTH)), /* r+3-s  */
      2 COS_NAME_INITIALIZATION           BIT(8), /*  s+1   */
```

```
      2 COS_NAME                          CHAR(8); /*s+2-s+9 */


DCL 1 CDINIT_RQ_FMT1
            BASED(ADDR(CDINIT_RQ.FORMAT_DATA)), /* Byte(s)*/
      2 TYPE                             BIT(8), /*   4     */
      2 RESERVED                         BIT(5), /*   5     */
      2 QUEUING_STATUS                   BIT(2),
      2 RESERVED                         BIT(1),
      2 LU_STATUS                        BIT(8), /*   6     */
      2 PCID                            CHAR(8), /*  7-14   */
      2 LU1_ADDRESS                     BIT(16), /* 15-16   */
      2 LU2_ADDRESS                     BIT(16); /* 17-18   */

0-2       X'818641' NS header
3         Format
          bits 0-3,  0000  Format 0:  used when Type = I,
                           I/Q, or Q; bytes 17-18 are
                           reserved and no COS fields are
                           specified for Format 0; Format 0
                           includes bytes 0 through s
                     0001  Format 1:  used when Type = DQ
                           and specifies a subset of the
                           parameters; Format 1 includes
                           bytes 0 through 18
                     0010  Format 2:  specifies COS fields
                           and an additional OLU status
                           (byte 6, bit 5) in addition to
                           the parameters in Format 0;
                           Format 2 includes bytes 0 through
                           s+9
          bits 4-7, reserved
4-(s|s+9) Formats 0 and 2 Continue (See Format 1
          continuation below.)
4         Type:
          bits 0-1,  00  reserved
                     01  initiate only (I)
                     10  queue only (Q)
                     11  initiate or queue (I/Q)
          bits 2-5, reserved
          bit 6,  0  DLU is PLU
                  1  OLU is PLU
          bit 7, reserved
5         Queuing Conditions For DLU
          bit 0,  0  do not queue if session limit exceeded
                  1  queue if session limit exceeded
          bit 1,  0  do not queue if DLU is not currently
                     able to comply with the PLU/SLU
                     specification (as given in byte 4, bit
                     6)
                  1  queue if DLU is not currently able to
                     comply with the PLU/SLU specification
          bit 2,  0  do not queue if CDINIT loses contention
                  1  queue if CDINIT loses contention
```

bit 3, 0 do not queue if no SSCP(DLU)-DLU path
1 queue if no SSCP(DLU)-DLU path
bit 4, reserved
bits 5-6, queuing position/service
00 put this request on the bottom of the queue (this request is put at the bottom of the queue and serviced last)
01 enqueue this request FIFO
10 enqueue this request LIFO
11 reserved
bit 7, 0 do not queue for recovery retry
1 queue for recovery retry (The element will be maintained on the recovery retry queue even after the activation of the session so that the session can be retried in the event of a session failure.)

Note: Queuing will not be done if the DLU is unknown, or the domain of the DLU is in takedown status.

6 OLU status
bit 0, reserved
bit 1, 0 LU is not available
1 LU is available
bits 2-3, (used if LU is not available; otherwise, reserved)
00 LU session limit exceeded
01 reserved
10 LU is not currently able to comply with the PLU/SLU specification
11 reserved
bit 4, 0 existing SSCP to LU path
1 no existing SSCP to LU path (connectivity is lost)
bit 5, (reserved in format 0)
0 UNBIND and SESSEND cannot be sent by the LU or by its boundary function (if any)
1 UNBIND and SESSEND may be sent by the LU or by its boundary function (if any)
bits 6-7, 01 OLU is PLU
10 OLU is SLU

7-14 PCID
7-8 The network address of SSCP(ILU)
9-14 A unique 6-byte value, generated by the SSCP (ILU), that is retained and used in all cross-domain requests dealing with the same procedure until it is completed
15-16 Network address of OLU
17-18 Reserved
19 INITIATE origin:
bit 0, 0 OLU is origin
1 third party is origin

	bits 1-2, reserved bit 3, 0 network user is the initiator 1 network manager is the initiator bits 4-7, reserved
20	NOTIFY specification: bits 0-1, 00 do not send NOTIFY to LUs in session with DLU 01 send NOTIFY to all LUs in session with DLU 10 send NOTIFY to all LUs in session with DLU only if the CDINIT request is queued 11 reserved bits 2-7, reserved
21-28	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 to be used by the SSCP(PLU) to build the CINIT request
29-m	Network Name of DLU
29	Type: X'F3' logical unit
30	Length, in binary, of symbolic name
31-m	Symbolic name, in EBCDIC characters
m+1-n	Requester ID
m+1	Length, in binary, of requester ID Note: X'00' = no requester ID is present
m+2-n	Requester ID: the ID, in EBCDIC characters, of the end user initiating the request (May be used to establish the authority of the end user to access a particular resource.)
n+1-p	Password
n+1	Length, in binary, of password Note: X'00' = no password is present
n+2-p	Password used to verify the identity of the end user
p+1-q	User Field
p+1	Length, in binary, of user data Note: X'00' = no user data is present
p+2-q	User data: user-specific data that is passed to the primary LU on the CINIT request
p+2	User data key X'00' structured subfields follow ¬X'00' first byte of unstructured user data Note: Individual structured subfields may be omitted entirely. When present, they appear in ascending field number order.
	• For unstructured user data
p+3-q	Remainder of unstructured user data
	• For structured user data
p+3-q	Structured subfields (For detailed definitions, see the structured user data section on page E-129.)
q+1-r	Network Name of OLU

```
q+1         Type: X'F3' logical unit
q+2         Length, in binary, of symbolic name
q+3-r       Symbolic name in EBCDIC characters
r+1-s       Uninterpreted Name of DLU
r+1         Type: X'F3' logical unit
r+2         Length, in binary, of DLU name
            Note:  X'00' = no uninterpreted name is present.
r+3-s       EBCDIC character string; when present, this name
            is obtained from the preceding INIT-SELF or
            INIT-OTHER (when ILU=OLU)
Note:  End of Format 0; Format 2 continues below.
s+1         COS name initialization indicators:
            bit 0,  0  COS name not received from ILU (see
                       bits 1-2)
                    1  COS name received from ILU
            bits 1-2, (reserved if byte s+1, bit 0 = 1)
                       01  SSCP(DLU) is to initialize COS name
                           (DLU is SLU)
                       10  SSCP(OLU) has initialized COS name
                           (OLU is SLU)
            bits 3-7, reserved
s+2-s+9     COS name (this field reserved if byte s+1, bits
            1-2 = 01):  symbolic name of class of service in
            EBCDIC characters
4-18        Format 1
4           Type
            bits 0-1,  00  dequeue (DQ)
            bits 2-3,  00  leave on queue if dequeue retry is
                           unsuccessful
                       01  remove from queue if dequeue retry
                           is unsuccessful
                       10  do not retry--remove from queue
                       11  reserved
            bit 4, reserved
            bits 5-6,  00  LU2 is PLU
                       01  LU2 is SLU
                       10  reserved
                       11  reserved
            bit 7, reserved
5           Queuing Status (For LU associated with SSCP
            sending CDINIT(DQ))
            bits 0-4, reserved
            bits 5-6,  00  request on bottom of queue
                       01  enqueued request FIFO
                       10  enqueued request LIFO
                       11  reserved
            bit 7, reserved
6           LU Status (For LU associated with SSCP sending
            CDINIT(DQ))
            bit 0, reserved
            bit 1,  0  LU is unavailable
                    1  LU is available
            bits 2-3, (if LU is unavailable)
                       00  LU session limit exceeded
```

```
                              01  reserved
                              10  LU is not currently  able to comply
                                  with the PLU/SLU specification
                              11  reserved
                    bit 4,   0  existing SSCP to LU path
                             1  no existing SSCP to LU path
                    bit 5, reserved
                    bits 6-7,  01  LU is PLU
                               10  LU is SLU
```

7-14 PCID

7-8 The network address of SSCP(ILU)

9-14 A unique 6-byte value, generated by the SSCP(ILU), that is retained and used in all cross-domain requests dealing with the same procedure until it is completed. (This PCID must be the same as in the original CDINIT request.)

15-16 Network address of LU1

17-18 Network address of LU2

CDSESSEND; SSCP(PLU)<-->SSCP(SLU), Norm; FMD NS(s) (CROSS-DOMAIN SESSION ENDED)

```
DCL 1 CDSESSEND_RQ                BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                         BIT(24), /*  0-2   */
      2 PCID                              CHAR(8), /*  3-10  */
      2 FORMAT                             BIT(8), /*   11   */
      2 FORMAT_DATA                       CHAR(*); /*  12-n  */


DCL 1 CDSESSEND_FMT0_RQ
         BASED(ADDR(CDSESSEND_RQ.FORMAT_DATA)), /* Byte(s)*/
      2 SESSION_KEY                        BIT(8), /*   12   */
                                           /* See page E-127 */
      2 SESSION_KEY_CONTENT               CHAR(*); /*  13-n  */


DCL 1 CDSESSEND_FMT2_RQ
         BASED(ADDR(CDSESSEND_RQ.FORMAT_DATA)), /* Byte(s)*/
      2 CAUSE                              BIT(8), /*   12   */
      2 ACTION                             BIT(8), /*   13   */
      2 RESERVED                          BIT(16), /* 14-15  */
      2 SESSION_KEY                        BIT(8), /*   16   */
                                           /* See page E-127 */
      2 SESSION_KEY_CONTENT               CHAR(*); /*  17-n  */
```

0-2 X'818648' NS header

3-10 PCID

3-4 Network address of SSCP(TLU)

Note: A network address value of 0 indicates that no PCID is present in bytes 5 through 10; bytes 5-10 are reserved when bytes 3-4 are 0.

5-10 A unique 6-byte value, generated by the SSCP(TLU), that is retained and used in all cross-domain requests dealing with the same procedure until it

	is completed.
11	bits 0-3, format: 0000 Format 0 0010 Format 2 bits 4-7, reserved
12-n	Format 0
12	Session key X'06' network name pair X'07' network address pair
13-n	Session Key Content
	• For session key X'06': network name pair
13	Type: X'F3' logical unit
14	Length, in binary, of symbolic name of PLU
15-m	Symbolic name in EBCDIC characters
m+1	Type: X'F3' logical unit
m+2	Length, in binary, of symbolic name of SLU
m+3-n	Symbolic name in EBCDIC characters
	• For session key X'07': network address pair
13-14	Network address of PLU
15-16(=n)	Network address of SLU
12-n	Format 2
12	Cause: indicates the reason for deactivation of the identified LU-LU session X'01' normal deactivation X'02' BIND forthcoming; retain the node resources allocated to this session, if possible X'04' restart mismatch; synch point records do not match; operator intervention is needed before the session can be activated X'05' LU not authorized: the secondary half-session has failed to supply an acceptable password or other authorization information in the User Data field 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 had to be 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' PLU failure: the identified (LU,LU) session had to be deactivated because of an abnormal termination of the PLU

13 Action (reserved for cause codes X'01' through X'06'):

X'01' normal, no resultant automatic action
X'02' primary half-session will restart
X'03' secondary half-session will restart

14-15 Reserved

16 Session key:

X'06' network name pair
X'07' network address pair

17-n Session Key Content

- For session key X'06': network name pair

17 Type: X'F3' logical unit
18 Length, in binary, of symbolic name of PLU
19-m Symbolic name in EBCDIC characters
m+1 Type: X'F3' logical unit
m+2 Length, in binary, of symbolic name of SLU
m+3-n Symbolic name in EBCDIC characters

- For session key X'07': network address pair

17-18 Network address of PLU
19-20(=n) Network address of SLU

CDSESSSF; SSCP(PLU)-->SSCP(SLU), Norm; FMD NS(s) (CROSS-DOMAIN SESSION SETUP FAILURE)

```
DCL 1 CDSESSSF_RQ                BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 PCID                             CHAR(8), /*  3-10  */
      2 SENSE_DATA                       BIT(32), /* 11-14  */
      2 REASON                            BIT(8), /*   15   */
      2 SESSION_KEY                       BIT(8), /*   16   */
                                          /* See page E-127 */
      2 SESSION_KEY_CONTENT              CHAR(*); /*  17-n  */
```

0-2 X'818645' NS header

3-10 PCID

3-4 The network address of SSCP (ILU)

5-10 A unique 6-byte value, generated by the SSCP(ILU), that is retained and used in all cross-domain requests dealing with the same procedure until it is completed

11-14 Sense data

15 Reason

bit 0, 1 CINIT error in reaching PLU
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
          16    Session key
                 X'06'  network name pair
                 X'07'  network address pair
          17-n  Session Key Content
              • For session key X'06':  network name pair
          17    Type: X'F3' logical unit
          18    Length, in binary, of symbolic name of PLU
          19-m  Symbolic name in EBCDIC characters
          m+1   Type: X'F3' logical unit
          m+2   Length, in binary, of symbolic name of SLU
          m+3-n Symbolic name in EBCDIC characters
              • For session key X'07':  network address pair
          17-18 Network address of PLU
          19-20(=n) Network address of SLU

CDSESSST; SSCP(PLU)-->SSCP(SLU), Norm; FMD NS(s) (CROSS-DOMAIN SESSION
          STARTED)

          DCL 1 CDSESSST_RQ              BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                      BIT(24), /*  0-2   */
                2 PCID                           CHAR(8), /*  3-10  */
                2 RESERVED                        BIT(8), /*   11   */
                2 SESSION_KEY                     BIT(8), /*   12   */
                                                  /* See page E-127 */
                2 SESSION_KEY_CONTENT            CHAR(*); /*  13-n  */

          0-2   X'818646' NS header
          3-10  PCID
          3-4   The network address of SSCP(ILU)
          5-10  A unique 6-byte value, generated by the SSCP(ILU),
                which  is retained  and used  in all  cross-domain
                requests dealing with the  same procedure until it
                is completed
          11    Reserved
          12    Session key
                 X'06'  network name pair
                 X'07'  network address pair
          13-n  Session Key Content
              • For session key X'06': network name pair
          13    Type: X'F3' logical unit
          14    Length, in binary, of symbolic name of PLU
          15-m  Symbolic name in EBCDIC characters
          m+1   Type: X'F3' logical unit
          m+2   Length, in binary, of symbolic name of SLU
          m+3-n Symbolic name in EBCDIC characters
              • For session key X'07': network address pair
          13-14 Network address of PLU
          15-16(=n) Network address of SLU
```

CDSESSTF; SSCP(PLU)-->SSCP(SLU), Norm; FMD NS(s) (CROSS-DOMAIN SESSION TAKEDOWN FAILURE)

```
DCL 1 CDSESSTF_RQ                BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 PCID                             CHAR(8), /*  3-10  */
      2 SENSE_DATA                      BIT(32), /* 11-14  */
      2 REASON                           BIT(8), /*   15   */
      2 SESSION_KEY                      BIT(8), /*   16   */
                                        /* See page E-127 */
      2 SESSION_KEY_CONTENT             CHAR(*); /*  17-n  */
```

0-2 X'818647' NS header

3-10 PCID

3-4 The network address of SSCP(TLU)

Note: A network address value of 0 indicates that no PCID is present; bytes 5-10 are reserved when bytes 3-4 are 0.

5-10 A unique 6-byte value, generated by the SSCP(TLU), that is retained and used in all cross-domain requests dealing with the same procedure until it is completed

11-14 Sense data

15 Reason:
bit 0, 1 CTERM error in reaching PLU
bit 1, 1 UNBIND error in reaching SLU
bit 2, 1 takedown reject at PLU
bits 3-7, reserved

16 Session key:
X'06' network name pair
X'07' network address pair

17-n Session Key Content

- For session key X'06': network name pair

17 Type: X'F3' logical unit

18 Length, in binary, of symbolic name of PLU

19-m Symbolic name in EBCDIC characters

m+1 Type: X'F3' logical unit

m+2 Length, in binary, of symbolic name of SLU

m+3-n Symbolic name in EBCDIC characters

- For session key X'07': network address pair

17-18 Network address of PLU

19-20(=n) Network address of SLU

CDTAKED; SSCP-->SSCP, Norm; FMD NS(s) (CROSS-DOMAIN TAKEDOWN)

```
DCL 1 CDTAKED_RQ                 BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 PCID                             CHAR(8), /*  3-10  */
      2 TYPE                              BIT(8), /*   11   */
      2 REASON                            BIT(8); /*   12   */
```

0-2 X'818649' NS header

3-10 PCID

3-4 The network address of the SSCP sending the

request

5-10 A unique 6-byte value generated by the sending SSCP and retained and used in all cross-domain requests dealing with the same procedure until it is completed

11 Type:

bits 0-1, 00 active and pending-active sessions
01 active, pending-active, and queued sessions
10 queued only sessions
11 reserved

bits 2-3, 00 quiesce
01 orderly
10 forced
11 cleanup (mutual procedure)

bits 4-7, reserved

12 Reason:

bit 0, 0 network user
1 network manager

bit 1, 0 normal
1 abnormal

bits 2-7, detailed reason (dependent upon bits 0-1):

- For bits 0-1, 00 user and normal:
  bits 2-7, 000000 general category (only value defined)
- For bits 0-1, 01 user and abnormal:
  bits 2-7, 000000 general category (only value defined)
- For bits 0-1, 10 manager and normal:
  bits 2-7, 000000 general category
  000011 operator command--domain is going away
- For bits 0-1, 11 manager and abnormal:
  bits 2-7, 000000 general category
  000001 operator command
  000010 restart procedure

CDTAKEDC; SSCP-->SSCP, Norm; FMD NS(s) (CROSS-DOMAIN TAKEDOWN COMPLETE)

```
DCL 1 CDTAKEDC_RQ              BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                      BIT(24), /*  0-2   */
      2 PCID                           CHAR(8), /*  3-10  */
      2 TYPE                            BIT(8), /*   11   */
      2 STATUS                          BIT(8); /*   12   */
```

0-2 X'81864A' NS header

3-10 PCID

3-4 The network address of the SSCP that initiated the takedown procedure

5-10 A unique 6-byte value, generated by the SSCP initiating the takedown procedure, that is retained and used in all cross-domain requests

dealing with the same procedure until it is completed

11 Type:
X'01' summary (only value defined)

12 Status:
• For Type X'01': summary
X'01' all sessions successfully taken down
X'02' takedown failures occurred

CDTERM; SSCP(OLU)-->SSCP(DLU), Norm; FMD NS(s) (CROSS-DOMAIN TERMINATE)

```
DCL 1 CDTERM_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 FORMAT                            BIT(8), /*   3    */
      2 TYPE                              BIT(8), /*   4    */
      2 PCID                             CHAR(8), /*  5-12  */
      2 REASON                            BIT(8), /*   13   */
      2 RESERVED                         BIT(16), /* 14-15  */
      2 SESSION_KEY                       BIT(8), /*   16   */
                                          /* See page E-127 */
      2 SESSION_KEY_CONTENT
             CHAR(REFER(SESSION_KEY_LENGTH)), /*  17-n  */
      2 REQUESTER_ID_LENGTH               BIT(8), /*  n+1   */
      2 REQUESTER_ID
            CHAR(REFER(REQUESTER_ID_LENGTH)), /* n+2-p  */
      2 PASSWORD_LENGTH                   BIT(8), /*  p+1   */
      2 PASSWORD  CHAR(REFER(PASSWORD_LENGTH)); /* p+2-q  */
```

0-2 X'818643' NS header

3 bits 0-3, 0000 Format 0 (only value defined)
bits 4-7, reserved

4 Type:
bits 0-1, 00 request applies to active and pending-active sessions
01 request applies to active, pending-active, and queued sessions
10 request applies to queued sessions only
11 reserved

bit 2, reserved if byte 4, bit 7 = 1; otherwise:
0 forced termination, session to be deactivated immediately and unconditionally
1 orderly termination, permitting an end-of-session procedure to be executed at the PLU before the session is deactivated

bit 3, 0 do not send DACTLU to DLU; another session initiation request will be sent for DLU
1 send DACTLU to DLU when appropriate; no further session initiation request will be sent (from this sender) for DLU

bits 4-6, reserved
bit 7, 0 orderly or forced (see byte 4, bit 2)
1 cleanup

5-12 PCID

5-6 The network address of the SSCP(TLU)

7-12 A unique 6-byte value, generated by the SSCP(TLU), that is retained and used in all cross-domain requests dealing with the same procedure until it is completed

13 Reason:
bit 0, 0 network user
1 network manager
bit 1, 0 normal
1 abnormal
bits 2-7, detailed reason (dependent upon bits 0-1):

- For bits 0-1, 00 user and normal:
  bits 2-7, 000000 general category
  000001 self, OLU=PLU
  000010 self, OLU=SLU
  000011 other
- For bits 0-1, 01 user and abnormal:
  bits 2-7, 000000 general category
- For bits 0-1, 10 manager and normal:
  bits 2-7, 000000 general category
  000001 operator command--session
  000010 operator command--LU
  000011 operator command--domain
- For bits 0-1, 11 manager and abnormal:
  bits 2-7, 000000 general category
  000001 operator command
  000010 restart procedure
  000011 preempt procedure
  000100 unrecoverable path error
  000101 unrecoverable destination error

14-15 Reserved

16 Session key:
X'05' PCID
X'06' network name pair
X'07' network address pair
X'08' network address-network name

17-n Session Key Content

- For session key X'05': PCID

17-18 Network address of the SSCP(ILU)

19-24(=n) A unique six-byte value, generated by the SSCP(ILU), which is retained and used in all cross-domain requests dealing with the same procedure until it is completed
Note: This PCID is different from the one in bytes 5-12, which is generated by the SSCP(TLU).

- For session key X'06': network name pair

17 Type: X'F3' logical unit

18 Length, in binary, of symbolic name of OLU

19-m Symbolic name in EBCDIC characters

```
m+1         Type: X'F3' logical unit
m+2         Length, in binary, of symbolic name of DLU
m+3-n       Symbolic name in EBCDIC characters
          • For session key X'07':  network address pair
17-18       Network address of PLU
19-20(=n)   Network address of SLU
          • For  session key  X'08':  network  address-network
            name
17-18       Network address of OLU
19          Type: X'F3' logical unit
20          Length, in binary, of symbolic name of DLU
21-n        Symbolic name in EBCDIC characters
n+1-p       Requester ID
n+1         Length, in binary, of requester ID
            Note:  X'00' = no requester ID
n+2-p       Requester ID: the ID, in EBCDIC characters, of the
            end user initiating the request
p+1-q       Password
p+1         Length, in binary, of password
            Note:  X'00' = no password is present
p+2-q       Password used  to verify the  identity of  the end
            user
```

CHASE; LU-->LU, Norm; DFC (CHASE)

```
DCL 1 CHASE_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                           BIT(8); /*   0    */
```

```
0         X'84' request code
```

CINIT; SSCP-->PLU, Norm; FMD NS(s) (CONTROL INITIATE)

```
DCL 1 CINIT_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                         BIT(24), /*  0-2   */
      2 FORMAT                             BIT(8), /*   3    */
      2 INITIATE_ORIGIN                    BIT(8), /*   4    */
      2 SESSION_KEY                        BIT(8), /*   5    */
      2 PLU_ADDRESS                       BIT(16), /*  6-7   */
      2 SLU_ADDRESS                       BIT(16), /*  8-9   */
      2 BIND_IMAGE_LENGTH                 BIT(16), /* 10-11  */
      2 BIND_IMAGE
              CHAR(REFER(BIND_IMAGE_LENGTH)), /*  12-m  */
      2 SLU_TYPE                           BIT(8), /*  m+1   */
      2 SLU_NTWK_NAME_LENGTH               BIT(8), /*  m+2   */
      2 SLU_NTWK_NAME
           CHAR(REFER(SLU_NTWK_NAME_LENGTH)), /* m+3-n  */
      2 REQUESTER_ID_LENGTH                BIT(8), /*  n+1   */
      2 REQUESTER_ID
            CHAR(REFER(REQUESTER_ID_LENGTH)), /* n+2-p  */
      2 PASSWORD_LENGTH                    BIT(8), /*  p+1   */
      2 PASSWORD  CHAR(REFER(PASSWORD_LENGTH)), /* p+2-q  */
      2 USER_DATA_LENGTH                   BIT(8), /*  q+1   */
      2 USER_DATA
              CHAR(REFER(USER_DATA_LENGTH)), /* q+2-r  */
```

```
        2 SNA_DEV_CHAR_LENGTH              BIT(16), /*r+1-r+2 */
        2 SNA_DEV_CHAR
              CHAR(REFER(SNA_DEV_CHAR_LENGTH)), /* r+3-s  */
        2 CRYPTO_SESS_KEY_LENGTH            BIT(8), /*  s+1   */
        2 CRYPTO_SESS_KEY
            CHAR(REFER(CRYPTO_SESS_KEY_LENGTH)), /* s+2-t  */
        2 CONTROL_VECTORS                  CHAR(*); /* t+1-u  */


    DCL 1 SNA_DEVICE_CHARACTERISTICS         BASED, /* Byte(s)*/
          2 DEVICE_CHAR_FORMAT               BIT(8), /*  r+3   */
          2 SCHEDULING_INFO                  BIT(8), /*  r+4   */
          2 DEVICE_TYPE                      BIT(8), /*  r+5   */
          2 MODEL_INFORMATION                BIT(8), /*  r+6   */
          2 FEATURE_INFORMATION              BIT(8), /*  r+7   */
          2 PHYSICAL_ADDRESS                 BIT(8), /*  r+8   */
          2 MISC_FLAGS                       BIT(8), /*  r+9   */
          2 DATA_STREAM                      BIT(8), /*  r+10  */
          2 RESERVED                         BIT(8), /*  r+11  */
          2 SCREEN_SIZE                     BIT(40), /*r+12-16 */
          2 WORK_AREA_FORMAT                 BIT(8), /*  r+17  */
          2 WORK_AREA                       CHAR(*); /* r+18-s */
```

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 (which ends with the Session Cryptography Key field).
bits 4-7, reserved

4 INITIATE Origin:
bit 0, 0 ILU is OLU
1 ILU is not OLU
bit 1, reserved
bit 2, 0 SLU is OLU
1 PLU is OLU
bit 3, 0 network user is the initiator
1 network manager is the initiator
bits 4-5, reserved
bit 6, 0 no recovery retry
1 recovery retry to be used
bit 7, reserved

5 Session key:
X'07' network address pair

6-7 Network address of PLU

8-9 Network address of SLU

10-11 Length 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: If the length of the URC field is 0, the Length field itself is excluded from the BIND image.

m+1-n Name of SLU
m+1 Type: X'F3' logical unit
m+2 Length, in binary, of symbolic name
m+3-n Symbolic name, in EBCDIC characters
n+1-p Requester ID
n+1 Length, in binary, of requester ID
Note: X'00' = no requester ID
n+2-p Requester ID: the ID, in EBCDIC characters, of the end user initiating the session activation request (May be used to establish the authority of the end user to access a particular resource.)
p+1-q Password
p+1 Length, in binary, of password
Note: X'00' = no password is present
p+2-q Password used to verify the identity of the end user
q+1-r User Field (from INITIATE RU)
q+1 Length, in binary, of user data
Note: X'00' = no user data is present
q+2-r User data: user-specific data
q+2 User data key
X'00' structured subfields follow
-X'00' first byte of unstructured user data
Note: Individual structured subfields may be omitted entirely. When present, they appear in ascending field number order.
• For unstructured user data
q+3-r Remainder of unstructured user data
• For structured user data
q+3-r Structured subfields (For detailed definitions, see the structured user data section on page E-129.)
r+1-s LU or Non-SNA Device Specifications
r+1-r+2 Length of characteristics field, including both format and characteristics fields--i.e., bytes r+4 - s
Note: X'0000' = no Format and no Characteristics fields are present.
r+3 Characteristics format:
X'01' device characteristics (only value defined)
r+4-s LU or Non-SNA Device Characteristics
• Format X'01': (This format represents an access-method-unique LU/device characteristics definition. For more specific information refer to access method implementation documentation.)
r+4 Scheduling information:
X'80' input device
X'40' output device
X'20' conversational mode
X'10' reserved
x'08' start print sensitive
X'04' reserved
X'02' additional information provided (always

```
                  on)
                  X'01'  specific poll=on; general poll=off
        r+5       Device type:
                  X'00'  undefined device type
                  X'04'  2741
                  X'08'  WTTY
                  X'10'  115A
                  X'20'  TWX (33-35)
                  X'30'  83B3
                  X'40'  2740
                  X'80'  1050
                  X'90'  2780
                  X'19'  3277
                  X'1A'  3284
                  X'1B'  3286/3288
                  X'1C'  3275
                  X'91'  3780
                  X'6D'  SNA logical unit
        r+6       Model information:
                  X'00'  Model 1
                  X'01'  Model 2
        r+7       Feature information:
                  bits 0-1,  00  SLDC
                             01  start/stop
                             10  BSC
                             11  reserved
                  bits 2-7,  X'20'  XMIT interrupt feature
                             X'10'  SWITCHED LINE =  ON; LEASED LINE
                                    = OFF
                             X'08'  attention
                             X'04'  checking
                             X'02'  station control
                             X'01'  selector pen
        r+8       Physical device address
        r+9       Miscellaneous flags:
                  X'80'  SNA    compatible   application    program
                         interface (always on)
                  X'40'  non-SNA   application  program   interface
                         (always off)
                  X'20'  buffered
                  X'10'  continue mode
                  X'08'  contention mode
                  X'04'  inhibit mode (text timeout)
                  X'02'  end-to-end control
                  X'01'  3270  extended data  stream requiring  BSC
                         transparency
        r+10      Device data stream  compatibility characteristics:
                  (This field is used in conjunction with the Device
                  Type field, r+5, when that  field is set to X'6D':
                  SNA logical unit; otherwise, it is reserved.)
                  X'00'  no  data  stream  characteristics  defined
                         here
                  X'04'  2741
                  X'08'  WTTY
```

```
                    X'10'  115A
                    X'20'  TWX (33-35)
                    X'30'  83B3
                    X'40'  2740
                    X'80'  1050
                    X'90'  2780
                    X'19'  3277
                    X'1A'  3284
                    X'1B'  3286/3288
                    X'1C'  3275
                    X'91'  3780
                    X'A0'-X'FF'  available for installation-defined
                                 use
          r+11      Reserved
          r+12-r+16 Screen size (see the PS Usage field in the BIND RU
                    for format)
          r+17-s    Work Area (This field is optional—if not present,
                    s = r+16.)
          r+17      Work area format:
                     X'00'  unformatted
                     X'01'  TCAM format
          r+18-s    Work area excluding format
          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
          Note:  End of base RU
          t+1-u     Control vector, as described in the section,
                    "Control Vectors and Control lists," later in this
                    appendix
                    Note:  The following vector key is used in CINIT:
                     X'0D'  Mode/Class of Service/Virtual Route List

CLEANUP; SSCP-->SLU, Norm; FMD NS(s) (CLEAN UP SESSION)

          DCL 1 CLEANUP_RQ                BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                       BIT(24), /*  0-2   */
                2 FORMAT                           BIT(4), /*   3    */
                2 RESERVED                        BIT(12), /*  3-4   */
                2 REASON                           BIT(8), /*   5    */
                2 SESSION_KEY                      BIT(8), /*   6    */
                                                  /* See page E-127 */
                2 SESSION_KEY_CONTENT             CHAR(*); /*  7-n   */

          0-2       X'810629' NS header
          3         bits 0-3, 0000 Format 0 (only value defined)
                    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, detailed  reason  (dependent  upon  bits
                        0-1):
            • For bits 0-1, 00 user and normal
              bits 2-7,  000000  general category
                         000001  self, OLU=PLU
                         000010  self, OLU=SLU
                         000011  other
            • For bits 0-1, 01 user and abnormal
              bits 2-7,  000000  general  category  (only  value
                                 defined)
            • For bits 0-1, 10 manager and normal
              bits 2-7,  000000  general category
                         000001  operator command--clean  up the
                                 session
                         000010  operator command--clean  up all
                                 sessions for LU
                         000011  operator command--clean  up all
                                 LU-LU sessions  for LUs  in the
                                 domain
            • For bits 0-1, 11 manager and abnormal
              bits 2-7,  000000  general category
                         000001  operator command
                         000010  restart procedure
                         000011  preempt procedure
                         000100  unrecoverable path error
                         000101  unrecoverable destination error
6           Session key
             X'06'  uninterpreted name pair
             X'07'  network address pair
7-n         Session Key Content
          • For session key X'06': uninterpreted name pair
7           Type: X'F3' logical unit
8           Length, in binary, of PLU name
9-m         EBCDIC character string
m+1         Type: X'F3' logical unit
m+2         Length, in binary, of SLU name
m+3-n       EBCDIC character string
          • For session key X'07': network address pair
7-8         Network address of PLU
9-10(=n)    Network address of SLU

CLEAR; PLU-->SLU, SSCP-->SSCP, Exp; SC (CLEAR)

          DCL 1 CLEAR_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
                2 RQ_CODE                          BIT(8); /*   0    */

          0         X'A1' request code
```

CONNOUT; SSCP-->PU_T4|5, PUCP-->PU, Norm; FMD NS(c) (CONNECT OUT)

```
DCL 1 CONNOUT_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                         BIT(24), /*  0-2   */
      2 LINK_ADDRESS                      BIT(16), /*  3-4   */
      2 LOCAL_ADDRESS                      BIT(8), /*   5    */
      2 CONNECT_OUT_TYPE                   BIT(1), /*   6    */
      2 CONNECT_OUT_FEATURE                BIT(2),
      2 RESERVED                           BIT(5),
      2 RETRY_LIMIT                        BIT(8), /*   7    */
      2 NUMBER_OF_DIAL_DIGITS              BIT(8), /*   8    */
      2 DIAL_DIGITS
          CHAR(REFER(NUMBER_OF_DIAL_DIGITS)); /*  9-n   */
```

0-2 X'01020E' NS header

3-4 Network address of link

5 SDLC link station identifier

6 bit 0, type: 0 (only value defined)
bits 1-2, connect-out feature:
- 00 automatic connect out (dial digits are provided)
- 01 reserved
- 10 manual connect out (no dial digits are provided); this bit setting does not apply to CCITT X.21 connections
- 11 CCITT X.21 direct connect out (no dial digits are provided)

bits 3-7, reserved
Note: Bytes 7-n are not included on manual connect calls (bits 1-2 = 10).

7 Retry limit: number of times the connect-out procedure is to be retried

8 Number of dial digits (zero for X.21 direct connect out)

9-n Dial digits: decimal EBCDIC characters plus end-of-numbers (X'FC', or X'4E' for X.21) and separator (X'FA' or X'FD') characters, where used

CONTACT; SSCP-->PU_T4|5, PUCP-->PU, Norm; FMD NS(c) (CONTACT)

```
DCL 1 CONTACT_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                         BIT(24), /*  0-2   */
      2 ALS_ADDRESS                       BIT(16); /*  3-4   */
```

0-2 X'010201' NS header

3-4 Network address of adjacent link station of the node to be contacted

CONTACTED; PU_T4|5-->SSCP, PU-->PUCP, Norm; FMD NS(c) (CONTACTED)

```
DCL 1 CONTACTED_RQ              BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 ALS_ADDRESS                     BIT(16), /*  3-4   */
      2 STATUS                           BIT(8), /*   5    */
      2 STATUS_DATA                     CHAR(*); /*  6-p   */
```

0-2	X'010280' NS header
3-4	Network address of adjacent link station of the node being contacted
5	Status of adjacent link station or node associated with adjacent link station: X'01' loaded (no field follows) X'02' load required (no field follows) X'03' error on CONTACT (no field follows) X'04' loaded (additional field, bytes 6-p, follows) X'05' exchanged parameters in XID Format 2 I-field not compatible (additional field, bytes 6-p, follows) X'07' no routing capability to adjacent node (additional field, bytes 6-p, follows) X'08' incompatible parameters in XID Format 2 I-field for addition of link station to currently active TG (additional field, bytes 6-p, follows)
6-p	Additional fields for status bytes X'04', X'05', X'07', and X'08'
	• For status byte X'04'
6	Resolved TG number
7-10	Adjacent node subarea address (right-justified with leading zeros)
11-18	IPL load module ID received from the adjacent node: an eight-character EBCDIC symbolic name of the IPL load module currently operating in the adjacent node Note: X'40...40' = no information conveyed.
	• For status bytes X'05', X'07', and X'08'
6	Length, in binary, of XID Format 2 I-field received
7-n	XID Format 2 I-field received (See the later section, "DLC XID Information-Field Format," for format details.)
n+1	Length, in binary, of XID Format 2 I-field sent
n+2-p	XID Format 2 I-field sent (See the later section, "DLC XID Information-Field Format," for format details.)

CRV; PLU-->SLU, Exp; SC (CRYPTOGRAPHY VERIFICATION)

```
DCL 1 CRV_RQ                    BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                          BIT(8), /*   0    */
      2 ENCIPHERED_SEED                CHAR(8); /*  1-8   */
```

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 zero; the transform is the cryptography session-seed value with the first four bytes inverted

Note: The cryptography session-seed is used as the seed for all session-level cryptography encipherment and decipherment provided for FMD RUs.

CTERM; SSCP-->PLU, Norm; FMD NS(s) (CONTROL TERMINATE)

```
DCL 1 CTERM_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 FORMAT                            BIT(8), /*   3    */
      2 TYPE                              BIT(8), /*   4    */
      2 REASON                            BIT(8), /*   5    */
      2 RESERVED                         BIT(16), /*  6-7   */
      2 SESSION_KEY                       BIT(1), /*   8    */
                                          /* See page E-127 */
      2 SESSION_KEY_CONTENT              CHAR(4), /*  9-12  */
      2 REQUESTER_ID_LENGTH               BIT(8), /*  13    */
      2 REQUESTER_ID
            CHAR(REFER(REQUESTER_ID_LENGTH)), /*  14-n  */
      2 PASSWORD_LENGTH                   BIT(8), /*  n+1   */
      2 PASSWORD  CHAR(REFER(PASSWORD_LENGTH)); /* n+2-p  */
```

0-2 X'810602' NS header

3 bits 0-3, 0000 Format 0 (only value defined)
bits 4-7, reserved

4 Type:
bits 0-1, reserved
bits 2-3, 00 reserved
01 orderly
10 forced
11 cleanup
bits 4-7, reserved

5 Reason:
bit 0, 0 network user
1 network manager
bit 1, 0 normal
1 abnormal
bits 2-7, detailed reason (dependent upon bits 0-1):

- For bits 0-1, 00 user and normal

bits 2-7, 000000 general category
000001 self, OLU = PLU
000010 self, OLU = SLU
000011 other

- For bits 0-1, 01 user and abnormal
  bits 2-7, 000000 general category (only value defined)
- For bits 0-1, 10 manager and normal
  bits 2-7, 000000 general category
  000001 operator command--session
  000010 operator command--LU
  000011 operator command--domain
- For bits 0-1, 11 manager and abnormal
  bits 2-7, 000000 general category
  000001 operator command
  000010 restart procedure
  000011 preempt procedure
  000100 unrecoverable path error
  000101 unrecoverable destination error

6-7	Reserved
8	Session key: X'07' network address pair
9-10	Network address of PLU
11-12	Network address of SLU
13-n	Requester ID
13	Length, in binary, of requester ID Note: X'00' = no requester ID
14-n	Requester ID: the ID, in EBCDIC characters, of the end user initiating the session deactivation request (May be used to establish the authority of the end user to access a particular resource or service.)
n+1-p	Password
n+1	Length, in binary, of password Note: X'00' = no password is present
n+2-p	Password used to verify the identity of the end user

DACTCDRM; SSCP-->SSCP, Exp; SC (DEACTIVATE CROSS-DOMAIN RESOURCE MANAGER)

```
DCL 1 DACTCDRM_RQ                BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                           BIT(8), /*   0    */
      2 FORMAT                            BIT(4), /*   1    */
      2 TYPE_DEACTIVATION                 BIT(4),
      2 SON_CAUSE                        CHAR(*); /* 2-3|5  */
```

0	X'15' request code
1	bits 0-3, format: X'0' (only value defined) bits 4-7, type deactivation requested: X'1' normal end of session X'2' invalid activation parameter, sent by the primary half-session to deactivate the session and to

indicate to the secondary that the response to ACTCDRM contained an invalid parameter

X'3' session outage notification (SON)

- End of Type 1; Type 2 Continues

2-5 Reason code (included only if type deactivation requested is invalid activation parameter, i.e., byte 1, bits 4-7 = X'2'): sense data (see Appendix G) corresponding to the error

- Type 3 Continues

2 Cause of session outage notification:

X'07' virtual route inoperative: the virtual route being used bythe SSCP-SSCP session has become inoperative, thus forcing the deactivation of the SSCP-SSCP session

X'0B' virtual route deactivated: the identified SSCP-SSCP session is being deactivated because of a forced deactivation of the virtual route being used by the session

X'0C' SSCP failure--unrecoverable: the identified (SSCP,SSCP) session had to be deactivated because of an abnormal termination of one of the SSCPs of the session; recovery from the failure was not possible

X'0D' session override: the subject session has to be deactivated because of a more recent session activation request for the same session over a different virtual route

X'0E' SSCP failure--recoverable: the identified (SSCP,SSCP) session had to be deactivated because of an abnormal termination of one of the SSCPs of the session; recovery from the failure may be possible

X'0F' cleanup: the SSCP is resetting its half-session before it receives the response from the partner SSCP receiving the DACTCDRM

X'10' SSCP contention: two SSCPs have sent each other an ACTCDRM request over different virtual routes; the SSCP receiving the ACTCDRM from the SSCP with the greater SSCP ID sends DACTCDRM, with this SON code, to the other SSCP over the same virtual route on which the contention-losing ACTCDRM was sent

3 Reserved

```
DACTCONNIN;  SSCP-->PU_T4|5, PUCP-->PU,  Norm;  FMD NS(c)  (DEACTIVATE
          CONNECT IN)

          DCL 1 DACTCONNIN_RQ            BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                      BIT(24), /*  0-2   */
                2 LINK_ADDRESS                   BIT(16); /*  3-4   */

          0-2       X'010217' NS header
          3-4       Network address of link

DACTLINK; SSCP-->PU_T4|5, PUCP-->PU, Norm; FMD NS(c) (DEACTIVATE LINK)

          DCL 1 DACTLINK_RQ              BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                      BIT(24), /*  0-2   */
                2 LINK_ADDRESS                   BIT(16); /*  3-4   */

          0-2       X'01020B' NS header
          3-4       Network address of link

DACTLU; SSCP<-->LU, Exp; SC (DEACTIVATE LOGICAL UNIT)

          DCL 1 DACTLU_RQ                BASED(ADDR(RU)), /* Byte(s)*/
                2 RQ_CODE                        BIT(8), /*    0    */
                2 TYPE_DEACTIVATION              BIT(8), /*    1    */
                2 SON_CAUSE                      BIT(8); /*    2    */

          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 identified session
                            is being deactivated because of a
                            +RSP(ACTPU, Cold)
                     X'0B'  virtual route deactivated: the identified
                            (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
                            subject session had to be reset because of
                            an abnormal termination; recovery from the
                            failure was not possible
                     X'0E'  SSCP or LU failure--recoverable: the
                            identified (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

DACTPU; SSCP|PUCP-->PU, PU-->SSCP, Exp; SC (DEACTIVATE PHYSICAL UNIT)

```
DCL 1 DACTPU_RQ                 BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                           BIT(8), /*   0    */
      2 TYPE_DEACTIVATION                 BIT(8), /*   1    */
      2 SON_CAUSE                         BIT(8); /*   2    */
```

0 X'12' request code

1 Type deactivation requested:

- X'01' final use, physical connection may be broken
- X'02' not final use, physical connection should not be broken
- X'03' session outage notification (SON)

2 Cause (not present if byte 1 ¬= X'03'):

- X'07' virtual route inoperative: the virtual route for the (SSCP,PU) session has become inoperative, thus forcing the deactivation of the (SSCP,PU) session
- X'08' route extension inoperative: the route extension serving the (SSCP,PU) session has become inoperative, thus forcing the deactivation of the (SSCP,PU) session
- X'09' hierarchical reset: the identified session is being deactivated because of a +RSP(ACTPU, Cold)
- X'0B' virtual route deactivated: the identified (SSCP,PU) session is being deactivated because of a forced deactivation of the virtual route being used by the session
- X'0C' SSCP or PU failure--unrecoverable: the identified (SSCP,PU) session had to be deactivated because of an abnormal termination of the SSCP or PU of the session; recovery from the failure was not possible
- X'0D' session override: the subject session has to be deactivated because of a more recent session activation request for the same session over a different virtual route
- X'0E' SSCP or PU failure--recoverable: the identified (SSCP,PU) session had to be deactivated because of an abnormal termination of the SSCP or PU 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 PU that is being deactivated.

DACTTRACE; SSCP-->PU_T4|5, Norm; FMD NS(ma) (DEACTIVATE TRACE)

```
DCL 1 DACTTRACE_RQ              BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2    */
      2 LINK_ADDRESS                    BIT(16), /*  3-4    */
      2 TRACE_TYPE                       BIT(8), /*   5     */
      2 TRACE_DATA                      CHAR(*); /*  6-n    */
```

0-2 X'010303' NS header
3-4 Network address of resource to be traced
5 Selected trace
bit 0, transmission group trace
bits 1-6, reserved
bit 7, link trace
6-n Data to support trace deactivation

DELETENR; SSCP-->PU_T4|5, Norm; FMD NS(c) (DELETE NETWORK RESOURCE)

```
DCL 1 DELETENR_RQ               BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2    */
      2 RESOURCE_ADDRESS                BIT(16); /*  3-4    */
```

0-2 X'41021C' NS header
3-4 Network address of resource being deleted

DELIVER; SSCP-->LU, Norm; FMD NS(mn) (DELIVER)

```
DCL 1 DELIVER_RQ                BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2    */
      2 FORMAT                           BIT(8), /*   3     */
      2 RESERVED                         BIT(7), /*   4     */
      2 FORMAT_EMBEDDED_RU               BIT(1),
      2 RESERVED                         BIT(8), /*   5     */
      2 EMBEDDED_RU_LENGTH              BIT(16), /*  6-7    */
      2 EMBEDDED_RU
              CHAR(REFER(EMBEDDED_RU_LENGTH)), /*  8-n    */
      2 ORIGIN_NAME_TYPE                 BIT(8), /*  n+1    */
      2 ORIGIN_NAME_LENGTH               BIT(8), /*  n+2    */
      2 ORIGIN_NAME
              CHAR(REFER(ORIGIN_NAME_LENGTH)), /* n+3-p   */
      2 DESTINATION_NAME_TYPE            BIT(8), /*  p+1    */
      2 DESTINATION_NAME_LENGTH          BIT(8), /*  p+2    */
      2 DESTINATION_NAME
        CHAR(REFER(DESTINATION_NAME_LENGTH)), /* p+3-q   */
      2 CONFIG_HIERARCHY_STRUCTURE     CHAR(*); /*  q-t    */
```

0-2 X'810812' NS header
3 Format: X'00' format 0 (only value defined)
4 Flags:
bits 0-6, reserved
bit 7, format of embedded NS RU:

0 embedded NS RU contains a CNM header
1 embedded NS RU does not contain a CNM header

5	Reserved
6-7	Length, in binary, of embedded NS RU
8-n	Embedded NS RU
n+1-p	Network Name of Origin PU
n+1	Type: X'F1' PU
n+2	Length, in binary, of symbolic name
n+3-p	Symbolic name in EBCDIC characters
p+1-q	Network Name of Target PU, LU, Adjacent Link Station, or Link
p+1	Type: X'F1' PU X'F3' LU X'F7' adjacent link station X'F9' link
p+2	Length, in binary, of symbolic name
p+3-q	Symbolic name in EBCDIC characters

- If the target is a PU in a PU_T1|2 node or is an adjacent link station attached to a PU_T4|5 node

q+1-s+1	Configuration Hierarchy Network Name List
q+1	Type: X'F9' link connecting the PU_T1|2 node to the PU_T4| 5 node containing the boundary function for the target PU or connecting the adjacent link station to the PU_T4|5 node
q+2	Length, in binary, of symbolic name
q+3-r	Symbolic name in EBCDIC characters
r+1	Type: X'F1' PU in the PU_T4|5 node containing the boundary function for the target PU or attaching the target adjacent link station
r+2	Length, in binary, of symbolic name
r+3-s	Symbolic name in EBCDIC characters
s+1	X'00' (end of configuration hierarchy network name list)

- If the target is an LU in a PU_T1|2 node:

q+1-t+1	Configuration Hierarchy Network Name List
q+1	Type: X'F1' PU in the PU_T1|2 node containing the target L U
q+2	Length, in binary, of symbolic name
q+3-r	Symbolic name in EBCDIC characters
r+1	Type: X'F9' link connecting the PU_T1|2 node to the PU_T4| 5 node containing the boundary function for the target LU
r+2	Length, in binary, of symbolic name
r+3-s	Symbolic name in EBCDIC characters
s+1	Type: X'F1' PU in the PU_T4|5 node containing the boundary function for the target LU
s+2	Length, in binary, of symbolic name
s+3-t	Symbolic name in EBCDIC characters
t+1	X'00' (end of configuration hierarchy network name list)

- If the target is a link attached to, or a PU or LU

```
                in, a PU_T4|5 node:
        q+1-q+1 Configuration Hierarchy Network Name List
        q+1     X'00' (end of configuration hierarchy network name
                list)
```

DISCONTACT; SSCP-->PU_T4|5, PUCP-->PU, Norm; FMD NS(c) (DISCONTACT)

```
        DCL 1 DISCONTACT_RQ             BASED(ADDR(RU)), /* Byte(s)*/
              2 NS_HEADER                       BIT(24), /*  0-2   */
              2 ALS_ADDRESS                     BIT(16); /*  3-4   */

        0-2     X'010202' NS header
        3-4     Network address of adjacent link station to be
                discontacted
```

DISPSTOR; SSCP-->PU_T4|5, Norm; FMD NS(ma) (DISPLAY STORAGE)

```
        DCL 1 DISPSTOR_RQ               BASED(ADDR(RU)), /* Byte(s)*/
              2 NS_HEADER                       BIT(24), /*  0-2   */
              2 RESOURCE_ADDRESS                BIT(16), /*  3-4   */
              2 TYPE                             BIT(8), /*   5    */
              2 RESERVED                         BIT(8), /*   6    */
              2 DISPLAY_LENGTH                  BIT(16), /*  7-8   */
              2 DISPLAY_LOCATION                BIT(32); /*  9-12  */
                .

        0-2     X'010331' NS header
        3-4     Network address of resource to be displayed
        5       Display target and type:
                bits 0-3, target address space to be displayed
                          Note: Refer to implementation
                          documentation for description of these
                          values.
                bits 4-7, display type:
                           0001 nonstatic storage display
                           0010 static snapshot display
        6       Reserved
        7-8     Number of bytes to be displayed
        9-12    Beginning location of display
```

DSRLST; SSCP-->SSCP, Norm; FMD NS(s) (DIRECT SEARCH LIST)

```
        DCL 1 DSRLST_RQ                 BASED(ADDR(RU)), /* Byte(s)*/
              2 NS_HEADER                       BIT(24), /*  0-2   */
              2 LIST_TYPE                        BIT(8), /*   3    */
              2 CONTROL_LIST                    CHAR(*); /*  4-m   */

        0-2     X'818627' NS header
        3       Control list type : X'01' (only value defined)
        4-m     Control list search argument: network name of LU
                (only value defined)
        4       Type: X'F3' logical unit
        5       Length, in binary, of symbolic name
        6-m     Symbolic name in EBCDIC characters
```

DUMPFINAL; SSCP-->PU_T4|5, Norm; FMD NS(c) (DUMP FINAL)

```
DCL 1 DUMPFINAL_RQ                BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                         BIT(24), /*  0-2   */
      2 ALS_ADDRESS                       BIT(16); /*  3-4   */
```

0-2 X'010208' NS header
3-4 Network address of adjacent link station of the node being dumped

DUMPINIT; SSCP-->PU_T4|5, Norm; FMD NS(c) (DUMP INITIAL)

```
DCL 1 DUMPINIT_RQ                 BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                         BIT(24), /*  0-2   */
      2 ALS_ADDRESS                       BIT(16); /*  3-4   */
```

0-2 X'010206' NS header
3-4 Network address of adjacent link station of the node to be dumped

DUMPTEXT; SSCP-->PU_T4|5, Norm; FMD NS(c) (DUMP TEXT)

```
DCL 1 DUMPTEXT_RQ                 BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                         BIT(24), /*  0-2   */
      2 ALS_ADDRESS                       BIT(16); /*  3-4   */
```

0-2 X'010207' NS header
3-4 Network address of adjacent link station of the node to be dumped
5-8 Starting address where dump data is to begin
9-10 Length of text: two-byte binary count of the number of bytes of dump data to be returned

ECHOTEST; SSCP-->LU, Norm; FMD NS(ma) (ECHO TEST)

```
DCL 1 ECHOTEST_RQ                 BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                         BIT(24), /*  0-2   */
      2 TEST_DATA_LENGTH                   BIT(8), /*   3    */
      2 TEST_DATA
                  CHAR(REFER(TEST_DATA_LENGTH)); /*  4-n   */
```

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

ER_INOP; PU_T4|5-->SSCP, Norm; FMD NS(c) (EXPLICIT ROUTE INOPERATIVE)

```
DCL 1 ER_INOP_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                         BIT(24), /*  0-2   */
      2 FORMAT                             BIT(8), /*   3    */
      2 REASON_CODE                        BIT(8), /*   4    */
      2 ORIGINATING_SA                    BIT(32), /*  5-8   */
```

```
          2 TG_ADJ_SA                          BIT(32), /*  9-12  */
          2 TG_NUM                              BIT(8), /*   13   */
          2 CNT_ER_FIELD                        BIT(8), /*   14   */
          2 ER_FIELD(1:REFER(CNT_ER_FIELD)),
            3 SA                               BIT(32), /*15-18+6n*/
            3 MASK                             BIT(16); /*19-20+6n*/
```

0-2	X'41021D' NS header
3	Format: X'01' (only value defined)
4	Reason code for INOP: X'01' unexpected routing interruption over a transmission group, e.g., the last active link on a TG has failed X'02' controlled routing interruption such as the result of a DISCONTACT
5-8	Address of the subarea that originated the corresponding NC_ER_INOP
9-12	Subarea address on the other end of the transmission group that had the routing interruption
13	TGN of the transmission group that had routing interruption
14	Number of destination subareas that are on the ERs using the above TG
15-20	Inoperative ER Field
15-18	Subarea address of a destination that is routed to over an ER using the above TG
19-20	Inoperative explicit route mask: a bit is on if the ER of the corresponding ERN is inoperative (Bit 0 corresponds to ERN 0, bit 1 to ERN 1, and so forth.)
21-n	Any additional six-byte entries in the same format as bytes 15-20

ER_TESTED; PU_T4|5-->SSCP, Norm; FMD NS(ma) (EXPLICIT ROUTE TESTED)

```
DCL 1 ER_TESTED_RQ              BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 FORMAT                           BIT(8), /*   3    */
      2 TYPE                             BIT(8), /*   4    */
      2 ER_LENGTH                        BIT(8), /*   5    */
      2 MAX_ER_LENGTH                    BIT(8), /*   6    */
      2 DESTINATION_SA                  BIT(32), /*  7-10  */
      2 RESERVED                        BIT(12), /* 11-12  */
      2 ER_NUM                           BIT(4),
      2 ORIGINATING_SA                  BIT(32), /* 13-16  */
      2 REV_ERN_MASK                    BIT(16), /* 17-18  */
      2 MAX_PIU_SIZE                    BIT(16), /* 19-20  */
      2 MAX_PIU_SIZE_FROM_TEST          BIT(16), /* 21-22  */
      2 ORIGINATING_SSCP                BIT(48), /* 23-28  */
      2 RQ_CORRELATION                 CHAR(10), /* 29-38  */
      2 REPLY_SA                        BIT(32), /* 39-42  */
      2 TG_ADJ_SA                       BIT(32), /* 43-46  */
      2 TG_NUM                           BIT(8), /*   47   */
```

```
      2 ORIGINATING_ADJ_SA              BIT(32), /* 48-51  */
      2 ORIGINATING_TGN                  BIT(8); /*   52   */
```

0-2	X'410386' NS header
3	Format: X'1' Format 1 X'2' Format 2; same as Format 1, except that it includes bytes 48-52
4	Type: X'00' the corresponding NC_ER_TEST reached its destination subarea X'02' ER not reversible since there is no reverse ERN defined X'03' encountered a PU that does not support ER and VR protocols X'04' ER length exceeded that specified in the NC_ER_TEST request X'05' ER requires a TG that is not active X'06' ER is not defined in the NC_ER_TEST_REPLY originating node
5	Explicit route length, in terms of the number of transmission groups in the explicit route, as accumulated in NC_ER_TEST
6	Maximum ER length, as specified in the NC_ER_TEST request
7-10	Subarea address of the destination PU of the corresponding NC_ER_TEST
11	Reserved
12	bits 0-3, reserved bits 4-7, ERN of the ER tested
13-16	Subarea address of the originating PU of the corresponding NC_ER_TEST
17-18	Reverse ERN mask: A bit is <u>on</u> if the corresponding ERN can be used to route from the NC_ER_TEST_REPLY originating subarea to the NC_ER_TEST originating subarea (Bit 0 corresponds to ERN 0, bit 1 to ERN 1, and so forth.)
19-20	Maximum PIU length allowed on the reverse ERN specified in byte 17-18: X'00' no restriction (only value defined)
21-22	Maximum PIU size accumulated by the corresponding NC_ER_TEST: X'00' no restriction (only value defined)
23-28	Network address of the SSCP originating the test request
29-38	Request Correlation field, as specified in the corresponding ROUTE_TEST
39-42	Subarea address of the PU that originated the corresponding NC_ER_TEST_REPLY
43-46	Subarea address depending on the Type field (Byte 4) as follows:

Type	Contents of this field
X'00'	reserved
X'02'	subarea on the ER prior to that with no reverse ERN defined
X'03'	subarea that does not support ER and VR protocols
X'04'	subarea on the ER preceding the subarea where the explicit route length (byte 5 of NC_ER_TEST) is incremented to a value one more than the maximum ER length limit (byte 6)
X'05'	subarea on the other end of the TG that is not active
X'06'	subarea on the ER from which the PU (that does not have the ER defined) received the corresponding NC_ER_TEST

47 TGN of the TG between the subareas specified in bytes 39-42 and 43-46; reserved if Type is X'00'.

Note: End of Format 1; Format 2 continues below

48-51 Subarea address of the adjacent node through which the tested explicit route flows from this node

52 Transmission group number of the TG (to the node identified in bytes 48-51) over which the tested explicit route flows from this node

ESLOW; PU_T4-->SSCP, Norm; FMD NS(c) (ENTERING SLOWDOWN)

```
DCL 1 ESLOW_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 PU_ADDRESS                      BIT(16); /*  3-4   */
```

0-2 X'010214' NS header

3-4 Network address of PU

EXECTEST; SSCP-->PU_T4|5, Norm; FMD NS(ma) (EXECUTE TEST)

```
DCL 1 EXECTEST_RQ               BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 RESOURCE_ADDRESS                BIT(16), /*  3-4   */
      2 TEST_SELECTION                  BIT(32), /*  5-8   */
      2 TEST_SELECTION_DATA             CHAR(*); /*  9-n   */
```

0-2 X'010301' NS header

3-4 Network address of resource to be tested

5-8 Binary code selecting the test

9-n Data to support the selected test

EXSLOW; PU_T4-->SSCP, Norm; FMD NS(c) (EXITING SLOWDOWN)

```
DCL 1 EXSLOW_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 PU_ADDRESS                       BIT(16); /*  3-4   */
```

0-2 X'010215' NS header
3-4 Network address of PU

FNA; SSCP-->PU_T4|5, Norm; FMD NS(c) (FREE NETWORK ADDRESSES)

```
DCL 1 FNA_RQ                     BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 TARGET_ADDRESS                   BIT(16), /*  3-4   */
      2 ENTRY_CNT                         BIT(8), /*   5    */
      2 TYPE                              BIT(8), /*   6    */
      2 SUBFIELD(1:REFER(ENTRY_CNT))     BIT(16); /*  7-8   */
```

0-2 X'01021A' NS header
3-4 Network address of target link, SPU, or LU (X'0000' indicates that the network addresses in bytes 7-n are to be freed without verification of their attachment to a specific target link, SPU, or LU.)
5 Number of SPU (if bytes 3-4 specify a link), BF.LU (if bytes 3-4 specify an SPU), or LU (if bytes 3-4 specify an LU network address used for the SSCP-LU session) network addresses to be freed (X'00' = all--and bytes 7-n not present)
6 Type: X'80' noncontiguous
7-8 First network address to be freed
9-n Any additional network addresses (two-byte multiples)

Note: All the network addresses specified in bytes 7-n are associated with the same target link, SPU, or LU. See the following table for the relation of target resources to resources to free.

Target resource	Resources to free
PU	LUs identified by network addresses associated with SSCP-LU sessions
LU (identified by the network address associated with an SSCP-LU session)	LU network addresses used as primary network addresses in parallel sessions
Link	BF.PUs and adjacent link stations
BF.PU	BF.LUs

FORWARD; LU-->SSCP, Norm; FMD NS(mn) (FORWARD)

```
DCL 1 FORWARD_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                         BIT(24), /*  0-2    */
      2 FORMAT                             BIT(8), /*   3     */
      2 FLAGS                              BIT(8), /*   4     */
      2 RESERVED                           BIT(8), /*   5     */
      2 EMBEDDED_RU_LENGTH                BIT(16), /*  6-7    */
      2 EMBEDDED_RU
              CHAR(REFER(EMBEDDED_RU_LENGTH)), /*  8-n    */
      2 DESTINATION_NAME_TYPE              BIT(8), /*  n+1    */
      2 DESTINATION_NAME_LENGTH            BIT(8), /*  n+2    */
      2 DESTINATION_NAME
        CHAR(REFER(DESTINATION_NAME_LENGTH)), /* n+3-p  */
      2 TARGET_NAME_TYPE                   BIT(8), /*  p+1    */
      2 TARGET_NAME_LENGTH                 BIT(8), /*  p+2    */
      2 TARGET_NAME
              CHAR(REFER(TARGET_NAME_LENGTH)); /* p+3-q  */
```

0-2 X'810810' NS header

3 Format: X'00' format 0 (only value defined)

4 Flags:

bits 0-5, reserved

bit 6, solicitation indicator:

- 0 embedded NS RU solicits a reply request
- 1 embedded NS RU does not solicit a reply request

bit 7, format of embedded NS RU:

- 0 embedded NS RU contains a (partially initialized) CNM header
- 1 embedded NS RU does not contain a CNM header

5 Reserved

6-7 Length, in binary, of embedded NS RU

8-n Embedded NS RU

n+1-p Network Name of Destination PU

```
n+1       Type:
           X'F1'  PU
n+2       Length, in binary, of symbolic name
n+3-p     Symbolic name in EBCDIC characters
p+1-q     Network Name of Target PU, LU, Adjacent Link
          Station, or Link
p+1       Type:
           X'F1'  PU
           X'F3'  LU
           X'F7'  adjacent link station
           X'F9'  link
p+2       Length, in binary, of symbolic name
p+3-q     Symbolic name in EBCDIC characters
```

INIT-OTHER; ILU-->SSCP, Norm; FMD NS(s) (INITIATE-OTHER)

```
DCL 1 INIT_OTHER_RQ              BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 FORMAT                            BIT(4), /*   3    */
      2 RESERVED                          BIT(4),
      2 TYPE                              BIT(8), /*   4    */
      2 LU1_QUEUING_CONDITIONS            BIT(8), /*   5    */
      2 LU2_QUEUING_CONDITIONS            BIT(8), /*   6    */
      2 INITIATE_ORIGIN                   BIT(8), /*   7    */
      2 NOTIFY_SPECIFICATIONS             BIT(8), /*   8    */
      2 MODE_NAME                        CHAR(8), /*  9-16  */
      2 LU1_NAME_TYPE                     BIT(8), /*   17   */
      2 LU1_NAME_LENGTH                   BIT(8), /*   18   */
      2 LU1_NAME  CHAR(REFER(LU1_NAME_LENGTH)), /*  19-m  */
      2 LU2_NAME_TYPE                     BIT(8), /*  m+1   */
      2 LU2_NAME_LENGTH                   BIT(8), /*  m+2   */
      2 LU2_NAME  CHAR(REFER(LU2_NAME_LENGTH)), /* m+3-n  */
      2 REQUESTER_ID_LENGTH               BIT(8), /*  n+1   */
      2 REQUESTER_ID
            CHAR(REFER(REQUESTER_ID_LENGTH)), /* n+2-p  */
      2 PASSWORD_LENGTH                   BIT(8), /*  p+1   */
      2 PASSWORD  CHAR(REFER(PASSWORD_LENGTH)), /* p+2-q  */
      2 USER_DATA_LENGTH                  BIT(8), /*  q+1   */
      2 USER_DATA
              CHAR(REFER(USER_DATA_LENGTH)), /* q+2-r  */
      2 URC_LENGTH                        BIT(8), /*  r+1   */
      2 URC           CHAR(REFER(URC_LENGTH)), /* r+2-s  */
      2 COS_NAME                         CHAR(8); /*s+1-s+8 */
```

```
0-2       X'810680' NS header
3         Format:
          bits 0-3,  0001  Format 1
                     0010  Format 2: specifies the COS name
                           field in addition to the
                           parameters in Format 1
          bits 4-7, reserved
4         Type:
          bits 0-1,  00  dequeue (DQ) a previously enqueued
                         initiate request (See bits 2-3 for
```

further specification of dequeue actions.)
01 initiate only (I); do not enqueue
10 enqueue only (Q) (See bytes 5-6 for further specification of queuing conditions.)
11 initiate/enqueue (I/Q): enqueue the request if it cannot be satisfied immediately

bits 2-3, (used for DQ; otherwise, reserved)
00 leave on queue if dequeuing attempt is unsuccessful
01 remove from queue if dequeuing attempt is unsuccessful
10 remove from queue; do not attempt initiation
11 reserved

bit 4, reserved

bits 5-6, PLU/SLU specification:
00 LU1 is PLU
01 LU2 is PLU

bit 7, reserved

5 Queuing conditions for LU1 (when Type = DQ, bits 0-7 are reserved):

bit 0, 0 do not enqueue if session limit will be exceeded
1 enqueue if session limit will be exceeded

bit 1, 0 do not enqueue if the LU is not currently able to comply with the PLU/SLU specification (as given in byte 4, bits 5-6)
1 enqueue even though the LU might not be currently able to comply with the PLU/SLU specification

bit 2, 0 do not enqueue if CDINIT loses contention
1 enqueue if CDINIT loses contention

bit 3, 0 do not enqueue if there are no SSCP-LU paths
1 enqueue if there are no SSCP-LU paths

bit 4, reserved

bits 5-6, queuing position/service
00 enqueue this request at the bottom of the queue (the request is put at the bottom of the queue and serviced last)
01 enqueue this request FIFO
10 enqueue this request LIFO
11 reserved

bit 7, 0 do not enqueue for recovery retry
1 enqueue for recovery retry (This is a queue that is used for recovery-reactivating an LU-LU session

when the session, though it had been successfully activated, fails for some reason. Elements on this queue are not dequeued when a session activation is successfully completed; explicit session deactivation requests are needed to dequeue elements from this queue.)

6 Queuing conditions for LU2 (When Type = DQ, bits 0-7 are reserved):

bit 0, 0 do not enqueue if session limit will be exceeded
1 enqueue if session limit will be exceeded

bit 1, 0 do not enqueue if the LU is not currently able to comply with the PLU/SLU specification (as given in byte 4, bits 5-6)
1 enqueue even though the LU might not be currently able to comply with the PLU/SLU specification

bit 2, 0 do not enqueue if CDINIT loses contention
1 enqueue if CDINIT loses contention

bit 3, 0 do not enqueue if there are no SSCP-LU paths
1 enqueue if there are no SSCP-LU paths

bit 4, reserved

bits 5-6, queuing position/service
00 enqueue this request at the bottom of the queue (the request is put at the bottom of the queue and serviced last)
01 enqueue this request FIFO
10 enqueue this request LIFO
11 reserved

bit 7, 0 do not queue for recovery retry
1 enqueue for recovery retry (This is a queue that is used for recovery-reactivating an LU-LU session when the session, though it had been successfully activated, fails for some reason. Elements on this queue are not dequeued when a session activation is successfully completed; explicit session deactivation requests are needed to dequeue elements from this queue.)

Notes on Bytes 5-6:

- If enqueuing for recovery is desired, it must be indicated in both LU1 and LU2 Queuing Conditions bytes (bit 7 = '1').
- Bit 2 (CDINIT contention) must have the same setting for both LU1 and LU2. (Contention

occurs when both SSCPs try to set up a session between the same LUs at the same time.)

- Enqueueing is not performed if the DLU is unknown, or if the domain of either LU is in takedown status.

7 INITIATE origin:
bits 0-2, reserved
bit 3, (when Type = DQ, bit 3 is reserved)
0 network user is the initiator
1 network manager is the initiator
bits 4-7, reserved

8 NOTIFY
bits 0-1, (when Type = DQ, bits 0 and 1 are reserved)
00 do not send NOTIFY to LUs in session with LU1
01 send NOTIFY to all LUs in session with LU1
10 send NOTIFY to all LUs in session with LU1 only if the request is queued
11 reserved
bits 2-3, (when Type = DQ, bits 2 and 3 are reserved)
00 do not send NOTIFY to LUs in session with LU2
01 send NOTIFY to all LUs in session with LU2
10 send NOTIFY to all LUs in session with LU2 only if the request is enqueued
11 reserved
bit 4, 0 do not send NOTIFY to the ILU when INIT is dequeued
1 send NOTIFY to the ILU when INIT is dequeued
bit 5, 0 do not send NOTIFY to the ILU when the requested session is set up
1 send NOTIFY to the ILU when the requested session is set up
bits 6-7, reserved

9-16 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 (When Type = DQ, the Mode Name field is reserved.)

17-m Uninterpreted name of LU1
17 Type: X'F3' logical unit
18 Length, in binary, of LU1 name
19-m EBCDIC character string

m+1-n Uninterpreted name of LU2
m+1 Type: X'F3' logical unit

m+2	Length, in binary, of LU2 name
m+3-n	EBCDIC character string
n+1-p	Requester ID
n+1	Length, in binary, of requester ID Note: X'00' = no requester ID
n+2-p	Requester ID: the ID, in EBCDIC characters, of the end user initiating the request (May be used to establish the authority of the end user to access a particular resource.)
p+1-q	Password
p+1	Length, in binary, of password Note: X'00' = no password is present
p+2-q	Password used to verify the identity of the end user
q+1-r	User Field (When Type = DQ, user field is reserved)
q+1	Length, in binary, of user data Note: X'00' = no user data is present
q+2-r	User data
q+2	User data key X'00' structured subfields follow -X'00' first byte of unstructured user data Note: Individual structured subfields may be omitted entirely. When present, they appear in ascending field number order.
	• For unstructured user data
q+3-r	Remainder of unstructured user data
	• For structured user data
q+3-r	Structured subfields (For detailed definitions, see the structured user data section on page E-129.)
r+1-s	User Request Correlation (URC) field (When Type = DQ, the URC must be the same as on the original INIT-OTHER request.)
r+1	Length, in binary, of URC Note: X'00' = no URC
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 the initiating request End of Format 1; Format 2 Continues
s+1-s+8	COS name: symbolic name of class of service in EBCDIC characters (A value of eight space (X'40') characters may be specified; in this case, the COS name is derived from the mode name table, using the mode name received in byes 9-16.)

INIT-OTHER-CD; SSCP-->SSCP, Norm; FMD NS(s) (INITIATE-OTHER CROSS-DOMAIN)

```
DCL 1 INIT_OTHER_CD_RQ          BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 FORMAT                           BIT(8), /*   3    */
      2 TYPE                             BIT(8), /*   4    */
```

```
        2 LU1_QUEUING_CONDITIONS          BIT(8), /*   5    */
        2 LU2_QUEUING_CONDITIONS          BIT(8), /*   6    */
        2 PCID                           CHAR(8), /*  7-14  */
        2 INITIATE_ORIGIN                 BIT(8), /*   15   */
        2 NOTIFY_SPECIFICATIONS           BIT(8), /*   16   */
        2 MODE_NAME                      CHAR(8), /* 17-24  */
        2 LU1_NAME_TYPE                   BIT(8), /*   25   */
        2 LU1_NAME_LENGTH                 BIT(8), /*   26   */
        2 LU1_NAME  CHAR(REFER(LU1_NAME_LENGTH)), /*  27-m  */
        2 LU2_NAME_TYPE                   BIT(8), /*  m+1   */
        2 LU2_NAME_LENGTH                 BIT(8), /*  m+2   */
        2 LU2_NAME  CHAR(REFER(LU2_NAME_LENGTH)), /* m+3-n  */
        2 REQUESTER_ID_LENGTH             BIT(8), /*  n+1   */
        2 REQUESTER_ID
              CHAR(REFER(REQUESTER_ID_LENGTH)), /* n+2-p  */
        2 PASSWORD_LENGTH                 BIT(8), /*  p+1   */
        2 PASSWORD  CHAR(REFER(PASSWORD_LENGTH)), /* p+2-q  */
        2 USER_DATA_LENGTH                BIT(8), /*  q+1   */
        2 USER_DATA
                CHAR(REFER(USER_DATA_LENGTH)), /* q+2-r  */
        2 COS_NAME_INIT                   BIT(8), /*  r+1   */
        2 COS_NAME                       CHAR(8); /*r+2-r+9 */
```

0-2 X'818640' NS header

3 Format:

bits 0-3,
- 0000 Format 0
- 0010 Format 2: specifies COS name field in addition to the parameters in Format 0

bits 4-7, reserved

4 Type:

bits 0-1,
- 00 dequeue (DQ) a previously enqueued initiate request. (See bits 2-3 for further specification of dequeue actions.)
- 01 initiate only (I); do not enqueue
- 10 enqueue only (Q): (See bytes 5-6 for further specification of queuing conditions.)
- 11 initiate/enqueue (I/Q): enqueue the request if it cannot be satisfied immediately

bits 2-3, (used for DQ; otherwise, reserved)
- 00 leave on queue if dequeuing attempt is unsuccessful
- 01 remove from queue if dequeuing attempt is unsuccessful
- 10 remove from queue, do not attempt initiation
- 11 reserved

bit 4, reserved

bits 5-6, PLU/SLU specification:
- 00 LU1 is PLU
- 01 LU2 is PLU

bit 7, reserved

5 Queuing conditions for LU1 (When Type = DQ, bits 0-7, are reserved.):

bit 0,
- 0 do not enqueue if session limit will be exceeded
- 1 enqueue if session limit will be exceeded

bit 1,
- 0 do not enqueue if the LU is not currently able to comply with the PLU/SLU specification (as given in byte 4, bits 5-6)
- 1 enqueue if the LU is not currently able to comply with the PLU/SLU specification

bit 2,
- 0 do not enqueue if CDINIT loses contention
- 1 enqueue if CDINIT loses contention

bit 3,
- 0 do not enqueue if there are no SSCP-LU paths
- 1 enqueue if there are no SSCP-LU paths

bit 4, reserved

bits 5-6,
- 00 enqueue this request at the bottom of the queue (the request is put at the bottom of the queue and serviced last)
- 01 enqueue this request FIFO
- 10 enqueue this request LIFO
- 11 reserved

bit 7,
- 0 do not enqueue for recovery retry
- 1 enqueue for recovery retry (This is a queue that is used for recovery-reactivating an LU-LU session when the session, though it had been successfully activated, fails for some reason. Elements on this queue are not dequeued when a session activation is successfully completed. Explicit session deactivation requests are needed to dequeue elements from this queue.)

6 Queuing conditions for LU2 (When Type = DQ, bits 0-7 are reserved.):

bit 0,
- 0 do not enqueue if session limit will be exceeded
- 1 enqueue if session limit will be exceeded

bit 1,
- 0 do not enqueue if the LU is not currently able to comply with the PLU/SLU specification (as given in byte 4, bits 5-6)
- 1 enqueue even though the LU might not be currently able to comply with the PLU/SLU specification

bit 2,
- 0 do not enqueue if CDINIT loses

contention
1 enqueue if CDINIT loses contention

bit 3, 0 do not enqueue if there are no SSCP-LU paths
1 enqueue even if there are no SSCP-LU paths

bit 4, reserved

bits 5-6, queuing position/service:
00 enqueue this request at the bottom of the queue (the request at the bottom of the queue and is serviced last)
01 enqueue this request FIFO
10 enqueue this request LIFO
11 reserved

bit 7, 0 do not enqueue for recovery retry
1 enqueue for recovery retry (This is a queue that is used for recovery-reactivating an LU-LU session when the session, though it had been successfully activated, fails for some reason. Elements on this queue are not dequeued when a session activation is successfully completed; explicit session deactivation requests are needed to dequeue elements from this queue.)

Notes on Bytes 5-6:

- If enqueuing for recovery is desired, it is indicated in both LU1 and LU2 Queuing Conditions bytes (bit 7 = 1).
- Bit 2 (CDINIT contention) has the same setting for both LU1 and LU2. (Contention occurs when both SSCPs try to set up a session between the same LUs at the same time.)
- Enqueuing is not performed if the DLU is unknown, or if the domain of either LU is in takedown status.

7-14 PCID (When Type = DQ, the PCID is the same as in the original INIT-OTHER-CD request.)

7-8 Network address of SSCP(ILU)

9-14 A unique 6-byte value, generated by the SSCP(ILU), that is retained and used in all cross-domain requests dealing with the same procedure until it is completed; an SSCP maintains correlation between PCID and the URC, if a URC has been provided by the INIT-OTHER request

15 INITIATE origin

bits 0-2, reserved

bit 3, (reserved when Type = DQ.)
0 network user is the initiator
1 network manager is the initiator

bits 4-7, reserved

16 NOTIFY

bits 0-1, (When Type = DQ, bits 0-1 are reserved.)
- 00 do not send NOTIFY to LUs in session with LU1
- 01 send NOTIFY to all LUs in session with LU1
- 10 send NOTIFY to all LUs in session with LU1 only if the request is enqueued
- 11 reserved

bits 2-3, (When Type = DQ, bits 2-3 are reserved.)
- 00 do not send NOTIFY to LUs in session with LU2
- 01 send NOTIFY to all LUs in session with LU2
- 10 send NOTIFY to all LUs in session with LU2 only if the request is enqueued.
- 11 reserved

bit 4,
- 0 do not send NOTIFY to the SSCP(ILU) when INIT is dequeued
- 1 send NOTIFY to the SSCP(ILU) when INIT is dequeued

bits 5-7, reserved

17-24	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 (When Type = DQ, the Mode Name field is reserved.)
25-m	Network Name of LU1
25	Type: X'F3' logical unit
26	Length, in binary, of symbolic name
27-m	Symbolic name, in EBCDIC characters
m+1-n	Network Name of LU2
m+1	Type: X'F3' logical unit
m+2	Length, in binary, of symbolic name
m+3-n	Symbolic name, in EBCDIC characters
n+1-p	Requester ID
n+1	Length, in binary, of requester ID Note: X'00' = no requester ID is present
n+2-p	Requester ID: the ID, in EBCDIC characters, of the end user initiating the request (May be used to establish the authority of the end user to access a particular resource.)
p+1-q	Password
p+1	Length, in binary, of password Note: X'00' = no password is present
p+2-q	Password used to verify the identity of the end user
q+1-r	User Field (When Type = DQ, this field is reserved.)
q+1	Length, in binary, of user data Note: X'00' = no user data is present

q+2-r User data: user-specific data that is passed to the primary LU on the CINIT request

q+2 User data key
X'00' structured subfields follow
¬X'00' first byte of unstructured user data
Note: Individual structured subfields may be omitted entirely. When present, they appear in ascending field number order.

- For unstructured user data

q+3-r Remainder of unstructured user data

- For structured user data

q+3-r Structured subfields (For detailed definitions, see the structured user data section on page E-129.)
Note: With the exception of the NS header and PCID, all the fields in the INIT-OTHER-CD RU are derived from its corresponding INIT-OTHER RU.

- End of Format 0; Format 2 Continues

r+1 COS name field initialization indicator:
bit 0, 0 ILU did not specify COS name
1 ILU did specify COS name
bits 1-7, reserved

r+2-r+9 COS name (reserved if byte r+1, bit 0 = 0): symbolic name of class of service in EBCDIC characters (A value of eight space (X'40') characters may be specified; in this case, the COS name is derived from the mode name table using the mode name received in bytes 17-24.)

INITPROC; SSCP-->PU_T4|5, Norm; FMD NS(c) (INITIATE PROCEDURE)

```
DCL 1 INITPROC_RQ                BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 RESERVED                         BIT(32), /*  3-6   */
      2 TARGET_ADDRESS                   BIT(16), /*  7-8   */
      2 PROCEDURE_TYPE                    BIT(8), /*   9    */
      2 LOAD_MODULE                      CHAR(8); /* 10-17  */
```

0-2 X'410235' NS header

3-6 Reserved

7-8 Network address of PU_T2 for which the procedure is to be initiated

9 Procedure type:
X'00' load (only value defined)

- For procedure type = load

10-17 IPL load module: an eight-character EBCDIC symbolic name of the IPL load module to be sent to the PU identified in bytes 7-8

INIT-SELF; ILU-->SSCP, Norm; FMD NS(s) (INITIATE-SELF)

```
DCL 1 INIT_SELF_FMT0_RQ          BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2    */
      2 FORMAT                            BIT(8), /*   3     */
      2 MODE_NAME                        CHAR(8), /*  4-11   */
      2 DLU_UNINTRP_NAME_TYPE             BIT(8), /*   12    */
      2 DLU_UNINTRP_NAME_LENGTH           BIT(8), /*   13    */
      2 DLU_UNINTRP_NAME
        CHAR(REFER(DLU_UNINTRP_NAME_LENGTH)), /*  14-m   */
      2 REQUESTER_ID_LENGTH               BIT(8), /*  m+1    */
      2 REQUESTER_ID
          CHAR(REFER(REQUESTER_ID_LENGTH)), /* m+2-n   */
      2 PASSWORD_LENGTH                   BIT(8), /*  n+1    */
      2 PASSWORD  CHAR(REFER(PASSWORD_LENGTH)), /* n+2-p   */
      2 USER_DATA_LENGTH                  BIT(8), /*  p+1    */
      2 USER_DATA
              CHAR(REFER(USER_DATA_LENGTH)); /* p+2-q   */
```

0-2 X'010681' NS header

3 bits 0-3, format:

- 0000 Format 0: specifies a subset of the parameters shown in Format 1 of INIT-SELF (described separately, because the NS header differs in the first byte), with the receiver supplying default values

bit 4, reserved

bits 5-6, 00 DLU is PLU; 01 DLU is SLU

bit 7, 0 initiate only (I); do not enqueue. 1 initiate/enqueue (I/Q): enqueue the request if it cannot be satisfied immediately

4-11 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

12-m Uninterpreted Name of DLU

12 Type: X'F3' logical unit

13 Length, in binary, of DLU name

14-m EBCDIC character string

m+1-p Requester ID

m+1 Length, in binary, of requester ID

Note: X'00' = no requester ID

m+2-p Requester ID: the ID, in EBCDIC characters, of the end user initiating the request (May be used to establish the authority of the end user to access a particular resource.)

p+1-q Password

p+1 Length, in binary, of password

Note: X'00' = no password is present

p+2-q Password used to verify the identity of the end user

q+1-r User Field

q+1 Length, in binary, of user data

Note: X'00' = no user data is present

q+2-r User data: user-specific data that is passed to the primary LU on the CINIT request

q+2 User data key

X'00' structured subfields follow

¬X'00' first byte of unstructured user data

Note: Individual structured subfields may be omitted entirely. When present, they appear in ascending field number order.

- For unstructured user data

q+3-r Remainder of unstructured user data

- For structured user data

q+3-r Structured subfields (For detailed definitions, see the structured user data section on page E-129.)

Note: The following default values are supplied by the SSCP(ILU) receiving the Format 0 INIT-SELF request:

- Queuing conditions (if queuing is specified):
  - -- Enqueue if session count exceeded.
  - -- Enqueue this request FIFO.
- Initiate origin: network user is the initiator.
- NOTIFY: do not notify

INIT-SELF; ILU-->SSCP, Norm; FMD NS(s) (INITIATE-SELF)

```
DCL 1 INIT_SELF_FMT1_2_RQ      BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                      BIT(24), /*  0-2   */
      2 FORMAT                          BIT(8), /*   3    */
      2 TYPE                            BIT(8), /*   4    */
      2 DLU_QUEUING_CONDITIONS          BIT(8), /*   5    */
      2 INITIATE_ORIGIN                 BIT(8), /*   6    */
      2 NOTIFY_SPECIFICATIONS           BIT(8), /*   7    */
      2 MODE_NAME                      CHAR(8), /*  8-15  */
      2 DLU_UNINTRP_NAME_TYPE           BIT(8), /*  16    */
      2 DLU_UNINTRP_NAME_LENGTH         BIT(8), /*  17    */
      2 DLU_UNINTRP_NAME
          CHAR(REFER(DLU_UNINTRP_NAME_LENGTH)), /*  18-n  */
      2 REQUESTER_ID_LENGTH             BIT(8), /*  n+1   */
      2 REQUESTER_ID
            CHAR(REFER(REQUESTER_ID_LENGTH)), /* n+2-p  */
      2 PASSWORD_LENGTH                 BIT(8), /*  p+1   */
      2 PASSWORD  CHAR(REFER(PASSWORD_LENGTH)), /* p+2-q  */
      2 USER_DATA_LENGTH                BIT(8), /*  q+1   */
      2 USER_DATA
              CHAR(REFER(USER_DATA_LENGTH)), /* q+2-r  */
      2 URC_LENGTH                      BIT(8), /*  r+1   */
      2 URC             CHAR(REFER(URC_LENGTH)), /* r+2-s  */
      2 COS_NAME                       CHAR(8); /*s+1-s+8 */
```

```
0-2     X'810681' NS header
3       bits 0-3, format:
                  0001  Format 1:  specifies queuing,
                        initiate origin, NOTIFY, and URC
                        in addition to the parameters in
                        Format 0
                  0010  Format 2: specifies the COS name
                        field in addition to the
                        parameters in Format 1
        bits 4-7, reserved
4       Type:
        bits 0-1,  00  dequeue (DQ) a previously enqueued
                       initiate request (Note: Value 00 is
                       reserved if not Format 1.) (See
                       bits 2-3 for further specification
                       of setup actions.)
                   01  initiate only(I); do not enqueue
                   10  enqueue only (Q) (See byte 5 for
                       further specification of queuing
                       conditions.)
                   11  initiate/enqueue (I/Q): enqueue
                       the request if it cannot be
                       satisfied immediately
        bits 2-3, (used for DQ; otherwise, reserved)
                  00  leave on queue if setup attempt is
                      unsuccessful
                  01  remove from queue if setup attempt
                      is unsuccessful
                  10  remove from queue; do not attempt
                      setup
                  11  reserved
        bit 4, reserved
        bits 5-6, PLU/SLU specification:
                  00  DLU is PLU
                  01  DLU is SLU
        bit 7, reserved
5       Queuing conditions for DLU (When Type = DQ, bits
        0-7 are reserved.):
        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
        bit 2,  0  do not enqueue if CDINIT loses
                   contention
                1  enqueue if CDINIT loses contention
        bit 3,  0  do not enqueue if no SSCP(DLU)-DLU path
                1  enqueue if no SSCP(DLU)-DLU path
        bit 4, reserved
        bits 5-6, queuing position/service:
```

00 put this request at the bottom of the queue (the request is put at the bottom of the queue and serviced last)
01 enqueue this request FIFO
10 enqueue this request LIFO
11 reserved

bit 7, 0 do not enqueue for recovery retry
1 enqueue for recovery retry (The element is maintained on the recovery retry queue even after the activation of the session, so that the session can be retried in the event of a session failure.)

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 foot of the queue (FIFO).
- For "CDINIT contention" and "recovery retry," the default values are the same as those specified for the DLU (see bits 2 and 7 above).

6 INITIATE Origin:
bits 0-2, reserved
bit 3, (bit 3 is reserved when Type = DQ)
0 network user is the initiator
1 network manager is the initiator
bits 4-7, reserved

7 NOTIFY specifications:
bits 0-1, (bits 0 and 1 are reserved when Type = DQ)
00 do not notify LUs in session with DLU
01 notify all LUs in session with DLU that the ILU/OLU has requested a session with the DLU
10 notify LUs in session with DLU only if request is queued
11 reserved
bits 2-3, reserved
bit 4, 0 do not notify the ILU when the request is dequeued
1 notify the ILU when the request is dequeued
bits 5-7, 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 (When Type =

DQ, the Mode Name field is reserved.)

16-n Uninterpreted Name of DLU

16 Type: X'F3' logical unit

17 Length, in binary, of DLU name

18-n EBCDIC character string

n+1-p Requester ID

n+1 Length, in binary, of requester ID
Note: X'00' = no requester ID

n+2-p Requester ID: the ID, in EBCDIC characters, of the end user initiating the request (May be used to establish the authority of the end user to access a particular resource.)

p+1-q Password

p+1 Length, in binary, of password
Note: X'00' = no password is present

p+2-q Password used to verify the identity of the end user

q+1-r User Field (When Type = DQ, User field is reserved)

q+1 Length, in binary, of user data
Note: X'00' = no user data is present

q+2-r User data: user-specific data that is passed to the primary LU on the CINIT request

q+2 User data key
X'00' structured subfields follow
-X'00' first byte of unstructured user data
Note: Individual structured subfields may be omitted entirely. When present, they appear in ascending field number order.

- For unstructured user data

q+3-r Remainder of unstructured user data

- For structured user data

q+3-r Structured subfields (For detailed definitions, see the structured user data section on page E-129.)

r+1-s User Request Correlation (URC) Field (When Type = DQ, the URC must be the same as in the original INIT-SELF request.)

r+1 Length, in binary, of URC
Note: X'00' = no URC

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

End of Format 1; Format 2 Continues

s+1-s+8 COS name: symbolic name of class of service in EBCDIC characters (A value of eight space characters may be specified; in this case, the COS name is derived from the mode name table using the mode name received in bytes 8-15.)

INOP; PU_T4|5-->SSCP, PU-->PUCP, Norm; FMD NS(c) (INOPERATIVE)

```
DCL 1 INOP_RQ                          BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                            BIT(24), /*  0-2   */
      2 INOP_LINK_OR_ALS_ADDRESS             BIT(16), /*  3-4   */
      2 FORMAT                                BIT(4), /*   5    */
      2 INOP_REASON                           BIT(4),
      2 X21_CALL_PROG_SIG_LAST_RCVD          BIT(16); /*  6-7   */
```

0-2 X'010281' NS header

3-4 Network address of an inoperative (1) link or (2) adjacent link station

5 bits 0-3, format: X'0' (only value defined)
bits 4-7, reason:

- X'1' adjacent link station: loss of contact, unexpected loss of connection, or connection establishment failure
- X'2' link: link failure
- X'3' adjacent link station: discontact--loss of synchronization
- X'4' adjacent link station: incomplete discontact--loss of synchronization
- X'5' adjacent link station: request resynchronization--unexpected request for resynchronization
- X'6' adjacent link station (IPL or DUMP in progress)
- X'7' adjacent link station (RPO in progress)
- X'A' link: CCITT X.21 call establishment failure; X.21 call progress signals were received but are not included in bytes 6-7
- X'B' link: CCITT X.21 outgoing call establishment failure because of DCE signalling DCE clear condition
- X'C' link: CCITT X.21 outgoing call establishment failure because of expiration of time-out on changing DCE conditions
- X'D' link: unexpected loss of connection during the CCITT X.21 call phase
- X'E' link: failure during the CCITT X.21 call clearing phase
- X'F' link: CCITT X.21 outgoing call establishment failure; X.21 call progress signals were received--the signal is included in bytes 6-7

6-7 The CCITT X.21 call progress signal last received--included only if byte 5, bits 4-7 = X'F'; otherwise, these bytes are omitted (The codes and meanings of these X.21 call progress signals are as described in the CCITT recommendation X.21.)

IPLFINAL; SSCP-->PU_T4|5, Norm; FMD NS(c) (IPL FINAL)

```
DCL 1 IPLFINAL_RQ                BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 ALS_ADDRESS                      BIT(16), /*  3-4   */
      2 ENTRY_POINT                      BIT(32); /*  5-8   */
```

0-2 X'010205' NS header
3-4 Network address of adjacent link station associated with the node being loaded
5-8 Entry point location within load module

IPLINIT; SSCP-->PU_T4|5, Norm; FMD NS(c) (IPL INITIAL)

```
DCL 1 IPLINIT_RQ                 BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 ALS_ADDRESS                      BIT(16); /*  3-4   */
```

0-2 X'010203' NS header
3-4 Network address of adjacent link station associated with the node to be loaded

IPLTEXT; SSCP-->PU_T4|5, Norm; FMD NS(c) (IPL TEXT)

```
DCL 1 IPLTEXT_RQ                 BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 ALS_ADDRESS                      BIT(16), /*  3-4   */
      2 TEXT                             CHAR(*); /*  5-n   */
```

0-2 X'010204' NS header
3-4 Network address of adjacent link station associated with the node to be loaded
5-n Text: a variable-length byte-string in the form required by the node being loaded

LCP; PU_T4|5-->SSCP, PU_T4-->PUCP, Norm; FMD NS(c) (LOST CONTROL POINT)

```
DCL 1 LCP_RQ                     BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 REASON                            BIT(8), /*   3    */
      2 RESERVED                          BIT(8), /*   4    */
      2 SSCP_SUBAREA_ADDRESS             BIT(32), /*  5-8   */
      2 SSCP_ELEMENT_ADDRESS             BIT(16); /*  9-10  */
```

0-2 X'410287' NS header
3 Reason code, specifying why LCP was generated:

X'07' virtual route inoperative: VR_INOP received for the virtual route used by the (SSCP,PU) session (where the SSCP is the lost control point identified later, and the PU is the originator of the LCP)

X'0A' forced deactivation of the (SSCP,PU) session (DACTPU(¬SON) received by the PU)

X'0B' virtual route deactivated: NC_DACTVR(Forced) received for the virtual route used by the (SSCP,PU) session (where the SSCP is the lost control point identified later and the PU is the originator of the LCP)

X'0C' SSCP failure: the session between this PU and the identified SSCP was reset because of an abnormal termination of the SSCP (DACTPU(SON,Cause = X'0C') was received by the PU)

4 Reserved

5-10 Network address of the lost control point (SSCP)

5-8 Subarea address of the lost control point

9-10 Element address of the lost control point

LDREQD; PU_T2-->SSCP, Norm; FMD NS(c) (LOAD REQUIRED)

```
DCL 1 LDREQD_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 LOAD_MODULE                      CHAR(8), /*  3-10  */
      2 RESERVED                          BIT(7), /*   11   */
      2 ADJ_PU_LOAD_CAPABILITY            BIT(1);
```

0-2 X'410237' NS header

3-10 IPL load module: an eight-character EBCDIC symbolic name of the IPL load module requested:
X'4040...40' any load module will be accepted
¬X'4040...40' specific load module specified

11 bits 0-6, reserved
bit 7, adjacent PU load capability (initialized to 0 by the PU_T2):
0 the adjacent PU is unable to load the PU_T2
1 the adjacent PU can load the PU_T2 (set by the boundary function in the adjacent subarea node)

LSA; PU_T4|5-->PU_T4|5, Exp; NC (LOST SUBAREA)

```
DCL 1 LSA_RQ                     BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                          BIT(8), /*   0    */
      2 RESERVED                        BIT(16), /*  1-2   */
      2 REASON                           BIT(8), /*   3    */
      2 FORMAT                           BIT(8), /*   4    */
      2 RESERVED                        BIT(16), /*  5-6   */
      2 PU_ADDRESS                      BIT(16), /*  7-8   */
```

```
          2 SUBAREAS(*),
            3 RESERVED                          BIT(16), /*9-10+4n */
            3 SUBAREA_ADDRESS                    BIT(8), /* 11+4n  */
            3 RESERVED                           BIT(8); /* 12+4n  */
```

0	X'05' request code
1-2	Reserved
3	Reason code, specifying why LSA was originated: X'01' unexpected routing interruption X'02' controlled routing interruption
4	Format: X'01' (only value defined)
5-8	Origination Address
5-6	Reserved
7-8	Network address of the PU that originated the LSA
9-12	Lost Subarea Address Field
9-10	Reserved
11	Subarea address (left-justified) for a lost subarea
12	Reserved
13-n	Additional 4-byte fields in the form of bytes 9-12, corresponding to additional lost subareas

LUSTAT; LU-->LU|SSCP, Norm; DFC (LOGICAL UNIT STATUS)

```
          DCL 1 LUSTAT_RQ                BASED(ADDR(RU)), /* Byte(s)*/
                2 RQ_CODE                         BIT(8), /*   0    */
                2 STATUS                         BIT(32); /*  1-4   */
```

0	X'04' request code
1-4	Status value + status extension field (two bytes each): X'0000'+'uuuu' user status (no system-defined status) + user-defined field X'0001'+'ccdd' component now available + component identification (see Note) X'0002'+'rrrr' sender will have no (more) FMD requests to transmit during the time that this session remains active + reserved field X'0003'+'ccdd' component entering attended mode of operation + component identification (see Note) X'0004'+'ccdd' component entering unattended mode of operation + component identification (see Note) X'0005'+'iiii' prepare to commit all resources required for the unit of work + information field: X'0001' request End Bracket be sent on next chain (only value defined) X'0006'+'rrrr' no-op (used to allow an RH to be sent when no other request is

available or allowed) + reserved field

X'0007'+'rrrr' sender currently has no FMD requests to transmit (but may have later during the time that this session remains active) + reserved field

X'0801'+'ccdd' component not available (e.g., not configured) + component identification (see Note)

X'0802'+'ccdd' component failure (intervention required) + component identification (see Note)

X'081C'+'ccdd' component failure (permanent error) + component identification (see Note)

X'0824'+'rrrr' function canceled + reserved field

X'082B'+'ccdd' component available, but presentation space integrity lost + component identification (see Note)

X'0831'+'ccdd' component disconnected (power off or some other disconnecting condition) + component identification (see Note)

X'0848'+'rrrr' cryptography component failure + reserved field

X'400A'+'ssss' no-response mode not allowed + sequence number of the request specifying no-response

Note: Values for cc byte are:

X'00' LU itself rather than a specific LU component (For this cc value, dd=X'00'.)

X'FF' The dd byte specifies the LU component medium class and device address. (See SNA LU-LU Session Types for definitions of these terms and usage of the values according to LU-LU session type.)

-X'(00|FF)' LU component medium class and device address (For these cc values, dd=X'00'.)

NC_ACTVR; PU_T4|5-->PU_T4|5, Exp; NC (ACTIVATE VIRTUAL ROUTE)

```
DCL 1 NC_ACTVR_RQ              BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                        BIT(8), /*   0    */
      2 RESERVED                      BIT(16), /*  1-2   */
      2 FORMAT                         BIT(8), /*   3    */
      2 RESERVED                       BIT(8), /*   4    */
      2 RCV_ERN_MASK                  BIT(16), /*  5-6   */
      2 SEND_ERN_MASK                 BIT(16), /*  7-8   */
```

```
        2 RESERVED                          BIT(4), /*   9     */
        2 VR_SEND_SEQ_NO                   BIT(12), /*  9-10   */
        2 RESERVED                          BIT(8), /*   11    */
        2 MAX_WINDOW_SIZE                   BIT(8), /*   12    */
        2 RESERVED                          BIT(8), /*   13    */
        2 MIN_WINDOW_SIZE                   BIT(8), /*   14    */
        2 MAX_SEND_PIU_LENGTH              BIT(16), /* 15-16   */
        2 MAX_RCV_PIU_LENGTH               BIT(16); /* 17-18   */
```

0	X'0D' request code
1-2	Reserved
3	Format: X'01' (only value defined)
4	Reserved
5-6	Receive ERN mask: a bit is <u>on</u> if that ERN can be used to send PIUs to NC_ACTVR originator; multiple bits may be set to 1 (bit 0 corresponds to reverse ERN 0, bit 1 to reverse ERN 1, and so forth)
7-8	Send ERN mask: a bit is <u>on</u> if that ERN can be used to send PIUs from the NC_ACTVR originator: exactly one bit is set to 1 (bit 0 corresponds to ERN 0, bit 1 to ERN 1, and so forth)
9-10	bits 0-3, reserved bits 4-15, initial VR send sequence number
11	Reserved
12	Maximum window size permitted on the VR
13	Reserved
14	Minimum window size permitted on the VR
15-16	Maximum PIU size permitted to be sent by the NC_ACTVR originator: X'0000' no restriction (only value defined)
17-18	Maximum PIU length permitted to be received by the NC_ACTVR originator: X'0000' no restriction (only value defined) <u>Note</u>: The NC_ER_ACT and NC_ER_ACT_REPLY RUs accumulate the maximum PIU size permitted to flow in each direction of the ER. NC_ACTVR communicates these limits to the other end of the VR.

NC_DACTVR; PU_T4|5-->PU_T4|5, Exp, NC (DEACTIVATE VIRTUAL ROUTE)

```
DCL 1 NC_DACTVR_RQ              BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                          BIT(8), /*   0     */
      2 RESERVED                        BIT(16), /*  1-2    */
      2 FORMAT                           BIT(8), /*   3     */
      2 TYPE                             BIT(8); /*   4     */
```

0	X'0E' request code
1-2	Reserved
3	Format: X'01'
4	Type X'01' orderly: receiver of NC_DACTVR to deactivate the VR if there are no sessions on the VR

X'02' forced: receiver of NC_DACTVR to deactivate the VR even if there are sessions on the VR; it also results in session outage notification for sessions using the VR

NC_ER_ACT; PU_T4|5-->PU_T4|5, Exp; NC (EXPLICIT ROUTE ACTIVATE)

```
DCL 1 NC_ER_ACT_RQ              BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                        BIT(8), /*   0    */
      2 RESERVED                      BIT(16), /*  1-2   */
      2 FORMAT                         BIT(8), /*   3    */
      2 RESERVED                       BIT(8), /*   4    */
      2 ER_LENGTH                      BIT(8), /*   5    */
      2 MAX_ER_LENGTH                  BIT(8), /*   6    */
      2 DESTINATION_SA                BIT(32), /*  7-10  */
      2 DYNAMIC_ER_DEFN                BIT(1), /*   11   */
      2 RESERVED                      BIT(11), /* 11-12  */
      2 ER_NUM                         BIT(4),
      2 ORIGINATING_SA                BIT(32), /* 13-16  */
      2 REV_ERN_MASK                  BIT(16), /* 17-18  */
      2 MAX_PIU_SIZE                  BIT(16), /* 19-20  */
      2 RESERVED                      BIT(64), /* 21-28  */
      2 ACT_SEQ_ID                    CHAR(8); /* 29-36  */
```

0	X'0B' request code
1-2	Reserved
3	Format: X'01' (only value defined)
4	Reserved
5	Explicit route length: initially set to 0 at the originating PU, incremented by 1 at each receiver of the original or propagated NC_ER_ACT
6	Maximum ER length, as specified by the request originator
7-10	Subarea address of the destination PU corresponding to the ERN specified in byte 12, bits 4-7
11	bit 0, route definition capability of RU sender: 0 RU sender does not allow route usage except by explicit installation definition 1 RU sender allows route usage without requiring explicit installation definition bits 1-7, reserved
12	bits 0-3, reserved bits 4-7, ERN of the explicit route being activated
13-16	Subarea address of the PU that originated the NC_ER_ACT request
17-18	Reverse ERN mask: a bit is <u>on</u> if the corresponding ERN can be used to route to the originating subarea (bit 0 corresponds to ERN 0, bit 1 to ERN 1 and so forth)

19-20 Maximum PIU length allowed on the ER in the direction of flow of this NC_ER_ACT:
X'0000' no restriction (only value defined)
21-28 Reserved
29-36 Activation request sequence identifier: an 8-byte binary value, generated by the originator of NC_ER_ACT, and included by the destination node in NC_ER_ACT_REPLY to correlate an NC_ER_ACT with its corresponding NC_ER_ACT_REPLY (The 8-byte field has the following characteristic: If n1 was generated at time t1, and n2 was generated at time t2, then t1 < t2 implies n1 < n2.)

NC_ER_ACT_REPLY; PU_T4|5-->PU_T4|5, Exp; NC (EXPLICIT ROUTE ACTIVATE REPLY)

```
DCL 1 NC_ER_ACT_REPLY_RQ        BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                        BIT(8), /*   0    */
      2 RESERVED                      BIT(16), /*  1-2   */
      2 FORMAT                         BIT(8), /*   3    */
      2 TYPE                           BIT(8), /*   4    */
      2 ER_LENGTH                      BIT(8), /*   5    */
      2 MAX_ER_LENGTH                  BIT(8), /*   6    */
      2 DESTINATION_SA                BIT(32), /*  7-10  */
      2 RESERVED                      BIT(12), /* 11-12  */
      2 ER_NUM                         BIT(4),
      2 ORIGINATING_SA                BIT(32), /* 13-16  */
      2 REV_ERN_MASK                  BIT(16), /* 17-18  */
      2 MAX_PIU_SIZE                  BIT(16), /* 19-20  */
      2 MAX_PIU_SIZE_FROM_ACTIVATE    BIT(16), /* 21-22  */
      2 RESERVED                      BIT(48), /* 23-28  */
      2 ACT_SEQ_ID                    CHAR(8), /* 29-36  */
      2 RESERVED                      BIT(16), /* 37-38  */
      2 REPLY_SA                      BIT(32), /* 39-42  */
      2 TG_ADJ_SA                     BIT(32), /* 43-46  */
      2 TG_NUM                         BIT(8), /*   47   */
      2 RESERVED                       BIT(8); /*   48   */
```

0 X'0C' request code
1-2 Reserved
3 Format: X'01' (only value defined)
4 Type
X'00' explicit route activated
X'01' race condition resulting from NC_ER_ACT being sent by both nodes, each of which allows routing usage without requiring explicit installation definition; this condition is resolved in favor of the NC_ER_ACT from the PU having the greater subarea address (thus, this Type code is sent by the PU having the larger subarea address)
X'02' ER is not reversible since there is no reverse ERN defined

X'03' encountered a PU that does not support ER and VR protocols
X'04' ER length exceeded the maximum specified in NC_ER_ACT
X'05' ER requires a TG that is not active
X'06' ER is not defined in the NC_ER_ACT_REPLY originating node

5	Explicit route length, in terms of the number of transmission groups in the explicit route as accumulated by NC_ER_ACT
6	Maximum ER length, as specified in NC_ER_ACT request
7-10	Subarea address of the destination PU of corresponding NC_ER_ACT
11	Reserved
12	bits 0-3, reserved bits 4-7, ERN of the ER being activated
13-16	Subarea address of the PU originating the corresponding NC_ER_ACT
17-18	Reverse ERN mask: a bit is on if the corresponding ERN can be used to route to the NC_ER_ACT originating subarea (bit 0 corresponds to ERN 0, bit 1 to ERN 1, and so forth) X'0000' no restriction (only value defined)
19-20	Maximum size of PIU allowed to flow on the reverse ERNs specified in bytes 17-18: X'0000' no restriction (only value defined)
21-22	Maximum PIU length accumulated by the NC_ER_ACT: X'0000' no restriction (only value defined)
23-28	Reserved
29-36	Activation request sequence identifier: same value as specified in the corresponding NC_ER_ACT
37-38	Reserved
39-42	Subarea address of the node that originated this NC_ER_ACT_REPLY
43-46	Subarea address depending on the Type field (byte 4), as follows:

Type	Contents of this field
X'00'	reserved
X'01'	reserved
X'02'	subarea on the ER prior to that with no reverse ERN defined
X'03'	subarea that does not support ER and VR protocols
X'04'	subarea on the ER preceding the subarea where the explicit route length (byte 5 of NC_ER_ACT) is incremented to a value one more than the maximum ER length limit (byte 6)
X'05'	subarea on the other end of the TG that is not active
X'06'	subarea on the ER from which the PU (that does not have the ER defined) received the

corresponding NC_ER_ACT

47	TGN of the TG between the subareas specified in bytes 39-42 and 43-46; reserved if Type is X'00'or X'01'
48	Reserved

NC_ER_INOP; PU_T4|5-->PU_T4|5, Exp; NC (EXPLICIT ROUTE INOPERATIVE)

```
DCL 1 NC_ER_INOP_RQ              BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                          BIT(8), /*   0    */
      2 RESERVED                        BIT(16), /*  1-2   */
      2 FORMAT                           BIT(8), /*   3    */
      2 REASON_CODE                      BIT(8), /*   4    */
      2 ORIGINATING_SA                  BIT(32), /*  5-8   */
      2 TG_ADJ_SA                       BIT(32), /*  9-12  */
      2 TG_NUM                           BIT(8), /*   13   */
      2 CNT_ER_FIELD                     BIT(8), /*   14   */
      2 ER_FIELD(1:REFER(CNT_ER_FIELD)),
        3 SA                            BIT(32), /*15-18+6n*/
        3 MASK                          BIT(16); /*19-20+6n*/
```

0	X'06' request code
1-2	Reserved
3	Format: X'01' (only value defined)
4	Reason code: X'01' unexpected routing interruption over a transmission group, such as the failure of the last active link in the TG X'02' controlled routing interruption, such as the result of a DISCONTACT
5-8	Subarea address of the PU that originated the NC_ER_INOP
9-12	Subarea address on other end of the transmission group that had the routing interruption
13	TG number of the transmission group that had the routing interruption
14	Number of destination subareas that are on the ERs using the above TG
15-20	Inoperative ER Field
15-18	Subarea address of a destination that is routed to using an ER requiring the TG that had the routing interruption
19-20	Inoperative explicit route mask: a bit is on if the ER of the corresponding ERN is inoperative (bit 0 corresponds to ERN 0, bit 1 corresponds to ERN 1, and so forth)
21-n	Any additional six-byte entries in the same format as bytes 15-20

NC_ER_OP; PU_T4|5-->PU_T4|5, Exp; NC (EXPLICIT ROUTE OPERATIVE)

```
DCL 1 NC_ER_OP_RQ                BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                            BIT(8), /*   0    */
      2 RESERVED                          BIT(16), /*  1-2   */
      2 FORMAT                             BIT(8), /*   3    */
      2 RESERVED                           BIT(8), /*   4    */
      2 ORIGINATING_SA                    BIT(32), /*  5-8   */
      2 TG_ADJ_SA                         BIT(32), /*  9-12  */
      2 TG_NUM                             BIT(8), /*   13   */
      2 CNT_ER_FIELD                       BIT(8), /*   14   */
      2 ER_FIELD(1:REFER(CNT_ER_FIELD)),
        3 SA                              BIT(32), /*15-18+6n*/
        3 MASK                            BIT(16); /*19-20+6n*/
```

0	X'0F' request code
1-2	Reserved
3	Format: X'01' (Only value defined)
4	Reserved
5-8	Subarea address of the PU that originated the NC_ER_OP
9-12	Subarea address on other end of the operational TG
13	TG number of the operational TG
14	Number of destination subareas that are routed to using the ERs requiring the above TG
15-20	Operative ER Field Note: This field is included if at least one operative ER exists for the subarea in bytes 15-18.
15-18	Subarea address of a destination that is routed to using an ER requiring the above TG
19-20	Operative explicit route mask: a bit is on if the ER for the corresponding ERN is operative (bit 0 corresponds to ERN 0, bit 1 to ERN 1, and so forth)
21-n	Any additional six-byte field entries in the same format as bytes 15-20

NC_ER_TEST; PU_T4|5-->PU_T4|5, Exp; NC (EXPLICIT ROUTE TEST)

```
DCL 1 NC_ER_TEST_RQ              BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                            BIT(8), /*   0    */
      2 RESERVED                          BIT(16), /*  1-2   */
      2 FORMAT                             BIT(8), /*   3    */
      2 RESERVED                           BIT(8), /*   4    */
      2 ER_LENGTH                          BIT(8), /*   5    */
      2 MAX_ER_LENGTH                      BIT(8), /*   6    */
      2 DESTINATION_SA                    BIT(32), /*  7-10  */
      2 RESERVED                          BIT(12), /* 11-12  */
      2 ER_NUM                             BIT(4),
      2 ORIGINATING_SA                    BIT(32), /* 13-16  */
      2 REV_ERN_MASK                      BIT(16), /* 17-18  */
      2 MAX_PIU_SIZE                      BIT(16), /* 19-20  */
      2 RESERVED                          BIT(16), /* 21-22  */
```

```
          2 ORIGINATING_SSCP                BIT(48), /* 23-28  */
          2 RQ_CORRELATION                  CHAR(10); /* 29-38  */
```

0	X'09' request code
1-2	Reserved
3	Format: X'01' (only value defined)
4	Reserved
5	Explicit route length: initially set to zero by the PU that originated the NC_ER_TEST, incremented by one at each receiver of the original or propagated NC_ER_TEST
6	Maximum ER length (number of TGs comprising the ER), specified by the request originator
7-10	Subarea address of the destination of ER corresponding to the ERN specified in byte 12, bits 4-7
11	Reserved
12	bits 0-3, reserved bits 4-7, ERN of the explicit route being tested
13-16	Subarea address of the PU that originated the NC_ER_TEST
17-18	Reverse ERN mask: a bit is on if the corresponding ERN can be used to route to the originating subarea (Bit 0 corresponds to ERN 0, bit 1, to ERN 1 and so forth.)
19-20	Maximum size of PIU allowed on the ERN specified in byte 12, bits 4-7: X'00' no restriction (only value defined)
21-22	Reserved
23-28	Network address of the SSCP that originated the corresponding NS request
29-38	Request correlation field: an implementation defined value, which is returned in NC_ER_TEST_REPLY for correlation of reply to request

NC_ER_TEST_REPLY; PU_T4|5-->PU_T4|5, EXP; NC (EXPLICIT ROUTE TEST REPLY)

```
          DCL 1 NC_ER_TEST_REPLY_RQ      BASED(ADDR(RU)), /* Byte(s)*/
                2 RQ_CODE                       BIT(8), /*   0    */
                2 RESERVED                     BIT(16), /*  1-2   */
                2 FORMAT                        BIT(8), /*   3    */
                2 TYPE                          BIT(8), /*   4    */
                2 ER_LENGTH                     BIT(8), /*   5    */
                2 MAX_ER_LENGTH                 BIT(8), /*   6    */
                2 DESTINATION_SA               BIT(32), /*  7-10  */
                2 RESERVED                     BIT(12), /* 11-12  */
                2 ER_NUM                        BIT(4),
                2 ORIGINATING_SA               BIT(32), /* 13-16  */
                2 REV_ERN_MASK                 BIT(16), /* 17-18  */
                2 MAX_PIU_SIZE                 BIT(16), /* 19-20  */
                2 MAX_PIU_SIZE_FROM_TEST       BIT(16), /* 21-22  */
                2 ORIGINATING_SSCP             BIT(48), /* 23-28  */
```

```
      2 RQ_CORRELATION                CHAR(10), /* 29-38  */
      2 REPLY_SA                       BIT(32), /* 39-42  */
      2 TG_ADJ_SA                      BIT(32), /* 43-46  */
      2 TG_NUM                          BIT(8); /*   47   */
```

0	X'0A' request code
1-2	Reserved
3	Format: X'01' (only value defined)
4	Type: X'00' The corresponding NC_ER_TEST reached its destination subarea X'02' ER not reversible since there is no reverse ERN defined X'03' encountered a PU that does not support ER and VR protocols X'04' ER length exceeded the limit specified in the NC_ER_TEST request X'05' ER requires a TG that is not active X'06' ER is not defined in the NC_ER_TEST_REPLY originating node
5	Explicit route length, in terms of number of the transmission groups in the explicit route as accumulated in NC_ER_TEST.
6	Maximum ER length, as specified in the NC_ER_TEST request
7-10	Subarea address of the destination PU for corresponding NC_ER_TEST
11	Reserved
12	bits 0-3, reserved bits 4-7, ERN of the ER being tested
13-16	Subarea address of the PU that originated the corresponding NC_ER_TEST
17-18	Reverse ERN mask: a bit is <u>on</u> if the corresponding ERN can be used to route to the originating subarea
19-20	Maximum PIU size permitted on the reverse ERN specified in bytes 17-18: X'0000' no restriction (only value defined)
21-22	Maximum PIU size accumulated by the NC_ER_TEST: X'0000' no restriction (only value defined)
23-28	Network address of the SSCP originating the corresponding NS test request
29-38	Request correlation field: same value as specified in the corresponding NC_ER_TEST
39-42	Subarea address of the PU that originated this NC_ER_TEST_REPLY
43-46	Subarea address depending on the type field (byte 4) as follows:

<u>Type</u>	<u>Contents of this field</u>
X'00'	reserved
X'02'	subarea on the ER prior to that with no reverse ERN defined

X'03' subarea that does not support ER and VR protocols
X'04' subarea on the ER preceding the subarea where the explicit route length (byte 5 of NC_ER_TEST) is incremented to a value one more than the maximum ER length limit (byte 6)
X'05' subarea on the other end of the TG that is not active
X'06' subarea on the ER from which the PU (that does not have the ER defined), received the corresponding NC_ER_TEST

47 TGN of the TG between the subareas specified in bytes 39-42 and 43-46; reserved if Type is X'00'

NC_IPL_ABORT; PU_T4|5-->PU_T2, Exp; NC (NC IPL ABORT)

```
DCL 1 NC_IPL_ABORT_RQ          BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                        BIT(8), /*   0    */
      2 SENSE_DATA                    BIT(32); /*  1-4   */
```

0 X'46' request code
1-4 Sense data

NC_IPL_FINAL; PU_T4|5-->PU_T2, Exp; NC (NC IPL FINAL)

```
DCL 1 NC_IPL_FINAL_RQ          BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                        BIT(8), /*   0    */
      2 ENTRY_POINT                   BIT(32); /*  1-4   */
```

0 X'02' request code
1-4 Entry point location (hexadecimal address) within load module

NC_IPL_INIT; PU_T4|5-->PU_T2, Exp; NC (NC IPL INITIAL)

```
DCL 1 NC_IPL_INIT_RQ           BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                        BIT(8), /*   0    */
      2 RESERVED                       BIT(8), /*   1    */
      2 LOAD_MODULE                   CHAR(8); /*  2-9   */
```

0 X'03' request code
1 Reserved
2-9 IPL load module: an eight-character EBCDIC symbolic name of the IPL load module to be transmitted

```
NC_IPL_TEXT; PU_T4|5-->PU_T2, Exp; NC (NC IPL TEXT)

          DCL 1 NC_IPL_TEXT_RQ            BASED(ADDR(RU)), /* Byte(s)*/
                2 RQ_CODE                          BIT(8), /*   0    */
                2 IPL_TEXT                       CHAR(*); /*  1-n   */

          0          X'04' request code
          1-n        Text: a  variable-length byte-string of  IPL data,
                     where the maximum value of n is 255

NOTIFY; SSCP-->SSCP|LU, LU-->SSCP, Norm; FMD NS(s) (NOTIFY)

          DCL 1 NOTIFY_RQ                 BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                       BIT(24), /*  0-2   */
                2 VECTOR_KEY                       BIT(8), /*   3    */
                2 VECTOR_DATA                    CHAR(*); /* 4-end  */


          DCL 1 NOTIFY_VECTOR_01
                      BASED(ADDR(NOTIFY_RQ.VECTOR_DATA)), /* Byte(s)*/
                2 REQUESTED_LU_NTWK_NAME_TYPE      BIT(8), /*   4    */
                2 REQUESTED_LU_NTWK_NAME_LENGTH    BIT(8), /*   5    */
                2 REQUESTED_LU_NTWK_NAME
              CHAR(REFER(REQUESTED_LU_NTWK_NAME_LENGTH)), /*  6-m   */
                2 REQUESTING_LU_NTWK_NAME_TYPE     BIT(8), /*  m+1   */
                2 REQUESTING_LU_NTWK_NAME_LENGTH   BIT(8), /*  m+2   */
                2 REQUESTING_LU_NTWK_NAME
             CHAR(REFER(REQUESTING_LU_NTWK_NAME_LENGTH)); /* m+3-p  */


          DCL 1 NOTIFY_VECTOR_03
                      BASED(ADDR(NOTIFY_RQ.VECTOR_DATA)), /* Byte(s)*/
                2 STATUS                           BIT(8), /*   4    */
                2 PCID                            CHAR(8), /*  5-12  */
                2 REASON                           BIT(8), /*   13   */
                2 SENSE_DATA                      BIT(32), /* 14-17  */
                2 SESSION_KEY                      BIT(8), /*   18   */
                                                   /* See page E-127 */
                2 SESSION_KEY_CONTENT
                        CHAR(REFER(SESSION_KEY_LENGTH)), /*  19-n  */
                2 URC_LENGTH                       BIT(8), /*  n+1   */
                2 URC             CHAR(REFER(URC_LENGTH)); /* n+2-p  */


          DCL 1 NOTIFY_VECTOR_04
                      BASED(ADDR(NOTIFY_RQ.VECTOR_DATA)), /* Byte(s)*/
                2 TYPE                             BIT(8), /*   4    */
                2 CAUSE                            BIT(8), /*   5    */
                2 ACTION                           BIT(8), /*   6    */
                2 SESSION_KEY                      BIT(8), /*   7    */
                                                   /* See page E-127 */
                2 SESSION_KEY_CONTENT
                        CHAR(REFER(SESSION_KEY_LENGTH)), /*  7-n   */
                2 URC_LENGTH                       BIT(8), /*  n+1   */
```

```
      2 URC              CHAR(REFER(URC_LENGTH)); /* n+2-p  */


DCL 1 NOTIFY_VECTOR_0C
            BASED(ADDR(NOTIFY_RQ.VECTOR_DATA)), /* Byte(s)*/
      2 VECTOR_LENGTH                    BIT(8), /*   4    */
      2 PRI_LU_CAPABILITY                BIT(4), /*   5    */
      2 SEC_LU_CAPABILITY                BIT(4),
      2 LU_LU_SESSION_LIMIT             BIT(16), /*  6-7   */
      2 LU_LU_SESSION_COUNT             BIT(16), /*  8-9   */
      2 PARALLEL_SESSION_CAPABILITY      BIT(1), /*   10   */
      2 RESERVED                         BIT(7),
      2 MODE_TABLE_NAME                 CHAR(8); /* 11-18  */
```

0-2 X'810620' NS header (for SSCP-->LU and LU-->SSCP)

0-2 X'818620' NS header (for SSCP-->SSCP)

3 NOTIFY vector key:

- X'01' resource requested: used to send NOTIFY to the current users (LUs) of a resource (LU) to inform them that another LU wishes to use the resource
- X'03' ILU/TLU or third-party SSCP notification:
  - ILU/TLU notification: used to send NOTIFY to the issuer of an INIT or TERM request to give the status of the session
  - third-party SSCP notification: used to send NOTIFY to a third-party SSCP (the SSCP whose LU issued an INIT-OTHER or TERM-OTHER request) to give the status of the setup/takedown procedure
- X'04' LU notification: used to send NOTIFY to an LU informing it of the completed deactivation of the identified LU-LU session
- X'0C' LU-LU session services capabilities: used to send NOTIFY to the SSCP having an active session with the sending LU, to convey the current LU-LU session services capability of that LU

4-p NOTIFY Vector Data

- For NOTIFY vector key X'01':

4-m Network name of requested LU

4 Type: X'F3' logical unit

5 Length, in binary, of symbolic name of LU

6-m Symbolic name in EBCDIC characters

m+1-p Network name of requesting LU

m+1 Type: X'F3' logical unit

m+2 Length, in binary, of symbolic name

m+3-p Symbolic name in EBCDIC characters

- For NOTIFY vector key X'03':

4 Status:

- X'01' session terminated
- X'02' session initiated

X'03' procedure error
X'04' setup process started

5-12 PCID

5-6 Network address of the SSCP(ILU) or SSCP(TLU)

7-12 A unique 6-byte value, generated by the SSCP(ILU) or SSCP(TLU), that is used in all cross-domain requests dealing with the same setup or takedown procedure until it is completed

13 Reason (defined for Status value of X'03' only)

Note: There are two encodings of the Reason byte:

- If bit 4 = 0, then the Reason byte is encoded for a setup procedure error.
- If bit 4 = 1, then the Reason byte is encoded for a takedown procedure error.

Setup Procedure Error

bit 0, 1 CINIT error in reaching the PLU
bit 1, 1 BIND error in reaching the SLU
bit 2, 1 setup reject at the PLU
bit 3, 1 setup reject at the SLU
bit 4, 0 setup procedure error
bit 5, reserved
bit 6, 1 setup reject at SSCP
bit 7, reserved

Takedown Procedure Error

bit 0, 1 CTERM error in reaching the PLU
bit 1, 1 UNBIND error in reaching the SLU
bit 2, 1 takedown reject at the PLU
bit 3, 1 takedown reject at the SLU
bit 4, 1 takedown procedure error
bit 5, 1 takedown reject at the SSCP
bit 6, 0 see following Note
bit 7, reserved

Note: The bit combination of 11 for bits 4 and 6 is set aside for implementation internal use and will not be otherwise defined.

14-17 Sense data (defined for Status value of X'03' only)

18 Session key:
X'05' PCID
X'06' network name pair
X'07' network address pair
X'0A' URC

19-n Session Key Content

- For session key X'05': PCID

19-20 Network address of the SSCP(ILU)

21-26(=n) A unique 6-byte value, generated by the SSCP(ILU), that is retained and used in all cross-domain requests dealing with the same procedure until it is completed

Note: This session key is applicable within a NOTIFY only for SSCP-to-SSCP(TLU); it differs from the PCID carried in the NOTIFY Vector Data field (bytes 5-12) for NOTIFY vector key X'03'.

	• For session key X'06': network name pair
19	Type: X'F3' logical unit
20	Length, in binary, of symbolic name of PLU (or OLU or LU1)
21-m	Symbolic name in EBCDIC characters
m+1	Type: X'F3' logical unit
m+2	Length, in binary, of symbolic name of SLU (or DLU or LU2)
m+3-n	Symbolic name in EBCDIC characters
	• For session key X'07': network address pair
19-20	Network address of PLU
21-22(=n)	Network address of SLU
	• For session key X'0A': URC
19	Length, in binary, of the URC
20-n	URC: end user defined identifier Note: This session key is applicable within a NOTIFY only for SSCP-to-TLU; it is the URC carried as the session key in TERM, and differs from the URC in bytes n+1 through p.
n+1-p	User Request Correlation (URC) Field
n+1	Length, in binary, of the URC
n+2-p	URC: end user defined identifier, specified in an INIT or TERM request; used to correlate the given session to the initiating or terminating requests Note: The URC length is zero for SSCP-to-SSCP.
	• For NOTIFY Vector key X'04'
4	Type: X'01' session count decremented; no corresponding INIT-SELF X'02' session count decremented; corresponding INIT-SELF
5	Cause: cause of deactivating the (LU,LU) session, as specified in byte 4 of SESSEND
6	Action: any reactivation of the (LU,LU) session to be performed by either the PLU or SLU as specified in SESSEND or CDSESSEND
7	Session key: X'06' network name pair X'07' network address pair
8-n	Session Key Content
	• For session key X'06': network name pair
8	Type: X'F3' logical unit
9	Length, in binary, of symbolic name of PLU (or OLU or LU1)
10-m	Symbolic name in EBCDIC characters
m+1	Type: X'F3' logical unit
m+2	Length, in binary, of symbolic name of SLU (or DLU or LU2)
m+3-n	Symbolic name in EBCDIC characters
	• For session key X'07': network address pair
8-9	Network address of PLU
10-11(=n)	Network address of SLU
n+1-p	User Request Correlation (URC) Field
n+1	Length, in binary, of the URC

n+2-p	URC (from INIT-SELF, if Type = X'02'; otherwise, not included)
	• For NOTIFY Vector Key X'0C':
4	Length, in binary, of vector data field
5	bits 0-3, primary LU capability: 0000 cannot ever act as primary LU 0001 cannot currently act as primary LU 0010 reserved 0011 can now act as primary LU bits 4-7, secondary LU capability: 0000 cannot ever act as secondary LU 0001 cannot currently act as secondary LU 0010 reserved 0011 can now act as secondary LU
6-7	LU-LU session limit (where a value of zero means that no session limit is specified)
8-9	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
10	bit 0, parallel session capability: 0 parallel sessions not supported 1 parallel sessions supported bits 1-7, reserved
11-18(=p)	Mode table name: a symbolic name in EBCDIC characters Note: A value of all space (X'40') characters means that the mode table name is to be selected by the SSCP.

NS_IPL_ABORT; SSCP-->PU_T2, Norm; FMD NS(c) (NS IPL ABORT)

```
DCL 1 NS_IPL_ABORT_RQ          BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                      BIT(24), /*  0-2   */
      2 SENSE_DATA                     BIT(32); /*  3-6   */
```

0-2	X'410246' NS header
3-6	Sense data

NS_IPL_FINAL; SSCP-->PU_T2, Norm; FMD NS(c) (NS IPL FINAL)

```
DCL 1 NS_IPL_FINAL_RQ          BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                      BIT(24), /*  0-2   */
      2 ENTRY_POINT                    BIT(32); /*  3-6   */
```

0-2	X'410245' NS header
3-6	Entry point location (hexadecimal address) within load module

NS_IPL_INIT; SSCP-->PU_T2, Norm; FMD NS(c) (NS IPL INITIAL)

```
DCL 1 NS_IPL_INIT_RQ          BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                     BIT(24), /*  0-2   */
      2 RESERVED                       BIT(8), /*   3    */
      2 LOAD_MODULE                   CHAR(8); /*  4-11  */
```

0-2	X'410243' NS header
3	Reserved
4-11	IPL load module: eight-character EBCDIC symbolic name of the IPL load module to be transmitted

NS_IPL_TEXT; SSCP-->PU_T2, Norm; FMD NS(c) (NS IPL TEXT)

```
DCL 1 NS_IPL_TEXT_RQ          BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                     BIT(24), /*  0-2   */
      2 IPL_TEXT                      CHAR(*); /*  3-n   */
```

0-2	X'410244' NS header
3-n	Text: a variable-length byte-string of IPL data

NS_LSA; PU_T4|5-->SSCP, Norm; FMD NS(c) (NS LOST SUBAREA)

```
DCL 1 NS_LSA_RQ               BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                     BIT(24), /*  0-2   */
      2 REASON                         BIT(8), /*   3    */
      2 FORMAT                         BIT(8), /*   4    */
      2 RESERVED                      BIT(16), /*  5-6   */
      2 PU_ADDRESS                    BIT(16), /*  7-8   */
      2 SUBAREAS(*),
        3 RESERVED                    BIT(16), /*9-10+4n */
        3 SUBAREA_ADDRESS              BIT(8), /* 11+4n  */
        3 RESERVED                     BIT(8); /* 12+4n  */
```

0-2	X'010285' NS header Note: Bytes 3-n are identical to those in the originated or propagated LSA.
3	Reason code, specifying why LSA was originated: X'01' unexpected routing interruption X'02' controlled routing interruption
4	Format: X'01' (only value defined)
5-8	Origination Address
5-6	Reserved
7-8	Network address of the PU that originated the LSA
9-12	Lost Subarea Address Field
9-10	Reserved
11	Subarea address (left-justified) for a lost subarea
12	Reserved
13-n	Additional 4-byte fields in the form of bytes 9-12, corresponding to additional lost subareas

NSPE; SSCP-->ILU or TLU, Norm; FMD NS(s) (NS PROCEDURE ERROR)

```
DCL 1 NSPE_RQ                    BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2    */
      2 REASON                            BIT(8), /*   3     */
      2 VARIABLE_FORMAT                  CHAR(*); /*  4-n    */


DCL 1 NSPE_COMPREHENSIVE_FORM_RQ
          BASED(ADDR(NSPE_RQ.VARIABLE_FORMAT)), /* Byte(s)*/
      2 SENSE_DATA                       BIT(32), /*  4-7    */
      2 SESSION_KEY                       BIT(8), /*   8     */
                                          /* See page E-127 */
      2 SESSION_KEY_CONTENT              CHAR(*); /*  9-n    */


DCL 1 NSPE_CONDENSED_FORM_RQ
          BASED(ADDR(NSPE_RQ.VARIABLE_FORMAT)), /* Byte(s)*/
      2 PLU_UNINTRP_NAME_TYPE             BIT(8), /*   4     */
      2 PLU_UNINTRP_NAME_LENGTH           BIT(8), /*   5     */
      2 PLU_UNINTRP_NAME
          CHAR(REFER(PLU_UNINTRP_NAME_LENGTH)), /*  6-m    */
      2 SLU_UNINTRP_NAME_TYPE             BIT(8), /*  m+1    */
      2 SLU_UNINTRP_NAME_LENGTH           BIT(8), /*  m+2    */
      2 SLU_UNINTRP_NAME
          CHAR(REFER(SLU_UNINTRP_NAME_LENGTH)); /* m+3-n   */
```

0-2 X'010604' NS header

Note: The remainder of this RU has two formats: a comprehensive form and a condensed form, based upon the setting of bit 7 of the Reason byte (byte 3). The choice is implementation-dependent.

Comprehensive Format

3 Reason

Note: There are two encodings of the Reason byte in the comprehensive format:

- If bit 4 = 0, then the Reason byte is encoded for a setup procedure error.
- If bit 4 = 1, then the Reason byte is encoded for a takedown procedure error.

Setup Procedure Error

bit 0, 1 CINIT error in reaching the PLU
bit 1, 1 BIND error in reaching the SLU
bit 2, 1 setup reject at the PLU
bit 3, 1 setup reject at the SLU
bit 4, 0 setup procedure error
bit 5, reserved
bit 6, 1 setup reject at SSCP
bit 7, 1 comprehensive format of Reason byte

Takedown Procedure Error

bit 0, 1 CTERM error in reaching the PLU
bit 1, 1 UNBIND error in reaching the SLU
bit 2, 1 takedown reject at the PLU
bit 3, 1 takedown reject at the SLU

```
                bit 4,  1  takedown procedure error
                bit 5,  1  takedown reject at SSCP
                bit 6,  0  see following Note
                bit 7,  1  comprehensive format of Reason byte
                Note:  The bit combination of 11  for bits 4 and 6
                is set  aside for implementation internal  use and
                will not be otherwise defined.
        4-7     Sense data
        8       Session key:
                 X'06'  uninterpreted name pair
        9-n     Session Key Content
              • For session key X'06': uninterpreted name pair
        9       Type: X'F3' logical unit
        10      Length, in binary, of the PLU name
        11-m    EBCDIC character string
        m+1     Type: X'F3' logical unit
        m+2     Length, in binary, of the SLU name
        m+3-n   EBCDIC character string
                Condensed Format
        3       Reason:
                bit 0,  1  CINIT error in reaching the PLU
                bit 1,  1  BIND error in reaching the SLU
                bit 2,  1  setup reject at the PLU
                bit 3,  1  setup reject at the SLU
                bit 4,  1  takedown failure
                bit 5,  1  takedown reject at SSCP
                bit 6,  1  setup reject at SSCP
                bit 7,  0  condensed format
        4-m     Uninterpreted name of PLU
        4       Type: X'F3' logical unit
        5       Length, in binary, of PLU name
        6-m     EBCDIC character string
        m+1-n   Uninterpreted name of SLU
        m+1     Type: X'F3' logical unit
        m+2     Length, in binary, of SLU name
        m+3-n   EBCDIC character string

PROCSTAT; PU_T4|5-->SSCP, Norm; FMD NS(c) (PROCEDURE STATUS)

        DCL 1 PROCSTAT_RQ                BASED(ADDR(RU)), /* Byte(s)*/
              2 NS_HEADER                        BIT(24), /*  0-2   */
              2 RESERVED                         BIT(32), /*  3-6   */
              2 PU_ADDRESS                       BIT(16), /*  7-8   */
              2 PROCEDURE_TYPE                    BIT(8), /*   9    */
              2 PROCEDURE_STATUS                  BIT(8), /*   10   */
              2 RESERVED                         BIT(16), /* 11-12  */
              2 FAILING_NC_RQ_CODE                BIT(8), /*   13   */
              2 SENSE_DATA                       BIT(32); /* 14-17  */

        0-2     X'410236' NS header
        3-6     Reserved
        7-8     Network address of PU for  which the procedure was
                initiated
        9       Procedure type
```

```
                X'00'  load (only value defined)
        10      Procedure status:
                X'00'  successful (bytes 13-17 set to 0's)
                X'01'  reserved
                X'02'  failure occurred--procedure failure; bytes
                       13-17 contain additional information
        11-12   Reserved
        13-17   Status Qualifier
        13      Request code of failing NC RU
        14-17   Sense data returned in the -RSP for the failing NC
                RU

QC; LU-->LU, Norm; DFC (QUIESCE COMPLETE)

        DCL 1 QC_RQ                     BASED(ADDR(RU)), /* Byte(s)*/
              2 RQ_CODE                           BIT(8); /*   0    */


QEC; LU-->LU, Exp; DFC (QUIESCE AT END OF CHAIN)

        DCL 1 QEC_RQ                    BASED(ADDR(RU)), /* Byte(s)*/
              2 RQ_CODE                           BIT(8); /*   0    */

        0       X'80' request code

RECFMS; PU-->SSCP|PUCP, Norm; FMD NS(ma) (RECORD FORMATTED MAINTENANCE
        STATISTICS)

        DCL 1 RECFMS_RQ                 BASED(ADDR(RU)), /* Byte(s)*/
              2 NS_HEADER                         BIT(24), /*  0-2   */
              2 CNM_HEADER,
                3 TARGET_ID                       BIT(16), /*  3-4   */
                3 TARGET_ID_DESCRIPTOR            BIT(16), /*  5-6   */
                3 REQUEST_SPECIFIC_INFO            BIT(8), /*   7    */
              2 REQUEST_SPECIFIC_DATA             CHAR(*); /*  8-n   */

        0-2     X'410384' NS header
        3-7     CNM Header
        3-4     CNM target ID, as specified in bytes 5-6, bits 2-3
        5-6     bits 0-1, reserved
                bits 2-3, CNM target ID descriptor:
                          00  byte 4 contains a local address for
                              a PU or LU in a PU_T2 node or an
                              LSID for a PU or LU in a PU_T1
                              node; byte 3 is reserved
                          01  bytes 3-4 contain a network address
                              identifying a link, adjacent link
                              station, PU, or LU in the origin
                              subarea
                bits 4-15, procedure related identifier (PRID)
                          (see Note below)
        7       Request-Specific Information
                bit 0, solicitation indicator:
                        0  unsolicited request
```

1 reply request
bit 1, not last request indicator:
0 last request in a series of related unsolicited or reply requests, e.g., last reply request in a series corresponding to a single soliciting request
1 not last request
bits 2-7, request-specific type code (see below)

Note: For reply (i.e., solicited) requests, bytes 3-6 and byte 7, bits 2-7, echo the corresponding fields in the CNM header received in the request that solicited the reply request(s).

For unsolicited requests, these fields--the CNM target ID descriptor, the CNM target ID, the PRID, and the request-specific information--are generated by the request sender. For unsolicited requests, the PRID field contains X'000'.

7-n	Alert
7	bit 0, reserved bit 1, not last request indicator (see above) bits 2-7, type code: 000000; any defined CNM target id is valid
8-13	Node Identification bits 0-11, block number bits 12-31, ID number
12-13	Reserved
14-19	Alert Classification
14	bits 0-1, reserved bits 2-7, alert classification code: valid values are the same as the valid Type codes for RECFMS (byte 7, bits 2-7), with the exception of 000000
15	Subclassification identifier: the subclassification for the classification indicated in byte 14; if the RECFMS type identified by byte 14, bits 2-7, has a further qualification (e.g., RECFMS types 000011 and 000110 have qualifiers in byte 14 of their formats), this byte contains the qualifying value; if not, the byte is reserved
16-19	Alert reason mask: a mask field selecting the item(s) that caused the alert event to be originated; a bit value of 1 indicates that the corresponding data item was a reason for the alert event; if the RECFMS type identified by byte 14, bits 2-7, and byte 15 has a validity mask field, the format of the Alert Reason Mask field is the same as the format of the Validity Mask field (e.g., RECFMS 000011 bytes 15-17); if the identified RECFMS does not contain a validity mask, the i'th bit of this field corresponds to

the i'th data item in the identified RECFMS

20-n Appended RECFMS vector(s): zero or more RECFMS vectors may be appended to the request to convey data available to the CNMS when the alert event was originated, including data represented in RECFMS types; inclusion of RECFMS vectors is optional; appended vectors must be ordered according to the binary value of the Vector Type field (lowest value first)

20 Vector length: a binary count of the length in bytes of this RECFMS vector (bytes 21-m)

21 bit 0, criticality indicator: for certain vector types, an indication of the urgency of the event being reported; if bits 2-7 of this byte are not 000000, this bit is reserved; if bits 2-7 of this byte are 000000, the bit has the following values:

0 the event cited is noncritical

1 the event cited is potentially terminal; if the CNMA is unavailable, the SSCP will display this text

Note: When the criticality indicator is set to 1 in an appended vector, the appended vector (vector type 000000) contains a message formatted for display at an operator console and must occur as the first appended vector. Only one vector of type 000000 with the criticality indicator equal to 1 may be appended.

bit 1, reserved

bits 2-7, vector type: an identifier of the information contained in this RECFMS vector; valid values are:

000000 the vector contains a text message, composed of SCS characters

-000000 any valid type code for RECFMS (byte 7, bits 2-7), with the exception of 000000; these values indicate that the balance of the vector contains the information specified in bytes 14-n for the identified RECFMS type

Note: The sending of information in appended RECFMS vectors does not cause reset of any counters.

22-m Bytes 14-n of the indicated RECFMS type or the SCS text message

m+1-(n-1) Additional vectors (if required) having the same format as bytes 20-m

n X'00' indicating end of appended vectors

7-17 SDLC Test Command/Response Statistics

7 bit 0, solicitation indicator (see above)

	bit 1, not last request indicator (see above) bits 2-7, type code: 000001; the CNM target ID identifies a PU_T1|2
8-13	Node identification: bits 0-11, block number bits 12-31, ID number
12-13	Reserved
14-15	Counter: the number of times the secondary SDLC station has received an SDLC Test command with or without a valid FCS
16-17	Counter: the number of times the secondary SDLC station has received an SDLC Test command with a valid FCS and has transmitted an SDLC Test response Note: All counters are in binary.
7-22	Summary error data
7	bit 0, solicitation indicator (see above) bit 1, not last request indicator (see above) bits 2-7, type code: 000010; the CNM target ID identifies a PU
8-13	Node identification: bits 0-11, block number bits 12-31, ID number
12-13	Reserved
14-16	Summary counter validity mask:
14	bit 0, set to 1 if product error counter is valid bit 1, set to 1 if communication adapter error counter is valid bit 2, set to 1 if SNA negative response counter is valid bits 3-7, reserved
15-16	Reserved
17-18	Product error counter: a count for the product identified by the Node Identification field (bytes 8-13) of certain product-detected hardware errors whose origins are failures designated as internal by that product's own logic capability (The identified product has the responsibility for further isolation of these failures using its own product-specific problem determination and maintenance procedures.)
19-20	Communication adapter error counter for communication adapter errors whose source is either external or internal to the product identified by the block number
21-22	Count of SNA negative responses originating at this node Note: All counters are in binary.
7-30|31	Communication Adapter Error Statistics: counts of selected errors, useful for problem determination, that have been supplied by the communication adapter (For these errors, the RECFMS Type 000010 communication adapter error counter is always incremented; the RECFMS Type 000010 product error

counter is also incremented for those errors classified as internal errors by the product identified by the block number.)

7 bit 0, solicitation indicator (see above)
bit 1, not last request indicator (see above)
bits 2-7, type code: 000011; the CNM target ID identifies a PU_T1|2

8-13 Node identification:
bits 0-11, block number
bits 12-31, ID number

12-13 Reserved

14 Communication adapter error counter sets:
X'01' counter set 1
X'02' counter set 2
X'03' counter set 3

15-30 Data for Counter Sets 1 and 2

15-17 Communication adapter counter validity mask bytes

15 Mask byte 1:
bit 0, set to 1 if nonproductive time-out or receive overrun counter is valid
bit 1, set to 1 if idle time-out counter is valid
bit 2, set to 1 if write retry counter is valid
bit 3, set to 1 if overrun counter is valid
bit 4, set to 1 if underrun counter is valid
bit 5, set to 1 if connection problem counter is valid
bit 6, set to 1 if FCS error counter is valid
bit 7, set to 1 if primary station abort counter is valid

16 Mask byte 2:
bit 0, set to 1 if command reject counter is valid
bit 1, set to 1 if DCE error counter is valid
bit 2, set to 1 if write time-out counter is valid
bit 3, set to 1 if invalid status counter is valid
bit 4, set to 1 if communication adapter machine check counter is valid
bits 5-7, reserved

17 Reserved

18 Nonproductive time-out counter: no valid SDLC frames have been received within the time interval specified by the communication adapter; or receive overrun counter: the line is "hung" or insufficient buffer space has been allocated
Note: Receive overrun applies only to counter set 2.

19 Idle time-out counter: no SDLC Flag octets received for n seconds, where n is specified by the communication adapter

20 Write retry counter: the number of retransmissions of one or more SDLC I-frames

21 Overrun counter: the number of times one or more received characters have been overlaid

22 Underrun counter: the number of times one or more characters have been transmitted more than once

23	Connection problem counter: incremented by one for every n retries of commands that establish connection with a station, when RLSD drops, or whenever write retry is updated--n is specified by the communication adapter
24	FCS error counter: the number of times a received SDLC frame had an invalid FCS
25	Primary station abort counter: number of times eight or more consecutive one bits have been received
26	SDLC command reject counter
27	DCE error counter: number of DCE interrupts or other unexpected conditions (e.g., "data set ready" drops)
28	Write time-out counter: number of time-outs during write operations, e.g., because of transmit clock failures
29	Invalid status counter: number of times status generated by the adapter was not meaningful
30	Communication adapter machine check counter: number of times the communication adapter has been identified as causing a machine check Note: All counters are in binary.
15-31	Data for Counter Set 3
15-17	Communication adapter counter validity mask:
15	bit 0, set to 1 if total transmitted frames counter is valid bit 1, set to 1 if write retry counter is valid bit 2, set to 1 if total received frames counter is valid bit 3, set to 1 if FCS error counter is valid bit 4, set to 1 if command reject counter is valid bit 5, set to 1 if DCE error counter is valid bit 6, set to 1 if nonproductive time-out counter is valid bit 7, reserved
16-17	Reserved
18-19	Total transmitted frames counter: the total number of SDLC I-frames transmitted successfully
20-21	Write retry counter: the number of retransmissions of one or more SDLC I-frames
22-23	Total received frames counter: the number of SDLC I-frames successfully received
24-25	FCS error counter: the number of SDLC frames received with FCS errors
26-27	SDLC command reject counter
28-29	DCE error counter: the number of DCE interrupts and other unexpected conditions (e.g., "data set ready" drops)
30-31	Nonproductive time-out counter: the number of times an SDLC frame has not been received within the time interval specified by the adapter Note: All counters are in binary.
15-33	Data for Counter Set 4 (Note: For a definition of

adapter, control unit, and System/370 channel commands, and orders see implementation documentation.)

15-17 Adapter counter validity mask bytes

15 Mask byte 1: bit is set to 1 if the counter is valid
- bit 0, command-reject-while-not-initialized counter
- bit 1, command-not-recognized counter
- bit 2, sense-while-not-initialized counter
- bit 3, channel-parity-check-during-selection-sequence counter
- bit 4, channel-parity-check-during-data-write-sequence counter
- bit 5, output-parity-check-at-control-unit counter
- bit 6, input-parity-check-at-control-unit counter
- bit 7, input-parity-check-at-adapter counter

16 Mask byte 2:
- bit 0, data-error-at-adapter counter
- bit 1, data-stop-sequence counter
- bit 2, short-frame-or-length-check counter
- bit 3, connect-received-when-already-connected counter
- bit 4, disconnect-received-while-PU-active counter
- bit 5, long-RU counter
- bit 6, connect-parameter-error counter
- bit 7, Read-Start-Old-received counter

17 Reserved

18 Command-reject-when-not-initialized counter: an initial Control command containing a valid Connect order was not received prior to a Restart Reset, Read Start 0/1, Write Start 0/1, Read, Write, or Write Break command

19 Command-not-recognized counter: control unit channel adapter received a command code that it did not recognize (invalid or not supported)

20 Sense-when-not-initialized counter: Sense command was received in response to the initial asynchronous interrupt (device-end,unit check), or Sense command was received without a preceding unit check ending status

21 Channel-parity-check-during-selection-sequence counter: control unit channel adapter detected a parity error from the channel during the selection sequence from the channel

22 Channel-parity-check-during-data-write-sequence counter: control unit channel adapter detected a parity error on channel bus-out during a channel Write operation

23 Output-parity-check-at-control-unit counter: control unit channel adapter detected a control unit parity error during a channel Write operation

24 Input-parity-check-at-control-unit counter: control unit detected a control unit parity error

during a channel Read operation

25 Input-parity-check-at-adapter counter: control unit channel adapter detected that it transmitted bad parity on channel bus-in during a channel Read operation

26 Data-error-at-adapter counter: control unit detected a channel adapter error during an internal channel adapter cycle-steal operation

27 Data-stop-sequence counter: the number of data bytes accepted by the System/370's Read command was less than that specified in Connect

28 Short-frame-or-length-check counter: a minimum four bytes have not been transferred as a link header; or the byte count specified in the first two bytes of the header did not equal the number of bytes received during a Control, Write, or Write Break operation

29 Connect-received-when-already-connected counter: a Connect was received when the control unit was already connected; this is an error condition and the PU is deactivated

30 Disconnect-received-while-PU-active counter: a Disconnect order was received from the System/370 while the PU is active (i.e., with no DACTPU preceding the Disconnect); this is an error condition

31 Long-RU counter: primary link station has sent an RU greater than the secondary link station can accept

32 Connect-parameter-error counter: the Connect was rejected because it specified an odd-number buffer length, or it specified a buffer size insufficient to hold the link header, TH, RH, and at least a 64-byte RU

33 Read-Start-Old-received counter: the secondary link station received a Read Start Old command

Note: All counters are in binary.

7-n PU/LU Dependent Data

7 bit 0, solicitation indicator (see above)
bit 1, not last request indicator (see above)
bits 2-7, type code: 000100; the CNM target ID identifies a PU|LU

8-13 Node identification
bits 0-11, block number
bits 12-31, ID number

12-13 Reserved

14-n PU/LU dependent data

7-n Engineering Change Levels

7 bit 0, solicitation indicator (see above)
bit 1, not last request indicator (see above)
bits 2-7, type code: 000101; the CNM target ID identifies a PU

8-13 Node identification
bits 0-11, block number

```
                bits 12-31, ID number
          12-13 Reserved
          14-n  Implementation defined data describing hardware,
                microcode, and programming levels
          7-n   Link Connection Subsystem Data
          7     bit 0, solicitation indicator (see above)
                bit 1, not last request indicator (see above)
                bits 2-7, type code: 000110; the CNM target ID
                      identifies an adjacent link station in
                      the origin subarea
          8-13  Node identification
                bits 0-11, block number
                bits 12-31, ID number
          12-13 Reserved
          14    Data selection:
                 X'01' available data (only value defined)
          15    Link connection subsystem type:
                 X'01' IBM 3863, 3864, or 3865 modem (only value
                       defined)
          16-n  Link connection subsystem data: product defined
                data

RECMS;  PU_T4|5-->SSCP, Norm; FMD NS(ma) (RECORD MAINTENANCE
        STATISTICS)

          DCL 1 RECMS_RQ                BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                     BIT(24), /*  0-2   */
                2 TARGET_ADDRESS                BIT(16), /*  3-4   */
                2 MAINTENANCE_STATISTICS        CHAR(*); /*  5-n   */

          0-2       X'010381' NS header
          3-4       Network address of resource
          5-n       Maintenance statistics

RECSTOR; PU_T4|5-->SSCP, Norm; FMD NS(ma) (RECORD STORAGE)

          DCL 1 RECSTOR_RQ              BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                     BIT(24), /*  0-2   */
                2 TARGET_ADDRESS                BIT(16), /*  3-4   */
                2 DISPLAY_TYPE                   BIT(8), /*   5    */
                2 RESERVED                       BIT(8), /*   6    */
                2 DISPLAY_LENGTH                BIT(16), /*  7-8   */
                2 DISPLAY_LOCATION              BIT(32), /*  9-12  */
                2 DISPLAY_DATA                  CHAR(*); /*  13-n  */

          0-2       X'010334' NS header
          3-4       Network address of resource to be displayed
          5         Display source and type:
                    bits 0-3, source (address space) of storage
                          display
                          Note: Refer to implementation
                          documentation for description of these
                          values.
                    bits 4-7, display type:
```

```
                              0001  nonstatic storage display
                              0010  static snapshot display
          6          Reserved
          7-8        Number of  bytes of  program storage  following in
                     this record
          9-12       Beginning location
          13-n       Storage display

RECTD; PU_T4|5-->SSCP, Norm; FMD NS(ma) (RECORD TEST DATA)

          DCL 1 RECTD_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                       BIT(24), /*  0-2   */
                2 TARGET_ADDRESS                  BIT(16), /*  3-4   */
                2 TEST_SELECTION                  BIT(32), /*  5-8   */
                2 TEST_STATUS                     CHAR(*); /*  9-n   */

          0-2        X'010382' NS header
          3-4        Network address of resource under test
          5-8        Binary code selecting the test
          9-n        Test status and results

RECTR; PU_T4|5-->SSCP, Norm; FMD NS(ma) (RECORD TEST RESULTS)

          DCL 1 RECTR_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                       BIT(24), /*  0-2   */
                2 CNM_HEADER,                              /*  3-7   */
                  3 TARGET_ID                     BIT(16), /*  3-4   */
                  3 TARGET_ID_DESC                BIT(16), /*  5-6   */
                  3 REQUEST_SPECIFIC_INFO          BIT(8), /*   7    */
                2 REQUEST_SPECIFIC_DATA           CHAR(*); /*  8-n   */

          0-2        X'410385' NS header

          3-7        CNM Header
          3-4        CNM target ID, as specified in bytes 5-6, bits 2-3
          5-6        bits 0-1, reserved
                     bits 2-3, CNM target ID descriptor:
                               00  byte 4 contains a local address for
                                   a PU  or LU in  a PU_T2 node  or an
                                   LSID  for a  PU or  LU  in a  PU_T1
                                   node; byte 3 is reserved
                               01  bytes 3-4 contain a network address
                                   identifying a  link, adjacent  link
                                   station, PU,  or LU  in the  origin
                                   subarea
                     bits 4-15, procedure  related  identifier  (PRID)
                               (see Note below)
          7          Request-Specific Information
                     bit 0, solicitation indicator:
                             0  unsolicited request
                             1  reply request
                     bit 1, not last request indicator:
                             0  last  request in  a  series of  related
                                unsolicited  or reply  requests,  e.g.,
```

last reply request in a series corresponding to a single soliciting request
1 not last request
bits 2-7, request-specific type code (see below)

Note: For reply (i.e., solicited) requests, bytes 3-6 and byte 7, bits 2-7, echo the corresponding fields in the CNM header received in the request that solicited the reply request(s).

For unsolicited requests, these fields--the CNM target ID descriptor, the CNM target ID, the PRID, and the request-specific information--are generated by the request sender. For unsolicited requests, the PRID field contains X'000'.

Link Level 2 Test Statistics

7	bit 0, solicitation indicator (see above) bit 1, not last request indicator (see above) bits 2-7, type code: 000001; the CNM target ID specifies an adjacent link station attached to a PU_T4|5 node (Note: When the attached adjacent link station is in a PU_T1|2 node, the PU CNM ID is used as the adjacent link station CNM ID.)
8	Reserved
9-10	Number of DLC link test frames transmitted
11-12	Number of DLC link test frames received with or without link errors
13-14	Number of DLC link test frames received without link errors
15-16	Reason for test termination: X'0000' test completed without error X'0001' test completed with error--see bytes 9-14 X'0002' test ended because of link inoperative condition X'0003' test initialization failure; bytes 9-14 contain zeros

RECTRD; PU_T4|5-->SSCP, Norm; FMD NS(ma) (RECORD TRACE DATA)

```
DCL 1 RECTRD_RQ                BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                      BIT(24), /*  0-2   */
      2 TARGET_ADDRESS                 BIT(16), /*  3-4   */
      2 TRACE_TYPE                      BIT(8), /*   5    */
      2 TRACE_DATA                     CHAR(*); /*  6-n   */
```

0-2	X'010383' NS header
3-4	Network address of resource under trace
5	Trace data type bit 0, transmission group trace bits 1-4, reserved

```
                bits 5-6, trace data format
                          10  fixed-length data segments
                          11  variable-length data segments
                bit 7, link trace
          6-n   Trace data
```

RELQ; LU-->LU, Exp; DFC (RELEASE QUIESCE)

```
          DCL 1 RELQ_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
                2 RQ_CODE                          BIT(8); /*   0    */

          0     X'82' request code
```

REQACTLU; PU_T4|5-->SSCP, Norm; FMD NS(c) (REQUEST ACTIVATE LOGICAL UNIT)

```
          DCL 1 REQACTLU_RQ              BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                       BIT(24), /*  0-2   */
                2 LU_ADDRESS                      BIT(16), /*  3-4   */
                2 LU_NTWK_NAME_TYPE                BIT(8), /*   5    */
                2 LU_NTWK_NAME_LENGTH              BIT(8), /*   6    */
                2 LU_NTWK_NAME
                        CHAR(REFER(LU_NTWK_NAME_LENGTH)); /*  7-m   */

          0-2   X'410240' NS header
          3-4   Network address of LU to be sent ACTLU
          5-m   Network Name of LU
          5     Type: X'F3' logical unit
          6     Length, in binary, of network name
          7-m   Symbolic name in EBCDIC characters
```

REQCONT; PU_T4|5-->SSCP, PU-->PUCP, Norm; FMD NS(c) (REQUEST CONTACT)

```
          DCL 1 REQCONT_RQ               BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                       BIT(24), /*  0-2   */
                2 LINK_ADDRESS                    BIT(16), /*  3-4   */
                2 XID_IMAGE,
                  3 FORMAT                         BIT(4), /*   5    */
                  3 PU_TYPE                        BIT(4),
                  3 NODE_ID                       BIT(48), /*  6-11  */
                  3 FORMAT_SPECIFIC_DATA          CHAR(*); /*  12-n  */

          0-2   X'010284' NS header
          3-4   Network address of link
          5-n   XID I-field image: the bytes received in the
                information field of the SDLC XID response; see
                the later section, "DLC XID Information-Field
                Formats," for format details
```

```
REQDISCONT; PU_T1|2-->SSCP, Norm; FMD NS(c) (REQUEST DISCONTACT)

          DCL 1 REQDISCONT_RQ              BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                        BIT(24), /*  0-2   */
                2 DISCONTACT_TYPE                   BIT(4), /*   3    */
                2 SEND_CONTACT_IMMEDIATELY          BIT(4);

          0-2       X'01021B' NS header
          3         bits 0-3, type:
                              X'0'  normal
                              X'8'  immediate
                    bits 4-7, CONTACT information:
                              X'0'  do not send CONTACT immediately
                              X'1'  send CONTACT immediately

REQECHO; LU-->SSCP, Norm; FMD NS(ma) (REQUEST ECHO TEST)

          DCL 1 REQECHO_RQ                 BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                        BIT(24), /*  0-2   */
                2 REPETITION_FACTOR                 BIT(8), /*   3    */
                2 ECHO_DATA_LENGTH                  BIT(8), /*   4    */
                2 ECHO_DATA
                          CHAR(REFER(ECHO_DATA_LENGTH)); /*  5-m   */

          0-2       X'810387' NS header
          3         Repetition factor:  number of times the  test data
                    is to be echoed to the target LU
                    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

REQFNA; PU_T4|5-->SSCP, Norm; FMD NS(c) (REQUEST FREE NETWORK ADDRESS)

          DCL 1 REQFNA_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                        BIT(24), /*  0-2   */
                2 LU_ADDRESS                       BIT(16), /*  3-4   */
                2 RESERVED                          BIT(8), /*   5    */
                2 REQUEST_TYPE                      BIT(8); /*   6    */

          0-2       X'410286' NS header
          3-4       Network address of LU to be deleted
          5         Reserved
          6         Type of request:
                     X'01'  request
                     X'02'  normal
                     X'03'  forced
                     X'04'  cleanup
```

REQMS; SSCP|PUCP-->PU, Norm; FMD NS(ma) (REQUEST MAINTENANCE STATISTICS)

```
DCL 1 REQMS_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 CNM_HEADER,                              /*  3-7   */
        3 TARGET_ID                     BIT(16), /*  3-4   */
        3 TARGET_ID_DESC                BIT(16), /*  5-6   */
        3 REQUEST_SPECIFIC_INFO          BIT(8), /*   7    */
      2 REQUEST_SPECIFIC_DATA           CHAR(*); /*  8-n   */
```

0-2 X'410304' NS header

3-7 CNM Header
3-4 CNM target ID, as specified in bytes 5-6, bits 2-3
5-6 bits 0-1, reserved
bits 2-3, CNM target ID descriptor:
- 00 byte 4 contains a local address for a PU or LU in a PU_T2 node or an LSID for a PU or LU in a PU_T1 node; byte 3 is reserved
- 01 bytes 3-4 contain a network address identifying a link, adjacent link station, PU, or LU in the destination subarea

bits 4-15, procedure related identifier (PRID): a CNM application program generated value for CNM application program correlation, or an SSCP generated value for SSCP routing

7 Request-Specific Information
bit 0, reset indicator (or reserved, as shown below for each Type code):
- 0 do not reset data when RECFMS is sent in reply
- 1 reset data when RECFMS is sent in reply

bit 1, reserved
bits 2-7, request-specific type code (see below)

Note: For reply (i.e., solicited) requests, bytes 3-6 and byte 7, bits 2-7, echo the corresponding fields in the CNM header received in the request that solicited the reply request(s).

7 SDLC Test Command/Response Statistics
bit 0, reset indicator
bit 1, reserved
bits 2-7, type code: 000001; the CNM target ID identifies a PU_T1|2

7 Summary Error Data
bit 0, reset indicator
bit 1, reserved
bits 2-7, type code: 000010; the CNM target ID identifies a PU

```
          7          Communication Adapter Data
                     bits 0-1, reserved
                     bits 2-7, type code: 000011; the CNM target ID
                               identifies a PU_T1|2
          7-n        PU- or LU-Dependent Data
          7          bit 0, reset indicator
                     bit 1, reserved
                     bits 2-7, type code: 000100; the CNM target ID
                               identifies a PU|LU
          8-n        PU- or LU-dependent request parameters:
                     implementation dependent information (See CNM
                     application product specifications for details.)
          7          Engineering Change Levels
                     bits 0-1, reserved
                     bits 2-7, type code: 000101; the CNM target ID
                               identifies a PU
          7-8        Link Connection Subsystem Data
          7          bit 0, reset indicator
                     bit 1, reserved
                     bits 2-7, type code: 000110; the CNM target ID
                               identifies an adjacent link station in
                               the destination subarea
          8          Data selection requested:
                      X'01'  available data (only value defined)

REQTEST;  LU-->SSCP, PU_T4|5-->SSCP, Norm; FMD NS(ma) (REQUEST TEST
          PROCEDURE)

          DCL 1 REQTEST_RQ                BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                        BIT(24), /*  0-2   */
                2 LU1_NAME_TYPE                     BIT(8), /*   3    */
                2 LU1_NAME_LENGTH                   BIT(8), /*   4    */
                2 LU1_NAME  CHAR(REFER(LU1_NAME_LENGTH)), /*  5-m   */
                2 LU2_NAME_TYPE                     BIT(8), /*  m+1   */
                2 LU2_NAME_LENGTH                   BIT(8), /*  m+2   */
                2 LU2_NAME  CHAR(REFER(LU2_NAME_LENGTH)), /* m+3-n  */
                2 PROC_NAME_TYPE                    BIT(8), /*  n+1   */
                2 PROC_NAME_LENGTH                  BIT(8), /*  n+2   */
                2 PROC_NAME
                          CHAR(REFER(PROC_NAME_LENGTH)), /* n+3-p  */
                2 REQUESTER_ID_LENGTH               BIT(8), /*  p+1   */
                2 REQUESTER_ID
                        CHAR(REFER(REQUESTER_ID_LENGTH)), /* p+2-q  */
                2 PASSWORD_LENGTH                   BIT(8), /*  q+1   */
                2 PASSWORD  CHAR(REFER(PASSWORD_LENGTH)), /* q+2-r  */
                2 USER_DATA_LENGTH                  BIT(8), /*  r+1   */
                2 USER_DATA
                          CHAR(REFER(USER_DATA_LENGTH)); /* r+2-s  */

          0-2        X'010380' NS header
                     Network Name 1
          3          Type: X'F3' logical unit
          4          Length: binary number of bytes in symbolic name
                     (X'00' = no symbolic name present)
```

5-m	Symbolic name, in EBCDIC characters, of LU controlling the test
	Network Name 2
m+1	Type: X'F1' physical unit X'F3' logical unit X'F9' link
m+2	Length: binary number of bytes in symbolic name (X'00' = no symbolic name present)
m+3-n	Symbolic name, in EBCDIC characters, of resource to be tested
n+1-p	Procedure Name
n+1	Type: X'F5' test procedure name
n+2	Length: binary number of bytes in symbolic name (X'00' = no symbolic name present)
n+3-p	Symbolic name, in EBCDIC characters, of test procedure to be executed
p+1-q	Requester ID
p+1	Length: binary number of bytes in requester ID (X'00' = no requester ID present)
p+2-q	Requester ID, in EBCDIC characters, of the end user initiating the request (May be used to verify end user's authority to access a particular resource.)
q+1-r	Password
q+1	Length: binary number of bytes in password (X'00' = no password present)
q+2-r	Password, field used to verify the identity of an end user
r+1-s	User Field
r+1	Length: binary number of bytes of user data (X'00' = no user data present)
r+1-s	User data

RNAA; SSCP-->PU_T4|5, Norm; FMD NS(c) (REQUEST NETWORK ADDRESS ASSIGNMENT)

```
DCL 1 RNAA_RQ                      BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                          BIT(24), /*  0-2   */
      2 TARGET_ADDRESS                     BIT(16), /*  3-4   */
      2 ASSIGNMENT_TYPE                     BIT(8), /*   5    */
      2 ENTRY_CNT                           BIT(8), /*   6    */
      2 SUBFIELD(1:REFER(ENTRY_CNT))       BIT(16); /*  7-n   */
```

0-2	X'410210' NS header
3-4	Network address of target link, adjacent link station, or LU
5	Assignment type: X'00' request is for network address assignment of adjacent link station(s) associated with target link X'01' request is for network address assignment of BF.LU(s) associated with the target adjacent link station X'02' request is for an additional network

address assignment for the target LU; bytes 3-4 contain the LU network address used in the SSCP-LU session

6 Number of network addresses to be assigned

7-8 DLC Header Link Station Address, LU Local Address, or LU Network Address Entry

- For Assignment Type 0

7 Reserved

8 DLC header link station address associated with the adjacent link station for which a network address is requested

- For Assignment Type 1

7 Reserved

8 Local address of a BF.LU for which a network address is requested, where the local address has either the one-byte format of FID2 or the six-bit local address format of FID3 (in which case, bits 0-1 of byte 8 are reserved)

- For Assignment Type 2

7-8 Reserved

9-n Any additional two-byte entries in the same format as bytes 7-8 for assignment types 0 and 1 (not present for assignment type 2)

ROUTE_TEST; SSCP-->PU_T4|5, Norm; FMD NS(ma) (ROUTE TEST)

```
DCL 1 ROUTE_TEST_RQ              BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 FORMAT                           BIT(8), /*   3    */
      2 TEST_CODE                        BIT(8), /*   4    */
      2 TEST_TYPE                        BIT(8), /*   5    */
      2 MAX_ER_LENGTH                    BIT(8), /*   6    */
      2 DESTINATION_SA                  BIT(32), /*  7-10  */
      2 ROUTE_MASK                      BIT(16), /* 11-12  */
      2 RQ_CORRELATION                 CHAR(10); /* 13-22  */
```

0-2 X'410306' NS header

3 Format: X'01' (only value defined)

4 Test code:

- X'01' test regardless of the states of ERs
- X'02' test each ER that is not inoperative
- X'03' test each ER that is inoperative
- X'04' do not test the ER; respond with the current ER state (See RSP(ROUTE_TEST))

5 Type of route to be tested:

- X'01' test the ERs corresponding to the ERNs specified in bytes 11-12
- X'02' test the VRs corresponding to the VRNs specified in bytes 11-12; Byte 4 applies to the underlying ERs for the VRs
- X'03' test the ERs corresponding to the defined TG for the ERNs specified in bytes 11-12

6 Maximum expected ER length of any ER being tested

7-10 Subarea address of destination PU for the

NC_ER_TEST request

11-12 A bit is on if the corresponding ERN or VRN (depending on the route type specified in byte 5) is to be tested (Bit 0 corresponds to ERN or VRN 0, bit 1 to ERN or VRN 1, and so forth.)

13-22 Request correlation field: an implementation defined value that is returned in ER_TESTED for correlation of reply to request

RPO; SSCP-->PU_T4|5, Norm; FMD NS(c) (REMOTE POWER OFF)

```
DCL 1 RPO_RQ                    BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 ALS_ADDRESS                     BIT(16); /*  3-4   */
```

0-2 X'010209' NS header

3-4 Network address of adjacent link station associated with the node to be powered off

RQR; SLU-->PLU, SSCP-->SSCP, Exp; SC (REQUEST RECOVERY)

```
DCL 1 RQR_RQ                    BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                         BIT(8); /*   0    */
```

0 X'A3' request code

RSHUTD; SLU-->PLU, Exp; DFC (REQUEST SHUTDOWN)

```
DCL 1 RSHUTD_RQ                 BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                         BIT(8); /*   0    */
```

0 X'C2' request code

RTR; LU-->LU, Norm; DFC (READY TO RECEIVE)

```
DCL 1 RTR_RQ                    BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                         BIT(8); /*   0    */
```

0 X'05' request code

SBI; LU-->LU, Exp; DFC (STOP BRACKET INITIATION)

```
DCL 1 SBI_RQ                    BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                         BIT(8); /*   0    */
```

0 X'71' request code

SDT; PLU-->SLU, SSCP-->PU|SSCP, Exp; SC (START DATA TRAFFIC)

```
DCL 1 SDT_RQ                    BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                         BIT(8); /*   0    */
```

0 X'A0' request code

SESSEND; LU-->SSCP, Norm; FMD NS(s) (SESSION ENDED)

Note: SESSEND is generated by the BF.LU.SVC_MGR on behalf of the SLU in a PU_T1|2 node.

```
DCL 1 SESSEND_RQ                BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 FORMAT                            BIT(4), /*   3    */
      2 RESERVED                          BIT(4),
      2 FORMAT_DATA                     CHAR(*); /*  4-n   */


DCL 1 SESSEND_FMT0_RQ
           BASED(ADDR(SESSEND_RQ.FORMAT_DATA)), /* Byte(s)*/
      2 SESSION_KEY                       BIT(8), /*   4    */
                                          /* See page E-127 */
      2 SESSION_KEY_CONTENT             CHAR(*); /*  5-n   */


DCL 1 SESSEND_FMT2_RQ
           BASED(ADDR(SESSEND_RQ.FORMAT_DATA)), /* Byte(s)*/
      2 CAUSE                             BIT(8), /*   4    */
      2 ACTION                            BIT(8), /*   5    */
      2 SESSION_KEY                       BIT(8), /*   6    */
                                          /* See page E-127 */
      2 SESSION_KEY_CONTENT             CHAR(*); /*  7-n   */
```

0-2 X'810688' NS header

3 bits 0-3, format:
- 0000 format 0
- 0010 format 2

bits 4-7, reserved

Format 0

4 Session key:
- X'06' uninterpreted name pair
- X'07' network address pair

5-n Session Key Content

- For session key X'06': Uninterpreted name pair

5 Type: X'F3' logical unit

6 Length, in binary, of PLU name

7-m EBCDIC character string

m+1 Type: X'F3' logical unit

m+2 Length, in binary, of SLU name

m+3-n EBCDIC character string

- For session key X'07': network address pair

5-6 Network address of PLU

7-8(=n) Network address of SLU

Format 2

4 Cause: indicates the reason for the deactivation of the identified (LU,LU) session (see UNBIND for values)

5 Action: indicates if any resultant action is to be taken and by whom:
- X'01' normal, no resultant automatic action
- X'02' primary half-session will restart

```
                X'03'  secondary half-session will restart
6             Session key:
                X'06'  network name pair
                X'07'  network address pair
7-n           Session Key Content
            • For session key X'06':  network name pair
7             Type: X'F3' logical unit
8             Length, in binary, of symbolic name of PLU
9-m           Symbolic name in EBCDIC characters
m+1           Type: X'F3' logical unit
m+2           Length, in binary, of symbolic name of SLU
m+3-n         Symbolic name in EBCDIC characters
            • For session key X'07': network address pair
7-8           Network address of PLU
9-10(=n)      Network address of SLU
```

SESSST; PLU-->SSCP, Norm; FMD NS(s) (SESSION STARTED)

```
DCL 1 SESSST_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 RESERVED                          BIT(8), /*   3    */
      2 SESSION_KEY                       BIT(8), /*   4    */
                                          /* See page E-127 */
      2 SESSION_KEY_CONTENT             CHAR(*); /*  5-n   */

0-2           X'810686' NS header
3             Reserved
4             Session key:
                X'06'  uninterpreted name pair
                X'07'  network address pair
5-n           Session Key Content
            • For session key X'06': Uninterpreted name pair
5             Type: X'F3' logical unit
6             Length, in binary, of PLU name
7-m           EBCDIC character string
m+1           Type: X'F3' logical unit
m+2           Length, in binary, of SLU name
m+3-n         EBCDIC character string
            • For session key X'07': network address pair
5-6           Network address of PLU
7-8(=n)       Network address of SLU
```

SETCV; SSCP-->PU_T4|5, Norm; FMD NS(c) (SET CONTROL VECTOR)

```
DCL 1 SETCV_RQ                   BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 TARGET_ADDRESS                   BIT(16), /*  3-4   */
      2 CONTROL_VECTOR                  CHAR(*); /*  5-n   */

0-2           X'010211' NS header
3-4           Network address of resource to which control
              vector applies, as described in the Note below
5-n           Control vector, as described in the section
              "Control Vectors and Control Lists," later in this
```

appendix
Note: The following combinations are used in SETCV (configuration services):

Vector Key (Byte 5)	Resource (Bytes 3-4)
X'01'	PU
X'02'	Link to be used for routing to the subarea specified in byte 6
X'03'	SPU
X'04'	LU
X'05'	Link (S/370 channel)

SETCV; SSCP-->PU_T4|5, Norm; FMD NS(ma) (SET CONTROL VECTOR)

```
/*       See the DCL for the NS(c) version of SETCV.      */
```

0-2 X'010311' NS header
3-4 Network address of resource to which control vector applies, as described in the Note below
5-n Control vector, as described in the section "Control Vectors and Control Lists," later in this appendix

Note: The following combination is used in SETCV (maintenance services):

Vector Key (Byte 5)	Resource (Bytes 3-4)
X'08'	Adjacent link station

SHUTC; SLU-->PLU, Exp; DFC (SHUTDOWN COMPLETE)

```
DCL 1 SHUTC_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                           BIT(8); /*   0    */
```

0 X'C1' request code

SHUTD; PLU-->SLU, Exp; DFC (SHUTDOWN)

```
DCL 1 SHUTD_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                           BIT(8); /*   0    */
```

0 X'C0' request code

SIG; LU-->LU, Exp; DFC (SIGNAL)

```
DCL 1 SIG_RQ                    BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                           BIT(8), /*   0    */
      2 SIGNAL_DATA                      BIT(32); /*  1-4   */
```

0 X'C9' request code
1-4 Signal code + signal extension field (2 bytes each), set by the sending end user or NAU services

manager; has meaning only to the NAU services level or above:

X'0000'+'uuuu'	no-op (no system-defined code) + user-defined field
X'0001'+'uuuu'	request to send + user-defined field
X'0002'+'uuuu'	assistance requested + user defined field
X'0003'+'uuuu'	intervention required (no data loss) + user-defined field

STSN; PLU-->SLU, Exp; SC (SET AND TEST SEQUENCE NUMBERS)

```
DCL 1 STSN_RQ                    BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                          BIT(8), /*   0    */
      2 ACTION_CODE_SEC_TO_PRI           BIT(2), /*   1    */
      2 ACTION_CODE_PRI_TO_SEC           BIT(2),
      2 RESERVED                         BIT(4),
      2 SEC_TO_PRI_SQN                  BIT(16), /*  2-3   */
      2 PRI_TO_SEC_SQN                  BIT(16); /*  4-5   */
```

0 X'A2' request code

1 bits 0-1, action code for S-->P flow (related data in bytes 2-3)
bits 2-3, action code for P-->S flow (related data in bytes 4-5)
Note: Each action code is set and processed independently. Values for either action code are:

- 00 ignore; this flow not affected by this STSN
- 01 set; the half-session value is set to the value in bytes 2-3 or 4-5, as appropriate
- 10 sense; secondary half-session's sync point manager returns the transaction processing program's sequence number for this flow in the response RU
- 11 set and test; the half-session value is set to the value in appropriate bytes 2-3 or 4-5, and the secondary half-session's sync point manager compares that value against the transaction processing program's number and responds accordingly

bits 4-7, reserved

2-3 Secondary-to-primary sequence number data to support S-->P action code

4-5 Primary-to-secondary sequence number data to support P-->S action code
Note: For action codes 01 and 11, the appropriate bytes 2-3 or 4-5 contain the value to which the half-session value is set and against which the

secondary half-session's sync point manager tests the transaction processing program's value for the respective flow. For action codes 00 and 10, the appropriate bytes 2-3 or 4-5 are reserved.

TERM-OTHER; TLU-->SSCP, Norm; FMD NS(s)(TERMINATE-OTHER)

```
DCL 1 TERM_OTHER_RQ              BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 FORMAT                            BIT(8), /*   3    */
      2 TYPE                              BIT(8), /*   4    */
      2 REASON                            BIT(8), /*   5    */
      2 NOTIFY_SPECIFICATIONS             BIT(8), /*   6    */
      2 SESSION_KEY                       BIT(8), /*   7    */
                                          /* See page E-127 */
      2 SESSION_KEY_CONTENT
              CHAR(REFER(SESSION_KEY_LENGTH)), /*  8-n   */
      2 REQUESTER_ID_LENGTH               BIT(8), /*  n+1   */
      2 REQUESTER_ID
            CHAR(REFER(REQUESTER_ID_LENGTH)), /* n+2-P  */
      2 PASSWORD_LENGTH                   BIT(8), /*  p+1   */
      2 PASSWORD  CHAR(REFER(PASSWORD_LENGTH)), /* p+2-q  */
      2 URC_LENGTH                        BIT(8), /*  q+1   */
      2 URC            CHAR(REFER(URC_LENGTH)); /* q+2-r  */
```

0-2 X'810682' NS header

3 bits 0-3, Format:
0001 Format 1 (Only value defined)
bits 4-7, reserved

4 Type
bits 0-1,
00 the request applies to active and pending-active sessions
01 the request applies to active, pending-active, and queued sessions
10 the request applies to queued sessions only
11 available only for implementation use

bit 2, reserved if byte 4, bit 7 = 1; otherwise:
0 forced termination--session to be deactivated immediately and unconditionally
1 orderly termination--permitting an end-of-session procedure to be executed at the PLU before the session is deactivated

bit 3,
0 do not send DACTLU to LU1; another session initiation request will be sent for LU1
1 send DACTLU to LU1 when appropriate; no further session initiation request will be sent (from this sender) for LU1

bit 4,
0 do not send DACTLU to LU2; another session initiation request will be sent

		for LU2
		1 send DACTLU to LU2 when appropriate; no further session initiation request will be sent (from this sender) for LU2
	bits 5-6,	00 select session(s) for which LU1 is PLU
		01 select session(s) for which LU2 is PLU
		10 select session(s) regardless of whether LU is PLU or SLU
		11 reserved
	bit 7,	0 orderly or forced (see byte 4, bit 2)
		1 cleanup
5	Reason	
	bits 0-2, reserved	
	bit 3,	0 network user requested the termination
		1 network manager requested the termination
	bit 4, reserved	
	bit 5,	0 normal termination
		1 abnormal termination
	bits 6-7, reserved	
6	NOTIFY specifications:	
	bits 0-5, reserved	
	bit 6,	0 do not notify TLU when the session takedown procedure is complete
		1 notify the TLU when the session takedown procedure is complete.
	bit 7, reserved	
7	Reserved	
8	Session key:	
	X'06' uninterpreted name pair	
	X'07' network address pair	
	X'0A' URC	
9-n	Session Key Content	
	• For session key X'06': uninterpreted name pair	
9	Type: X'F3' logical unit	
10	Length, in binary, of LU1 name	
11-m	EBCDIC character string	
m+1	Type: X'F3' logical unit	
m+2	Length, in binary, of LU2 name	
m+3-n	EBCDIC character string	
	Note: If the length of one of the uninterpreted names (LU1 or LU2, but not both) is zero then all sessions for the named LU, as specified by the Type byte, are terminated as a result of this TERM-OTHER request.	
	• For session key X'07': network address pair	
9-10	Network address of PLU	
11-12(=n)	Network address of SLU	
	• For session key X'0A': URC	
9	Length, in binary, of the URC	
10-n	URC: end user defined identifier	
	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 q+1 through r.

n+1-p	Requester ID
n+1	Length, in binary, of requester ID Note: X'00' = no requester ID
n+2-p	Requester ID: the ID, in EBCDIC characters, of the end user initiating the request
p+1-q	Password
p+1	Length, in binary, of password Note: X'00' = no password is present
p+2-q	Password used to verify the identity of the end user
q+1-r	User Request Correlation (URC) Field
q+1	Length, in binary, of the URC Note: X'00' = no URC
q+2-r	URC: end-user defined identifier; this value can be returned by the SSCP in a subsequent NOTIFY or NSPE to correlate a given session to this terminating request

TERM-OTHER-CD; SSCP(TLU)-->SSCP(OLU), Norm; FMD NS(s) (TERMINATE-OTHER CROSS-DOMAIN)

```
DCL 1 TERM_OTHER_CD_RQ          BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2    */
      2 FORMAT                           BIT(8), /*   3     */
      2 TYPE                             BIT(8), /*   4     */
      2 PCID                            CHAR(8), /*  5-12   */
      2 REASON                           BIT(8), /*  13     */
      2 RESERVED                        BIT(16), /* 14-15   */
      2 SESSION_KEY                      BIT(8), /*  16     */
                                         /* See page E-127 */
      2 SESSION_KEY_CONTENT
              CHAR(REFER(SESSION_KEY_LENGTH)), /*  17-n   */
      2 REQUESTER_ID_LENGTH              BIT(8), /*  n+1    */
      2 REQUESTER_ID
            CHAR(REFER(REQUESTER_ID_LENGTH)), /* n+2-p   */
      2 PASSWORD_LENGTH                  BIT(8), /*  p+1    */
      2 PASSWORD  CHAR(REFER(PASSWORD_LENGTH)); /* p+2-q   */
```

0-2	X'818642' NS header
3	bits 0-3, 0000 Format 0 (only value defined) bits 4-7, reserved
4	Type: bits 0-1, 00 the request applies to active and pending-active sessions 01 the request applies to active, pending-active, and queued sessions 10 the request applies to queued sessions only 11 reserved bit 2, reserved if byte 4, bit 7 = 1; otherwise: 0 forced termination--session to be

deactivated immediately and unconditionally
1 orderly termination--permitting an end-of-session procedure to be executed at the PLU before the session is deactivated

bit 3, 0 do not send DACTLU to LU1; another session initiation request will be sent for LU1
1 send DACTLU to LU1 when appropriate; no further session initiation request will be sent (from this sender) for LU1

bit 4, 0 do not send DACTLU to LU2; another session initiation request will be sent for LU2
1 send DACTLU to LU2 when appropriate; no further session initiation request will be sent (from this sender) for LU2

bits 5-6, 00 select session(s) for which LU1 is PLU
01 select session(s) for which LU2 is PLU
10 select session(s) regardless of whether LU is SLU or PLU
11 reserved

bit 7, 0 orderly or forced (see byte 4, bit 2)
1 cleanup

5-12 PCID

5-6 Network address of the SSCP(TLU)

7-12 A unique 6-byte value, generated by the SSCP(TLU), that is retained and used in all cross-domain requests dealing with the same procedure until it is completed

13 Reason:

bits 0-2, reserved

bit 3, 0 network user requested the termination
1 network manager requested the termination

bit 4, reserved

bit 5, 0 normal termination
1 abnormal termination

bits 6-7, reserved

14-15 Reserved

16 Session key:
X'05' PCID
X'06' network name pair
X'07' network address pair

17-n Session Key Content

- For session key X'05': PCID

17-18 Network address of the SSCP(ILU)

19-24(=n) A unique six-byte value, generated by the SSCP(ILU), that is retained and used in all cross-domain requests dealing with the same procedure until it is completed

Note: This is a PCID generated by the SSCP(ILU), and differs from the one in bytes 5-12.

- For session key X'06': network name pair

17	Type: X'F3' logical unit
18	Length, in binary, of symbolic name of LU1
19-m	Symbolic name in EBCDIC characters
m+1	Type: X'F3' logical unit
m+2	Length, in binary, of symbolic name of LU2
m+3-n	Symbolic name in EBCDIC characters

Note: If the length of one of the network names, but not both, is zero then all sessions specified by the Type byte are terminated as a result of this TERM-OTHER-CD request

- For session key X'07': network address pair

17-18	Network address of PLU
19-20(=n)	Network address of SLU
n+1-p	Requester ID
n+1	Length, in binary, of requester ID

Note: X'00' = no requester ID

n+2-p	Requester ID: the ID, in EBCDIC characters, of the end-user initiating the request
p+1-q	Password
p+1	Length, in binary, of password

Note: X'00' = no password is present

p+2-q	Password used to verify the identity of the end-user

TERM-SELF; TLU-->SSCP, Norm; FMD NS(s) (TERMINATE-SELF)

```
DCL 1 TERM_SELF0_RQ              BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 TYPE                             BIT(8), /*   3    */
      2 DLU_UNINTRP_NAME_TYPE            BIT(8), /*   4    */
      2 DLU_UNINTRP_NAME_LENGTH          BIT(8), /*   5    */
      2 DLU_UNINTRP_NAME
          CHAR(REFER(DLU_UNINTRP_NAME_LENGTH)); /*  6-m   */
```

0-2	X'010683' NS header
3	Type:

bits 0-1,
- 00 the request applies to active and pending-active sessions
- 01 the request applies to active, pending-active, and queued sessions
- 10 the request applies to queued only sessions
- 11 reserved

bit 2, reserved if byte 3, bit 4 = 1; otherwise:
- 0 forced termination--session to be deactivated immediately and unconditionally
- 1 orderly termination--permitting an end-of-session procedure to be executed at the PLU before the session is deactivated

bit 3, 0 do not send DACTLU to OLU; another session initiation request will be sent for OLU
1 send DACTLU to OLU when appropriate; no further session initiation request will be sent (from this sender) for OLU

bit 4, 0 orderly or forced (see byte 3, bit 2)
1 clean up

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 LU is SLU or PLU
11 reserved

bit 7, 0 indicates that the format of the RU is Format 0 and that byte 3 is the Type byte.

4-m Uninterpreted Name of DLU

4 Type: X'F3' logical unit

5 Length, in binary, of DLU name

Note: If the length value of the DLU name is zero, then the TERM-SELF applies to all sessions, as specified in the Type byte, where the TLU is a partner.

6-m EBCDIC character string

Note: The following defaults are supplied by the SSCP receiving a Format 0 TERM-SELF:

- Reason: network user, normal
- Notify: do not notify
- Requester ID, URC, and password are not used in mapping to subsequent requests.

TERM-SELF; TLU-->SSCP, Norm; FMD NS(s) (TERMINATE-SELF)

```
DCL 1 TERM_SELF1_RQ              BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 FORMAT                           BIT(8), /*   3    */
      2 TYPE                             BIT(8), /*   4    */
      2 REASON                           BIT(8), /*   5    */
      2 NOTIFY_SPECIFICATIONS            BIT(8), /*   6    */
      2 RESERVED                         BIT(8), /*   7    */
      2 SESSION_KEY                      BIT(8), /*   8    */
                                         /* See page E-127 */
      2 SESSION_KEY_CONTENT
              CHAR(REFER(SESSION_KEY_LENGTH)), /*  9-n   */
      2 REQUESTER_ID_LENGTH              BIT(8), /*  n+1   */
      2 REQUESTER_ID
            CHAR(REFER(REQUESTER_ID_LENGTH)), /* n+2-p  */
      2 PASSWORD_LENGTH                  BIT(8), /*  p+1   */
      2 PASSWORD  CHAR(REFER(PASSWORD_LENGTH)), /* p+2-q  */
      2 URC_LENGTH                       BIT(8), /*  q+1   */
      2 URC            CHAR(REFER(URC_LENGTH)); /* q+2-r  */
```

```
0-2       X'810683' NS header
3         bits 0-3, format:
                    0001  Format 1 (only value defined)
          bits 4-6, reserved
          bit 7,  1  indicates that byte 3, bits 0-3,
                     contain the format value
4         Type:
          bits 0-1,  00  the request applies to active and
                         pending-active sessions
                     01  the request applies to active,
                         pending-active, and queued sessions
                     10  the request applies to queued
                         sessions only
                     11  available only for implementation
                         use
          bit 2, reserved if byte 4, bit 7 = 1; otherwise:
                  0  forced termination--session to be
                     deactivated immediately and
                     unconditionally
                  1  orderly termination--permitting an
                     end-of-session procedure to be executed
                     at the PLU before the session is
                     deactivated
          bit 3,  0  do not send DACTLU to OLU; another
                     session initiation request will be sent
                     for OLU
                  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 LU is SLU or PLU
                     11  reserved
          bit 7,  0  orderly or forced (see byte 4, bit 2)
                  1  clean up
5         Reason:
          bits 0-2, reserved
          bit 3,  0  network user requested the termination
                  1  network manager requested the
                     termination
          bit 4, reserved
          bit 5,  0  normal termination
                  1  abnormal termination
          bits 6-7, reserved
6         NOTIFY specifications:
          bits 0-5, reserved
          bit 6,  0  do not notify TLU when the session
                     takedown procedure is complete
                  1  notify the TLU when the session
                     takedown procedure is complete
```

```
                bit 7, reserved
          7     Reserved
          8     Session key:
                 X'01'  uninterpreted name
                 X'07'  network address pair
                 X'0A'  URC
          9-n   Session Key Content
                • For session key X'01': uninterpreted name
          9     Type: X'F3' logical unit
          10    Length, in binary, of name
          11-n  EBCDIC character string
                Note: If the length value is zero, then the
                TERM-SELF applies to all sessions specified in the
                Type byte where the TLU is a partner.
                • For session key X'07': network address pair
          9-10  Network address of PLU
          11-12(=n) Network address of SLU
                • For session key X'0A': URC
          9     Length, in binary, of the URC
          10-n  URC: end user defined identifier
                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 q+1
                through r.
          n+1-p Requester ID
          n+1   Length, in binary, of requester ID
                Note: X'00' = no requester ID
          n+2-p Requester ID: the ID, in EBCDIC characters, of the
                end user initiating the request
          p+1-q Password
          p+1   Length, in binary, of password
                Note: X'00' = no password is present
          p+2-q Password used to verify the identity of the end
                user
          q+1-r User Request Correlation (URC) Field
          q+1   Length, in binary, of URC field
                Note: X'00' = no URC
          q+2-r URC: end-user defined identifier; this value can
                be returned by the SSCP in a subsequent NOTIFY to
                correlate a given session to this terminating
                request

TESTMODE; SSCP-->PU_T4|5, Norm; FMD NS(ma) (TEST MODE)

          DCL 1 TESTMODE_RQ              BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                     BIT(24), /*  0-2   */
                2 CNM_HEADER,                            /*  3-7   */
                  3 TARGET_ID                   BIT(16), /*  3-4   */
                  3 TARGET_ID_DESCRIPTOR        BIT(16), /*  5-6   */
                  3 REQUEST_SPECIFIC_INFO        BIT(8), /*   7    */
                2 REQUEST_SPECIFIC_DATA         CHAR(*); /*  8-n   */

          0-2     X'410305' NS header
```

3-7 CNM Header
3-4 CNM target ID, as specified in bytes 5-6, bits 2-3
5-6 bits 0-1, reserved
bits 2-3, CNM target ID descriptor:
00 byte 4 contains a local address for a PU or LU in a PU_T2 node or an LSID for a PU or LU in a PU_T1 node; byte 3 is reserved
01 bytes 3-4 contain a network address identifying a link, adjacent link station, PU, or LU in the destination subarea
bits 4-15, procedure related identifier (PRID): a CNM application program generated value for CNM application program correlation, or an SSCP generated value for SSCP routing
7 Request-Specific Information
bits 0-1, reserved
bits 2-7, request-specific type code (see below)

Note: For reply (i.e., solicited) requests, bytes 3-6 and byte 7, bits 2-7, echo the corresponding fields in the CNM header received in the request that solicited the reply request(s).

7-n Link Level 2 Test Statistics
7 bits 0-1, reserved
bits 2-7, type code: 000001; the CNM target ID specifies an adjacent link station attached to a PU_T4|5 node (Note: When the attached adjacent link station is in a PU_T1|2 node, the PU CNM ID is used as the adjacent link station CNM ID.)
8 Reserved
9-10 Test initiation/termination code:
X'0000'(=n1) terminate an ongoing link test previously initiated
X'FFFF'(=n2) initiate a link test and run it continuously
n=¬(n1|n2) initiate a link test and transmit n test frames
11-12 For point-to-point links this field is reserved; for multipoint links, this field specifies the number of test frame transmissions to be sent each time the secondary link station is serviced, e.g., in SDLC the time interval during which frames are being sent and received from a single secondary link station without another secondary link station on the link being polled or being sent frames
13-n Data to be sent in the data field of the link test frame

UNBIND; LU-->LU, Exp; SC (UNBIND SESSION)

```
DCL 1 UNBIND_RQ                  BASED(ADDR(RU)), /* Byte(s)*/
      2 RQ_CODE                          BIT(8), /*   0    */
      2 SON_CAUSE                        BIT(8), /*   1    */
      2 SENSE_DATA                      BIT(32); /*  2-5   */
```

0 X'32' request code

1 Type UNBIND:

- X'01' normal end of session
- X'02' BIND forthcoming; retain the node resources allocated to this session, if possible
- X'03' talk: the session will be resumed by the sender of UNBIND after alternate use of the physical connection
- X'04' restart mismatch: synch point records do not match; operator intervention is needed before the session can be established
- X'05' LU not authorized: the secondary half-session has failed to supply an acceptable password or other authorization information in the User Data field
- 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 identifed (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'FE' invalid session protocol: the session has failed because a protocol violation has been detected

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 orginally detected (e.g., for a negative response or EXR)

UNBINDF; PLU-->SSCP, Norm; FMD NS(s) (UNBIND FAILURE)

```
DCL 1 UNBINDF_RQ                BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 SENSE_DATA                      BIT(32), /*  3-6   */
      2 REASON                           BIT(8), /*   7    */
      2 SESSION_KEY                      BIT(8), /*   8    */
                                        /* See page E-127 */
      2 SESSION_KEY_CONTENT             CHAR(*); /*  9-n   */
```

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 Session key:
X'06' uninterpreted name pair
X'07' network address pair

9-n Session Key Content

- For session key X'06': uninterpreted name pair

9 Type: X'F3' logical unit
10 Length, in binary, of PLU name
11-m EBCDIC character string
m+1 Type: X'F3' logical unit
m+2 Length, in binary, of SLU name
m+3-n EBCDIC character string

- For session key X'07': network address pair

9-10 Network address of PLU
11-12(=n) Network address of SLU

VR_INOP; PU_T4|5-->SSCP, PU-->PUCP, Norm; FMD NS(c) (VIRTUAL ROUTE INOPERATIVE)

```
DCL 1 VR_INOP_RQ                BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2    */
      2 FORMAT                            BIT(8), /*   3     */
      2 REASON_CODE                       BIT(8), /*   4     */
      2 ORIGINATING_SA                   BIT(32), /*  5-8    */
      2 TG_ADJ_SA                        BIT(32), /*  9-12   */
      2 TG_NUM                            BIT(8), /*   13    */
      2 CNT_VR_FIELD                      BIT(8), /*   14    */
      2 VR_FIELD(1:REFER(CNT_VR_FIELD)),
        3 SA                             BIT(32), /*15-18+8n*/
        3 RESERVED                        BIT(8), /* 19+8n  */
        3 VR_ID                           BIT(8), /* 20+8n  */
        3 MASK                           BIT(16); /*21-22+8n*/
```

0-2	X'410223' NS header
3	Format: X'01' (only value defined)
4	Reason code: X'01' unexpected routing interruption over a transmission group, e.g., the last active link in a TG has failed X'02' controlled routing interruption such as the result of DISCONTACT
5-8	Subarea address of the PU that originated the NC_ER_INOP
9-12	Subarea address on other end of the transmission group that had the routing interruption
13	TGN of the transmission group that had the routing interruption
14	Number of VRs that map to an ER using the above TG
15-22	VR Field
15-18	Subarea address of a destination that is routed to over the VR that uses the failed TG
19	Reserved
20	Virtual route identifier: bits 0-3, VRN bits 4-5, reserved bits 6-7, transmission priority field
21-22	ER INOP mask: a bit is on for the ER used by the VRID (Bit 0 corresponds to ERN 0, bit 1 to ERN 1, and so forth.)
23-n	Any additional eight-byte entries in the same format as bytes 15-22

Session Key DCLs

Session Key DECLAREs

```
DCL SESSION_KEY_LENGTH FIXED BIN(16);


DCL 1 SESSION_KEY_FORMAT_01                    BASED, /* Byte(s)*/
      2 UNINTRP_NAME_TYPE                    BIT(8), /*   0     */
      2 UNINTRP_NAME_LENGTH                  BIT(8), /*   1     */
      2 UNINTRP_NAME
              CHAR(REFER(UNINTRP_NAME_LENGTH)); /*  2-n    */


DCL 1 SESSION_KEY_FORMAT_05                    BASED, /* Byte(s)*/
      2 ILU_SSCP_ADDRESS                   BIT(16), /*  0-1    */
      2 PCID                               BIT(48); /*  2-7    */


DCL 1 SESSION_KEY_FORMAT_06                    BASED, /* Byte(s)*/
      2 PLU_NAME_TYPE                        BIT(8), /*   0     */
      2 PLU_NAME_LENGTH                      BIT(8), /*   1     */
      2 PLU_NAME  CHAR(REFER(PLU_NAME_LENGTH)), /*  2-n    */
      2 SLU_NAME_TYPE                        BIT(8), /*  n+1    */
      2 SLU_NAME_LENGTH                      BIT(8), /*  n+2    */
      2 SLU_NAME  CHAR(REFER(SLU_NAME_LENGTH)); /* n+3-p   */


DCL 1 SESSION_KEY_FORMAT_07                    BASED, /* Byte(s)*/
      2 PLU_NETWORK_ADDRESS                BIT(16), /*  0-1    */
      2 SLU_NETWORK_ADDRESS                BIT(16); /*  2-3    */


DCL 1 SESSION_KEY_FORMAT_08                    BASED, /* Byte(s)*/
      2 OLU_ADDRESS                        BIT(16), /*  0-1    */
      2 DLU_NAME_TYPE                        BIT(8), /*   2     */
      2 DLU_NAME_LENGTH                      BIT(8), /*   3     */
      2 DLU_NAME  CHAR(REFER(DLU_NAME_LENGTH)); /*  4-n    */


DCL 1 SESSION_KEY_FORMAT_0A                    BASED, /* Byte(s)*/
      2 URC_LENGTH                           BIT(8), /*   0     */
      2 URC            CHAR(REFER(URC_LENGTH)); /*  1-n    */
```

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):

```
DCL 1 UNSTRUCTURED_USER_DATA                BASED, /* Byte(s)*/
      2 LENGTH_USER_DATA                   BIT(8), /*   0    */
      2 USER_DATA_TYPE                     bit(8), /*   1    */
      2 USER_DATA
                CHAR(REFER(LENGTH_USER_DATA)); /*  2-n   */
```

- Structured subfield X'00': unstructured data

0	Length of unstructured data field (if 0, this field may be omitted entirely)
1	X'00'
2-n	Unstructured data

```
DCL 1 STRUCTURED_USER_DATA                  BASED, /* Byte(s)*/
      2 LENGTH_USER_DATA                   BIT(8), /*   0    */
      2 USER_DATA_TYPE                     BIT(8), /*   1    */
      2 LENGTH_PRI_RES_QUAL                BIT(8), /*   2    */
      2 PRI_RES_QUAL
            CHAR(REFER(LENGTH_PRI_RES_QUAL)), /*  3-n   */
      2 LENGTH_SEC_RES_QUAL                BIT(8), /*  n+1   */
      2 SEC_RES_QUAL
            CHAR(REFER(LENGTH_PRI_RES_QUAL)); /* n+2-m  */
```

- Structured subfield X'01': session qualifier

0	Length of session qualifier field (if 0, this field may be omitted entirely)
1	X'01'
2	Length of primary resource qualifier (X'00' means no primary resource qualifier is present: values 0 to 8 are valid)
3-n	Primary resource qualifier
n+1	Length of secondary resource qualifier (X'00' means no secondary resource qualifier is present: values 0 to 8 are valid)
n+2-m	Secondary resource qualifier

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 or Response	Number of Bytes in RU
NC	1
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 RUs with Extended Formats."

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 to PU.SVC_MGR.CSC_MGR and PC (where a sensitivity to SSCP-based sessions versus LU-LU sessions does not necessarily exist) and can be chosen for other layers for implementation simplicity. Refer to Appendix G for sense data values and their corresponding meanings.

POSITIVE RESPONSE RU'S WITH EXTENDED FORMATS

Bytes Description

RSP(ACTCDRM); SSCP-->SSCP, Exp; SC

```
DCL 1 ACTCDRM_RSP                BASED(ADDR(RU)), /* Byte(s)*/
      2 REQUEST_CODE                       BIT(8), /*   0    */
      2 FORMAT                             BIT(4), /*   1    */
      2 TYPE_ACTIVATION                    BIT(4),
      2 FM_PROFILE                         BIT(8), /*   2    */
      2 TS_PROFILE                         BIT(8), /*   3    */
      2 CONTENTS_ID                       CHAR(8), /*  4-11  */
      2 SSCP_ID                           CHAR(6), /* 12-17  */
      2 RESERVED                           BIT(2), /*   18   */
      2 SEC_RCV_PAC_CNT                    BIT(6),
      2 CONTROL_VECTORS                  CHAR(*); /*  19-n  */
```

0	X'14' request code
1	bits 0-3, format: X'0' (only value defined) bits 4-7, type activation performed: X'1' cold X'2' ERP
2	FM profile (see Appendix F)
3	TS profile (see Appendix F)
4-11	Contents ID: eight-character EBCDIC symbolic name that represents implementation and installation dependent information about the SSCP issuing the response to ACTCDRM; eight space (X'40') characters is the value used if no information is to be conveyed (This field could be used to provide a check for a functional and configurational match between the SSCPs.)
12-17	SSCP ID: a six-byte field that includes the ID of the SSCP issuing the ACTCDRM response; the first four bits specify the format for the remaining bits: bits 0-3, 0000 bits 4-7, physical unit type (see Appendix F) of the node containing the SSCP bits 8-47, implementation and installation dependent binary identification
18	TS Usage bits 0-1, reserved bits 2-7, secondary CPMGR receive pacing count (zero means no pacing of requests flowing to the secondary)
19-n	Control vector, as described in the section "Control Vectors and Control lists," later in this appendix Note: The following vector keys may be used in RSP(ACTCDRM): X'06' CDRM control vector

X'09' activation request/response sequence identifier
X'FE' one or more control vector keys not recognized in the corresponding request

RSP(ACTLU); LU-->SSCP, Exp; SC

```
DCL 1 ACTLU_RSP                BASED(ADDR(RU)), /* Byte(s)*/
      2 REQUEST_CODE                    BIT(8), /*   0    */
      2 TYPE_ACTIVATION                 BIT(8), /*   1    */
      2 FM_PROFILE                      BIT(4), /*   2    */
      2 TS_PROFILE                      BIT(4),
      2 CONTROL_VECTORS                CHAR(*); /*  3-n   */
```

0 X'0D' request code
1 Type activation selected:
X'01' cold
X'02' ERP
2 bits 0-3, FM profile: same as the corresponding request
bits 4-7, TS profile: same as the corresponding request
3-7 SSCP-LU session capabilities control vector (See the section, "Control Vectors and Control Lists," later in this appendix, for control vector X'00'.)
8-23 LU-LU session services capabilities control vector (See the section "Control Vectors and Control Lists," later in this appendix, for control vector X'0C'.)
Note: A two-byte response can be sent; it means maximum RU size = 256 bytes, LU-LU session limit = 1, LU can act as a secondary LU, and all other fields in control vectors X'00' and X'0C' are defaulted to 0's. except Mode Table Name in control vector X'0C', which is defaulted to eight space (X'40') characters.

RSP(ACTPU); PU-->SSCP|PUCP, Exp; SC

```
DCL 1 ACTPU_RSP                BASED(ADDR(RU)), /* Byte(s)*/
      2 REQUEST_CODE                    BIT(8), /*   0    */
      2 RESERVED                        BIT(2), /*   1    */
      2 FORMAT                          BIT(2),
      2 TYPE_ACTIVATION                 BIT(4),
      2 CONTENTS_ID                    CHAR(8), /*  2-9   */
      2 FORMAT_DATA                    CHAR(*); /*  10-n  */


DCL 1 ACTPU_FMT1_RSP
            BASED(ADDR(ACTPU_RSP.FORMAT_DATA)), /* Byte(s)*/
      2 RESERVED                       CHAR(2), /* 10-11  */
      2 CONTROL_VECTORS                CHAR(*); /*  12-n  */
```

```
DCL 1 ACTPU_FMT2_RSP
          BASED(ADDR(ACTPU_RSP.FORMAT_DATA)), /* Byte(s)*/
      2 LOAD_MODULE_ID                  CHAR(8), /* 10-17  */
      2 RESERVED                        BIT(16), /* 18-19  */
      2 CONTROL_VECTORS                 CHAR(*); /*  20-n  */


DCL 1 ACTPU_FMT3_RSP
          BASED(ADDR(ACTPU_RSP.FORMAT_DATA)), /* Byte(s)*/
      2 CONTROL_VECTORS                 CHAR(*); /*  10-n  */
```

0 X'11' request code

1 bits 0-1, reserved
bits 2-3, format of response:
- 00 format 0
- 01 format 1 (defined only for PU_T1s and PU_T2s)
- 10 format 2 (this format requires that bits 4-7 be set to X'3')
- 11 format 3 (only for PU_T4|5s)

Note: If format 0 is used on a RSP(ACTPU) from a PU_T1|2, it implies that the PU cannot receive FMD requests from the SSCP; for format 1, a control vector specifies this capability--see the control vector with Key = X'07'. A PU_T4|5 does not use format 1, since it can receive FMD requests.
bits 4-7, type activation selected:
- X'1' cold, IPL not required
- X'2' ERP
- X'3' cold, IPL required

2-9 Contents ID: eight-character EBCDIC symbolic name of the load module currently operating in the node; eight space (X'40') characters is the default value

Note: End of Format 0; Formats 1-3 continue below.

10-n Format 1 Continues

10-11 Reserved

12-n Control vector as described in the section "Control Vectors and Control Lists," later in this appendix
Note: The following control vectors may be used in RSP(ACTPU):
- X'07' PU FMD-RU-Usage
- X'FE' vector key not recognized in the corresponding request

10-n Format 2 Continues

10-17 Load module ID: an eight-character EBCDIC symbolic name of the requested IPL load module:
- X'4040...40' any load module will be accepted
- -X'4040...40' identifies specific load module name

18-19 Reserved

20-n Control vector as described in the section "Control Vectors and Control lists," later in this

appendix
Note: The following control vectors may be used in RSP(ACTPU):
X'07' PU FMD-RU-Usage
X'FE' vector key not recognized in the corresponding request

10-n Format 3 Continues
10-n Control vector as described in the section "Control Vectors and Control Lists," later in this appendix
Note: The following control vectors may be used in RSP(ACTPU):
X'09' activation request/response sequence identifier
X'FE' vector keys not recognized in the corresponding request

RSP(ADDLINK); PU_T4|5-->SSCP, Norm; FMD NS(c)

```
DCL 1 ADDLINK_RSP                BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 LINK_ADDRESS                     BIT(16); /*  3-4   */
```

0-2 X'41021E' NS header
3-4 Link network address

RSP(ADDLINKSTA); PU_T4|5-->SSCP, Norm; FMD NS(C)

```
DCL 1 ADDLINKSTA_RSP             BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                        BIT(24), /*  0-2   */
      2 ALS_ADDRESS                      BIT(16); /*  3-4   */
```

0-2 X'410220' NS header
3-4 Adjacent link station network address

RSP(BIND); SLU-->PLU, Exp; SC

```
DCL 1 BIND_RSP                   BASED(ADDR(RU)), /* Byte(s)*/
      2 REQUEST_CODE                      BIT(8), /*    0    */
      2 FORMAT                            BIT(4), /*    1    */
      2 TYPE                              BIT(4),
      2 FM_PROFILE                        BIT(8), /*    2    */
      2 TS_PROFILE                        BIT(8), /*    3    */
      2 PRI_CHAIN_USE                     BIT(1), /*    4    */
      2 PRI_RQ_MODE                       BIT(1),
      2 PRI_CHAIN_RSP                     BIT(2),
      2 PRI_TWO_PHASE_COMMIT              BIT(1),
      2 RESERVED                          BIT(1),
      2 PRI_COMPRESSION_IND               BIT(1),
      2 PRI_EB_IND                        BIT(1),
      2 SEC_CHAIN_USE                     BIT(1), /*    5    */
      2 SEC_RQ_MODE                       BIT(1),
      2 SEC_CHAIN_RSP                     BIT(2),
      2 SEC_TWO_PHASE_COMMIT              BIT(1),
```

```
2 RESERVED                                BIT(1),
2 SEC_COMPRESSION_IND                     BIT(1),
2 SEC_EB_IND                              BIT(1),
2 RESERVED                                BIT(1), /*    6     */
2 FM_HEADER_USAGE                         BIT(1),
2 BRACKETS_USAGE                          BIT(1),
2 BRACKET_TERM_RULE                       BIT(1),
2 ALTERNATE_CODE                          BIT(1),
2 SQN_AVAILABILITY                        BIT(1),
2 BIS_SENT                                BIT(1),
2 RESERVED                                BIT(1),
2 SEND_RCV_MODE                           BIT(2), /*    7     */
2 RECOVERY_RESPONSIBILITY                 BIT(1),
2 CONT_WINNER_LOSER                       BIT(1),
2 RESERVED                                BIT(3),
2 HDX_FF_RESET_STATE                      BIT(1),
2 SEC_TO_PRI_STAGING_IND                  BIT(1), /*    8     */
2 RESERVED                                BIT(1),
2 SEC_SEND_PACING_CNT                     BIT(6),
2 RESERVED                                BIT(2), /*    9     */
2 SEC_RCV_PACING_CNT                      BIT(6),
2 SEC_SEND_MAX_RU_SIZE                    BIT(8), /*   10     */
2 PRI_SEND_MAX_RU_SIZE                    BIT(8), /*   11     */
2 PRI_TO_SEC_STAGING_IND                  BIT(1), /*   12     */
2 RESERVED                                BIT(1),
2 PRI_SEND_PACING_CNT                     BIT(6),
2 RESERVED                                BIT(2), /*   13     */
2 PRI_RCV_PACING_CNT                      BIT(6),
2 PS_PROFILE,                                     /*   14     */
  3 PS_USAGE_FMT                          BIT(1),
  3 LU_LU_SESSION_TYPE                    BIT(7),
2 PS_USAGE                               CHAR(11), /* 15-25    */
2 CRYPTOGRAPHY_PRIVATE                    BIT(2), /*   26     */
2 CRYPTOGRAPHY_SESSION_LEVEL              BIT(2),
2 CRYPTOGRAPHY_LENGTH                     BIT(4),
2 CRYPTOGRAPHY_KEY_ENCIPH_METHOD          BIT(2), /*   27     */
2 RESERVED                                BIT(3),
2 CRYPTOGRAPHY_CIPHER_METHOD              BIT(3),
2 SESS_CRYPTOGRAPHY_KEY
       CHAR(REFER(CRYPTOGRAPHY_LENGTH)), /*  28-k    */
2 PLU_NTWK_NAME_LENGTH                    BIT(8), /*  k+1     */
2 PLU_NTWK_NAME
      CHAR(REFER(PLU_NTWK_NAME_LENGTH)), /* k+2-m    */
2 USER_DATA_LENGTH                        BIT(8), /*  m+1     */
2 USER_DATA
           CHAR(REFER(USER_DATA_LENGTH)), /* m+2-n    */
2 URC_LENGTH                              BIT(8), /*  n+1     */
2 URC              CHAR(REFER(URC_LENGTH)), /* n+2-p    */
2 SLU_NTWK_NAME_LENGTH                    BIT(8), /*  p+1     */
2 SLU_NTWK_NAME
      CHAR(REFER(SLU_NTWK_NAME_LENGTH)); /* p+2-r    */
```

0 X'31' request code

Note: The following bytes are returned for the extended

nonnegotiable BIND response or for the negotiable BIND response. (The request code alone is sent if a nonnegotiable BIND request specifies no session-level cryptography.)

1 bits 0-3, format: 0000 (only value defined)
bits 4-7, type:
0000 negotiable
0001 nonnegotiable

2-25 Bytes as received on BIND request, for nonnegotiable response; or bytes having the same format, but possibly with values changed from those received on the BIND request, for negotiable response

26-k Cryptography Options

26 bits 0-1, private cryptography options: for nonnegotiable case, same value returned as received in the request, if present--see Note 3
bits 2-3, session-level cryptography options: for nonnegotiable case, same value returned as received in the request, if present--see Note 3
bits 4-7, session-level cryptography options field length: same value returned as received in the request, if present--see Note 3 (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--see Note 3
bits 2-4, reserved
bits 5-7, cryptography cipher method: same value returned as received in the request, if present--see Note 3

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; otherwise, same value as in BIND, if present--see Note 3

k+1-r Bytes as received on BIND request, for nonnegotiable response; or bytes having the same format, but possibly with values changed from those received on the BIND request, for negotiable response

Note 1: The extended format is required for the negotiable BIND response or if session-level cryptography is specified in the BIND request; otherwise, only the short form (request code) is used.

Note 2: On a response, if the last byte of a response is a length field and that field is zero, that byte may be dropped from the response. This applies also to byte 26 (where the count

occupies only bits 4-7) if bits 0-3 are also zero--the entire byte may be dropped if no bytes follow.

Note 3: The Cryptography Options field is returned on the response for a nonnegotiable BIND only when session-level cryptography was specified, or for a negotiable BIND.

RSP(CDINIT); SSCP-->SSCP, Norm; FMD NS(s)

```
DCL 1 CDINIT_RSP                BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 FORMAT                           BIT(8), /*   3    */
      2 PROCEDURE_STATUS                 BIT(8), /*   4    */
      2 DLU_ADDRESS                     BIT(16), /*  5-6   */
      2 LU_STATUS                        BIT(8), /*   7    */
      2 COS_ORIGIN                       BIT(8), /*   8    */
      2 COS_NAME                        CHAR(8), /*  9-16  */
      2 MODE_NAME                       CHAR(8); /* 17-24  */
```

0-2 X'818641' NS header

3 Format: same value as received in corresponding request
bits 4-7, reserved

4 Procedure Status:
bits 0-3, reserved
bits 4-7, Status at SSCP receiving CDINIT:
- 0000 reserved
- 0001 initiate successful--proceed
- 0010 initiate successful--queued
- 0011 dequeued--successful
- 0100 dequeued--unsuccessful

5-6 Network address of DLU for CDINIT; for CDINIT(DQ), it is the network address of the LU associated with the SSCP receiving the CDINIT(DQ) request

7 LU status for LU associated with the SSCP receiving the CDINIT request:
bit 0, reserved
bit 1,
- 0 LU is unavailable
- 1 LU is available

bits 2-3, (reserved if LU is available)
- 00 LU session limit exceeded
- 01 reserved
- 10 LU is not currently able to comply with the PLU/SLU specification
- 11 reserved

bit 4,
- 0 existing SSCP to LU path
- 1 no existing SSCP to LU path

bit 5, (reserved in formats 0 and 1)
- 0 UNBIND and SESSEND cannot be sent by the LU or by its boundary function (if any)
- 1 UNBIND and SESSEND will be sent by the LU or by its boundary function (if any)

bits 6-7, 00 reserved
01 LU is PLU
10 LU is SLU
11 reserved

End of Formats 0 and 1; Format 2 continues below

8 COS origin:
bit 0, 0 no COS name from ILU
1 COS name from ILU
bits 1-2, (reserved if byte 8, bit 0 ¬= 0)
01 SSCP(DLU) chose COS name (DLU is SLU)
10 SSCP(OLU) chose COS name (OLU is SLU)
bits 3-7, reserved

9-16 COS name (if byte 8, bits 1-2 ¬= 01, this field carries unpredictable values and is not used): symbolic name of class of service in EBCDIC characters

17-24 Mode name (if byte 8, bits 1-2 ¬= 01, this field carries unpredictable values and is not used): an eight-byte symbolic name (implementation and installation dependent) that identifies the set of rules and protocols to be used for the session (included here for use in reactivating the (LU,LU) session, if necessary; see CINIT and SESSEND for other details)

RSP(CDSESSEND); SSCP-->SSCP, Norm; FMD NS(s)

```
DCL 1 CDSESSEND_RSP              BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 FORMAT                           BIT(4), /*   3    */
      2 RESERVED                         BIT(4),
      2 CAUSE                            BIT(8), /*   4    */
      2 ACTION                           BIT(8); /*   5    */
```

0-2 X'818648' NS header

3 bits 0-3, format: 0010 Format 2 (only value defined)

Note: The extended form of RSP(CDSESSEND,Format 2) is used only in conjunction CDSESSEND(Format 2). For CDSESSEND(Format 0), RSP(CDSESSEND,Format 0) includes only bytes 0-2.

bits 4-7, reserved

4 Cause: cause of deactivation the (LU,LU) session, as specified in byte 12 of CDSESSEND

5 Action: any reactivation of the (LU,LU) session to be performed by either the PLU or SLU, as specified in SESSEND and CDSESSEND and resolved by the SSCPs

RSP(CDTERM); SSCP(DLU)-->SSCP(OLU), Norm; NS(s)

```
DCL 1 CDTERM_RSP                BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 FORMAT                           BIT(4), /*   3    */
      2 RESERVED                        BIT(12), /*   4    */
      2 DLU_ADDRESS                     BIT(16); /*  5-6   */
```

0-2	X'818643' NS header
3	bits 0-3, 0000 Format 0 (only value defined) bits 4-7, reserved
4	Reserved
5-6	Network address of DLU

RSP(CINIT); PLU-->SSCP, Norm; FMD NS(s)

```
DCL 1 CINIT_RSP                 BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 CONTROL_VECTORS                 CHAR(*); /*  3-n   */
```

0-2	X'810601' NS header
3-n	Control vectors as described in the section "Control Vectors and Control Lists," later in this appendix Note: The following control vector key is used in RSP(CINIT): X'FE' control vector keys not recognized

RSP(DSRLST); SSCP-->SSCP, Norm; NS(s)

```
DCL 1 DSRLST_RSP                BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 CONTROL_LIST_DATA               CHAR(*); /*  3-n   */
```

0-2	X'818627' NS header
3-n	Control list entry data for list type: X'01' (only value defined) See the section "Control Vectors and Control Lists" for the format of the control list.

RSP(DUMPINIT); PU_T4|5-->SSCP, Norm; FMD NS(c)

```
DCL 1 DUMPINIT_RSP              BASED(ADDR(RU)), /* Byte(s)*/
      2 NS_HEADER                       BIT(24), /*  0-2   */
      2 DUMP_TEXT                       CHAR(*); /*  3-n   */
```

0-2	X'010206' NS header
3-n	Dump data

```
RSP(DUMPTEXT)

RSP(DUMPTEXT); PU_T4|5-->SSCP, Norm; FMD NS(c)

          DCL 1 DUMPTEXT_RSP              BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                       BIT(24), /*  0-2   */
                2 DUMP_TEXT                       CHAR(*); /*  3-n   */

          0-2       X'010207' NS header
          3-n       Dump data

RSP(INIT-OTHER-CD); SSCP-->SSCP, Norm: FMD NS(s)

          DCL 1 INIT_OTHER_CD_RSP       BASED(ADDR(RU)), /* Byte(s)*/
                2 NS_HEADER                       BIT(24), /*  0-2   */
                2 FORMAT                           BIT(4), /*   3    */
                2 RESERVED                         BIT(4),
                2 PROCEDURE_STATUS                 BIT(8), /*   4    */
                2 LU1_STATUS                       BIT(8), /*   5    */
                2 LU2_STATUS                       BIT(8); /*   6    */

          0-2       X'818640' NS header
          3         Format
                    bits 0-3,  0000  Format 0 (only value defined)
                    bits 4-7, reserved
          4         Procedure Status:
                    bits 0-3, Status for SSCP(LU1)
                              0000  reserved
                              0001  initiate successful--proceed
                              0010  initiate successful--queued
                              0011  dequeued--successful
                              0100  dequeued--unsuccessful
                    bits 4-7, Status for SSCP(LU2)
                              0000  reserved
                              0001  initiate successful--proceed
                              0010  initiate successful--queued
                              0011  dequeued--successful
                              0100  dequeued--unsuccessful
          5         LU1 Status
                    bit 0, reserved
                    bit 1,  0  LU1 is unavailable
                            1  LU1 is available
                    bits 2-3, (reserved if LU1 is available)
                              00  LU1 session limit exceeded
                              01  reserved
                              10  LU1 is not currently able to comply
                                  with the PLU/SLU specification
                              11  reserved
                    bit 4,  0  existing SSCP to LU path
                            1  no existing SSCP to LU path
                    bit 5, reserved
                    bits 6-7,  00  reserved
                               01  LU1 is PLU
                               10  LU1 is SLU
                               11  reserved
          6         LU2 Status:
```

```
                bit 0, reserved
                bit 1,  0  LU2 is unavailable
                        1  LU2 is available
                bits 2-3, (reserved if LU2 is available)
                          00  LU2 session limit exceeded
                          01  reserved
                          10  LU2 is not currently able to comply
                              with the PLU/SLU specification
                          11  reserved
                bit 4,  0  existing SSCP to LU path
                        1  no existing SSCP to LU path
                bit 5, reserved
                bits 6-7,  00  reserved
                           01  LU2 is PLU
                           10  LU2 is SLU
                           11  reserved
```

RSP(RNAA); PU_T4|5-->SSCP, Norm; FMD NS(c)

```
        DCL 1 RNAA_RSP                  BASED(ADDR(RU)), /* Byte(s)*/
              2 NS_HEADER                         BIT(24), /*  0-2   */
              2 TARGET_ADDRESS                    BIT(16), /*  3-4   */
              2 ASSIGNMENT_TYPE                    BIT(8), /*   5    */
              2 ENTRY_CNT                          BIT(8), /*   6    */
              2 SUBFIELD(1:REFER(ENTRY_CNT))      BIT(16); /*  7-m   */
```

0-2	X'410210' NS header
3-5	Set to same value as bytes 3-5 in RNAA request:
3-4	Network address of target link, adjacent link station, or LU
5	Assignment type
6	Number of network addresses returned
7-8	Network address assigned: adjacent link station address for assignment type 0; BF.LU network address for assignment type 1; LU address for assignment type 2
9-n	Any additional network addresses assigned (two-byte multiples), in the same format as bytes 7-8; the order of the network addresses returned corresponds to the order of the entries (bytes 7-n) in the RNAA request

RSP(ROUTE_TEST); PU_T4|5-->SSCP, Norm; FMD NS(ma)

```
        DCL 1 ROUTE_TEST_RSP            BASED(ADDR(RU)), /* Byte(s)*/
              2 NS_HEADER                         BIT(24), /*  0-2   */
              2 FORMAT                             BIT(8), /*   3    */
              2 CNT_ROUTE_DATA                     BIT(8), /*   4    */
              2 ROUTE_DATA(1:REFER(CNT_ROUTE_DATA)),
                3 VR_ID                            BIT(8), /*   5    */
                3 VR_STATUS                        BIT(8), /*   6    */
                3 RESERVED                         BIT(4), /*   7    */
                3 ER_NUM                           BIT(4),
                3 ER_STATUS                        BIT(8), /*   8    */
```

```
          3 ORIGINATING_ADJ_SA          BIT(32), /*  9-12  */
          3 ORIGINATING_TGN              BIT(8); /*   13   */
```

0-2	X'410306' NS header
3	Format: X'01'
4	Count of the number of Route Data fields
5-13	Route Data: information about the ERs or VRs that were tested.
5	Virtual route identifier: bits 0-3, VRN of the VR tested bits 4-5, reserved bits 6-7, transmission priority field of the VR tested
6	VR status: X'00' VR is not defined X'01' VR is in reset state X'02' activation of the VR is pending notification of the activation of the underlying ER X'03' an NC_ACTVR was sent to activate the VR, but no RSP(NC_ACTVR) has been received X'04' an NC_ACTVR was received to activate the VR, but no RSP(NC_ACTVR) has been sent X'05' an NC_DACTVR(Orderly) has been sent, but no RSP(NC_DACTVR) has been received X'06' an NC_DACTVR(Orderly) was received, but no RSP(NC_DACTVR) has been sent X'07' an NC_DACTVR(Forced) was received, but no RSP(NC_DACTVR) has been sent X'08' an NC_DACTVR(Forced) was sent but no RSP(NC_DACTVR) has been received X'09' VR is active
7	bits 0-3, reserved bits 4-7, ERN of the ER tested
8	ER status: X'00' ER is not defined and not currently operative X'01' ER is defined but not currently operative X'02' ER is defined and operative, but not currently active X'03' an NC_ER_ACT was sent, but no NC_ER_ACT_REPLY has been received X'04' an NC_ER_ACT was received, but no NC_ER_ACT_REPLY has been sent X'05' an NC_ER_ACT was received and an NC_ER_ACT_REPLY was sent; an NC_ER_ACT was sent, but no NC_ER_ACT_REPLY has been received X'06' an NC_ER_ACT was received but no ER is defined; should the ER subsequently become defined, an NC_ER_ACT will be sent X'07' an NC_ER_ACT was received and an NC_ER_ACT_REPLY was sent (no NC_ER_ACT has been sent from this end)

X'08' ER is active and each node on the ER supports ER-VR protocols
X'09' ER is operative but not currently defined
X'0A' ER is active and traverses a node that does not support ER-VR protocols

9-12 Subarea address of the adjacent node through which the ER being tested flows from this node

13 Transmission group number of the TG (to the node identified in bytes 9-12) over which the ER being tested flows from this node

14-n Any additional 9-byte entries in the same format as bytes 5-13

RSP(STSN); SLU-->PLU, Exp; SC

```
DCL 1 STSN_RSP                  BASED(ADDR(RU)), /* Byte(s)*/
      2 REQUEST_CODE                      BIT(8), /*   0    */
      2 RESULT_CODE_SEC_TO_PRI            BIT(2), /*   1    */
      2 RESULT_CODE_PRI_TO_SEC            BIT(2),
      2 RESERVED                          BIT(4),
      2 SEC_TO_PRI_SQN                   BIT(16), /*  2-3   */
      2 PRI_TO_SEC_SQN                   BIT(16); /*  4-5   */
```

0 X'A2' request code

1 bits 0-1, result code for S-->P action code in the request (related data in bytes 2-3)

bits 2-3, result code for P-->S action code in the request (related data in bytes 4-5)

Note 1: Values for either result code are:

- For set or ignore action code:
  - 01 ignore (other values reserved); appropriate bytes 2-3 or 4-5 reserved
- For sense action code:
  - 00 for LU-LU session type 0: user-defined meaning; for all other LU-LU session types: reserved (appropriate bytes 2-3 or 4-5 reserved)
  - 01 reserved
  - 10 secondary half-session's sync point manager does not maintain or cannot return a valid transaction processing program sequence number (appropriate bytes 2-3 or 4-5 reserved)
  - 11 transaction processing program sequence number, as known at the secondary, is returned in bytes 2-3 or 4-5, as appropriate
- For set and test action code:
  - 00 for LU-LU session type 0: user-defined meaning; for all other

LU-LU session types: invalid sequence numbers have been detected by the secondary (appropriate bytes 2-3 or 4-5 return the secondary transaction processing program sequence number)

Note 2: invalid determination results when the sequence number indicated could not have occurred. For example, the mounting of an incorrect sync point log tape by the operator at one of the LUs would cause this condition.

01 value received in STSN request equals the transaction processing program sequence number value as known at the secondary (appropriate bytes 2-3 or 4-5 return the secondary's value for the transaction processing program sequence number)

10 secondary half-session's sync point manager does not maintain or cannot return a valid transaction processing program sequence number (appropriate bytes 2-3 or 4-5 reserved)

11 value received in STSN request does not equal the transaction processing program sequence number value as known at the secondary (appropriate bytes 2-3 or 4-5 return the secondary's value for the transaction processing program sequence number)

bits 4-7, reserved

2-3 Secondary-to-primary normal-flow sequence number data to support S-->P result code, or reserved (see Note 1 above)

4-5 Primary-to-secondary normal-flow sequence number data to support P-->S result code or reserved (see Note 1 above)

Note 2: Where the STSN request specified as action codes two "sets," two "ignores," or a combination of "set" and "ignore," the positive response RU optionally may consist of one byte--X'A2' (the STSN request code)--rather than all six bytes.

CONTROL VECTORS AND CONTROL LISTS

The following table shows, by key value, the requests and responses that carry the specific control vector:

Control Vector Key	Requests or Responses Carrying the Vector
X'00'	RSP(ACTLU)
X'01'	SETCV (NS(c))
X'02'	SETCV (NS(c))
X'03'	SETCV (NS(c))
X'04'	SETCV (NS(c))
X'05'	SETCV (NS(c))
X'06'	ACTCDRM, RSP(ACTCDRM)
X'07'	RSP(ACTPU)
X'08'	SETCV (NS(ma))
X'09'	ACTCDRM, ACTPU, RSP(ACTCDRM|ACTPU)
X'0B'	ACTPU
X'0C'	RSP(ACTLU)
X'0D'	CINIT
X'FE'	RSP(ACTCDRM|ACTPU|ACTLU|CINIT)

The following table shows, by list type, the requests and responses that carry the specific control list:

Control List Type	Requests or Responses Carrying the List
X'01'	+RSP(DSRLST)

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):

SSCP-LU Session Capabilities Control Vector

```
DCL 1 CONTROL_VECTOR_TYPE_00                BASED, /* Byte(s)*/
      2 KEY                                BIT(8), /*    0    */
      2 MAX_RU_SIZE                        BIT(8), /*    1    */
      2 CHAR_CODED_CAPABILITY              BIT(1), /*    2    */
      2 FIELD_FORMAT_CAPABILITY            BIT(1),
      2 RESERVED                          BIT(22); /*  3-4    */
```

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 for all possible values.

2-3 LU Capabilities

2 bit 0, character-coded capability:
- 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
- 1 the SSCP may send unsolicited character-coded requests

bit 1, field-formatted capability:
- 0 the SSCP may not send unsolicited field-formatted requests
- 1 the SSCP may send unsolicited field-formatted requests

2-3 bits 2-15, reserved

4 Reserved

Date-Time Control Vector

```
DCL 1 CONTROL_VECTOR_TYPE_01              BASED, /* Byte(s)*/
      2 KEY                             BIT(8), /*   0    */
      2 DATE                          CHAR(12), /*  1-12  */
      2 TIME                           CHAR(8); /* 13-20  */
```

0 Key: X'01'

1-12 Date, in EBCDIC: MM/DD/YY.ddd (MM = month; DD = day of month; YY = year; ddd = Nth day of year, 1-366)

13-20 Time, in EBCDIC: HH.MM.SS (HH = hours; MM = minutes; SS = seconds)

Subarea Routing Control Vector

```
DCL 1 CONTROL_VECTOR_TYPE_02              BASED, /* Byte(s)*/
      2 KEY                             BIT(8), /*   0    */
      2 SUBAREA_ADDRESS                 BIT(8); /*   1    */
```

0 Key: X'02'

1 Subarea address (left-justified)

SDLC Secondary Station Control Vector

```
DCL 1 CONTROL_VECTOR_TYPE_03                BASED, /* Byte(s)*/
      2 KEY                                 BIT(8), /*   0    */
      2 RESERVED                           BIT(13), /*  1-2   */
      2 PU_TYPE                             BIT(2), /*   2    */
      2 RESERVED                            BIT(1),
      2 TS_PROFILE_2                        BIT(1), /*   3    */
      2 LCP_RESET_OPTION                    BIT(1),
      2 RESERVED                            BIT(6),
      2 BTU_SEND_LIMIT                      BIT(8), /*   4    */
      2 MAX_BTU_PER_ALS                     BIT(8), /*   5    */
      2 ERROR_RETRY_INDICATOR               BIT(8), /*   6    */
      2 LINK_ERROR_RECOVERY_INFO           BIT(16), /*  7-8   */
      2 MAX_BTU_SIZE                       BIT(16); /*  9-10  */
```

0 Key: X'03'

1 Reserved

2 PU type identifier for SPU:
bits 0-4, reserved
bits 5-6, 01 PU_T2
10 PU_T1
bit 7, reserved

3 Type modifier:
bit 0, if byte 2 identifies PU_T1:
0 ¬ TS Profile 2
1 TS Profile 2 if byte 2 identifies ¬PU_T1: reserved
bit 1, 0 discontinue link-level contact with adjacent PU_T1|2 node if the PU_T4 initiates an auto network shutdown procedure for the SSCP controlling that PU_T1|2 node
1 continue link-level contact with adjacent PU_T1|2 node if the PU_T4 initiates an auto network shutdown procedure for the SSCP controlling that PU_T1|2 node
bits 2-7, reserved

4 SDLC BTU send limit

5 Maximum consecutive BTUs sent from the primary station to the specified secondary station without another secondary station on the link being polled or being sent BTUs

6 Error retry indicator

7-8 Link error recovery control information

9-10 Byte count of maximum BTU size permitted to be sent to the adjacent link station represented by the specified SPU

LU Control Vector

```
DCL 1 CONTROL_VECTOR_TYPE_04                BASED, /* Byte(s)*/
      2 KEY                                BIT(8), /*   0    */
      2 LU_LOCAL_ID                        BIT(8), /*   1    */
      2 RESERVED                           BIT(2), /*   2    */
      2 SECONDARY_RCV_PACING_CNT           BIT(6),
      2 RESERVED_ONES                      BIT(8), /*   3    */
      2 SCHEDULING_PRIORITY                BIT(8); /*   4    */
```

```
0         Key: X'04'
1         Local address form of LU network address
2         bits 0-1, reserved
          bits 2-7, secondary CPMGR's receive pacing count
3         Reserved, set to a value of 1
4         Scheduling priority to be used for the BF.TCs
          supporting secondary half-sessions involving the
          specified LU:
           X'01'  low priority (batch)
           X'02'  high priority (interactive)
```

Channel Control Vector

```
DCL 1 CONTROL_VECTOR_TYPE_05                BASED, /* Byte(s)*/
      2 KEY                                BIT(8), /*   0    */
      2 CHANNEL_DELAY                     BIT(16); /*  1-2   */
```

```
0         Key: X'05'
1-2       Channel delay: minimum interval between successive
          inbound transmissions (binary, in tenths of a
          second)
```

CDRM Control Vector (Carries information on the capabilities of the SSCP sending the control vector.)

```
DCL 1 CONTROL_VECTOR_TYPE_06                BASED, /* Byte(s)*/
      2 KEY                                BIT(8), /*   0    */
      2 VECTOR_LENGTH                      BIT(8), /*   1    */
      2 CDRM_PROFILE                       BIT(8), /*   2    */
      2 NAME_PAIR_SESSION_KEY              BIT(1), /*   3    */
      2 ADDRESS_PAIR_SESSION_KEY           BIT(1),
      2 PARALLEL_SESSIONS                  BIT(1),
      2 URC                                BIT(1),
      2 RESERVED                           BIT(1),
      2 PCID_SESSION_KEY                   BIT(1),
      2 FORMAT_2_CDINIT_SUPPORT            BIT(1),
      2 FORMAT_2_CDSESSEND_SUPPORT         BIT(1),
      2 RESERVED                          CHAR(*); /*  4-n   */
```

```
0         Key: X'06'
1         Length, in binary, of Description field (X'00' =
          no Description field present)
2-n       Description Field
2         CDRM profile: X'00' (only value defined)
```

```
3          CDRM usage:
           bit 0,  0  name pair session key (X'06') supported
                   1  name pair session key not supported
           bit 1,  0  address pair session key (X'07') not
                      supported
                   1  address pair session key supported
           bit 2,  0  parallel sessions not supported
                   1  parallel sessions supported
           bit 3,  0  URC not supported by SSCP (and all PLUs
                      within its domain) in cross-domain
                      session initiation
                   1  URC supported by SSCP (and all PLUs
                      within its domain) in cross-domain
                      session initiation
           bit 4, reserved
           bit 5,  0  PCID session key (X'05') not supported
                   1  PCID session key supported
           bit 6,  0  CDSESSEND from SSCP(SLU) and
                      CDINIT(Format 2) not supported;
                      requires NS_LSA to reset session
                      knowledge; therefore, all sessions
                      managed by the SSCP use virtual routes
                      mapping to ER0 from the subarea of the
                      SLU to the subarea of the PLU
                   1  CDSESSEND from SSCP(SLU) and
                      CDINIT(Format 2) supported; NS_LSA is
                      not used to reset session knowledge;
                      therefore, no ER restrictions exist for
                      sessions managed by this SSCP
           bit 7,  0  Format 2 CDSESSEND not supported
                   1  Format 2 CDSESSEND supported
           Note: If the control vector is omitted or the
           length is 0, the corresponding request or response
           implicitly specifies that the name pair session
           key is supported and the others are not.
4-n        Reserved
```

PU FMD-RU-Usage Control Vector

```
DCL 1 CONTROL_VECTOR_TYPE_07              BASED, /* Byte(s)*/
      2 KEY                              BIT(8), /*   0    */
      2 RESERVED                         BIT(6), /*   1    */
      2 ADJ_PU_LOAD_CAPABILITY           BIT(1),
      2 PU_FMD_REQUEST_CAPABILITY        BIT(1),
      2 RESERVED                        CHAR(6); /*  2-7   */
```

```
0          Key:  X'07'
1          bits 0-5, reserved
           bit 6, adjacent PU load capability (initialized to
                  0 by the PU_T2):
                   0  adjacent PU cannot load the PU_T2 node
                   1  adjacent PU can load the PU_T2 node
                      (set by the boundary function in the
                      adjacent subarea node)
```

bit 7, FMD request capability of the node:
0 PU cannot receive FMD requests from the SSCP
1 PU can receive FMD requests from the SSCP

2-7 Reserved

Intensive Mode Control Vector

```
DCL 1 CONTROL_VECTOR_TYPE_08              BASED, /* Byte(s)*/
      2 KEY                              BIT(8), /*   0    */
      2 INTENSIVE_MODE_SET_RESET         BIT(1), /*   1    */
      2 RESERVED                         BIT(7),
      2 MAX_NUMBER_OF_IMRS              BIT(16); /*  2-3   */
```

0 Key X'08'
1 bit 0, 0 reset intensive mode
1 set intensive mode
bits 1-7, reserved
2-3 Maximum number of intensive mode records (IMRs)

Activation Request/Response Sequence Identifier Control Vector

```
DCL 1 CONTROL_VECTOR_TYPE_09              BASED, /* Byte(s)*/
      2 KEY                              BIT(8), /*   0    */
      2 VECTOR_LENGTH                    BIT(8), /*   1    */
      2 ACT_REQ_SEQ_ID                  CHAR(8); /*  2-9   */
```

0 Key: X'09'
1 Length, in binary, of Vector Data field
2-9 Vector Data Field
2-9 Activation request/response sequence identifier: an eight-byte binary value, generated by the sender of ACTCDRM, RSP(ACTCDRM), ACTPU, and echoed in RSP(ACTPU), and used by the receiver to determine whether the current RU supersedes a previously received RU from the same sender (If the current RU has an activation request/response sequence identifier value greater than the corresponding activation request/response sequence identifier value of the earlier ACTPU, ACTCDRM, or RSP(ACTCDRM), the current RU is accepted and processed, while the earlier RU is superseded. The eight-byte field has the following characteristic: If n1 was generated at time t1, and n2 was generated at time t2, and t1 < t2, then n1 < n2.)

SSCP-PU Session Capabilities Control Vector

```
DCL 1 CONTROL_VECTOR_TYPE_0B                BASED, /* Byte(s)*/
      2 KEY                                BIT(8), /*    0     */
      2 VECTOR_LENGTH                      BIT(8), /*    1     */
      2 NS_LSA_REQUIRED                    BIT(1), /*    2     */
      2 ALS_ADDRESS_SUPPORT                BIT(1),
      2 RESERVED                           BIT(6);
```

```
0         Key: X'0B'
1         Length, in binary, of Vector Data field
2         Vector Data Field
2         bit 0,  0  NS_LSA required
                  1  NS_LSA not required
          bit 1,  0  adjacent link  station network  address
                     not supported
                  1  adjacent link  station network  address
                     supported
          bits 2-7, reserved
```

LU-LU Session Services Capabilities Control Vector

```
DCL 1 CONTROL_VECTOR_TYPE_0C                BASED, /* Byte(s)*/
      2 KEY                                BIT(8), /*    0     */
      2 VECTOR_LENGTH                      BIT(8), /*    1     */
      2 PRI_LU_CAPABILITY                  BIT(4), /*    2     */
      2 SEC_LU_CAPABILITY                  BIT(4),
      2 LU_LU_SESSION_LIMIT               BIT(16), /*  3-4     */
      2 LU_LU_SESSION_COUNT               BIT(16), /*  5-6     */
      2 PARALLEL_SESSION_CAPABILITY        BIT(1), /*    7     */
      2 NOTIFY_AT_SESSION_END              BIT(1),
      2 RESERVED                           BIT(6),
      2 MODE_NAME_TABLE                   CHAR(8); /*  8-15    */
```

```
0         Key: X'0C'
1         Length, in binary, of vector data field
2-15      Vector Data Field
2         bits 0-3, primary LU capability:
                    0000  cannot ever act as primary LU
                    0001  cannot currently  act as  primary
                          LU
                    0010  reserved
                    0011  can now act as primary LU
          bits 4-7, secondary LU capability:
                    0000  cannot ever act as secondary LU
                    0001  cannot currently act as secondary
                          LU
                    0010  reserved
                    0011  can now act as secondary LU
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:
- 0 parallel sessions not supported
- 1 parallel sessions supported

bit 1,
- 0 do not send NOTIFY at the completion of (LU,LU) session deactivation
- 1 send NOTIFY at the completion of the (LU,LU) session deactivation

bits 2-7, reserved

8-15 Mode table name: an eight-character symbolic name (implementation and installation dependent) that identifies the mode table that contains the mode name (A value of eight space (X'40') characters means that the mode table name is to be selected by the SSCP.)

Mode/Class-of-Service/Virtual-Route-Identifier-List Control Vector

```
DCL 1 CONTROL_VECTOR_TYPE_0D                BASED, /* Byte(s)*/
      2 KEY                                 BIT(8), /*   0    */
      2 VECTOR_LENGTH                       BIT(8), /*   1    */
      2 MODE_NAME                          CHAR(8), /*  2-9   */
      2 COS_NAME                           CHAR(8), /* 10-17  */
      2 VR_INFO_LENGTH                      BIT(8), /*   18   */
      2 VR_ID_LIST_FORMAT                   BIT(8), /*   19   */
      2 TYPE_OF_VR_REQUIRED                 BIT(8), /*   20   */
      2 NUMBER_OF_VRNS                      BIT(8), /*   21   */
      2 VR_ID_LIST(1:REFER(NUMBER_OF_VRNS))
                                           BIT(16); /*  22-n  */
```

0 Key: X'0D'

1 Length, in binary, of vector data field

2-n Vector Data Field

2-9 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

10-17 COS name: symbolic name of class of service in EBCDIC characters

18-n Virtual Route Information

18 Length (in bytes)--including format, type, number of entries, and entries of Virtual Route Information field

19 Format of virtual route identifier list:
- X'00' format 0 (only value defined)

20 Type of virtual route required:
- X'00' only virtual routes mapping to ER0 from the subarea of the SLU to the subarea of the PLU may be used
- 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

Control Vector Keys Not Recognized Control Vector

```
DCL 1 CONTROL_VECTOR_TYPE_FE                BASED, /* Byte(s)*/
      2 KEY                                BIT(8), /*   0    */
      2 VECTOR_LENGTH                      BIT(8), /*   1    */
      2 NOT_RECOGNIZED_VECTOR
                  CHAR(REFER(VECTOR_LENGTH)); /*  2-n   */
```

0 Key: X'FE'

1 Length, in binary, of vector data field

2-n Vector Data Field

2 Control vector key value not recognized in corresponding request

3-n Any additional unrecognized control vector keys

The control lists are defined, by type, as follows (with zero-origin indexing of the list bytes; see the individual RU description for the actual displacement within the RU):

Type X'01': LU Status Control List Entry

```
DCL 1 CONTROL_LIST_TYPE_01(32)              BASED, /* Byte(s)*/
      2 RESERVED                           BIT(1), /*   0    */
      2 LU_AVAILABILITY                    BIT(1),
      2 LU_SESSION_STATUS                  BIT(2),
      2 SSCP_LU_PATH_EXISTS                BIT(1),
      2 RESERVED                           BIT(3),
      2 LU_IN_PU_TYPE_5                    BIT(1), /*   1    */
      2 RESERVED                           BIT(6),
      2 LU_ACCEPTING_INIT_LOGON            BIT(1),
      2 SESSION_COUNT                     BIT(16); /*  2-3   */
```

0 LU status
bit 0, reserved
bit 1, 0 LU is unavailable
1 LU is available
bits 2-3, (if LU is unavailable)
00 LU session count exceeded
01 LU is being taken down (not accepting new sessions)
10 LU is not currently able to comply with the PLU/SLU specification
11 reserved
bit 4, 0 existing SSCP to LU path
1 no existing SSCP to LU path
bits 5-7, reserved

1 LU information:
bit 0, 0 LU does not reside in a PU_T5 node

	1 LU resides in a PU_T5 node
	bits 1-6, reserved
	bit 7, 0 LU is accepting INITIATEs/logons
	1 LU is temporarily not accepting INITIATEs/logons
2-3	Session count (range: 0-65535)

DLC XID INFORMATION-FIELD FORMATS

This section describes the formats of the information field of the XID command (sent by a primary link station) and response (sent by a secondary link station); XID Formats 0, 1, and 2 apply to SDLC, and Format 2 applies also to the System/370 channel DLC. The response format for Formats 0 and 1 is also carried in the REQCONT request RU, which is sent from the PPU to the SSCP or PUCP. The contents of XID Format 2 sent and received are also included in the CONTACTED RU, which is sent from the PU to the SSCP or PUCP.

```
DCL 1 XID
        BASED(ADDR(RU)) UNALIGNED, /* BYTES */ /* Byte(s)*/
      2 FORMAT                          BIT(4), /*   0    */
      2 PU_TYPE                         BIT(4),
      2 LENGTH                          BIT(8), /*   1    */
      2 NODE_ID,                                /*  2-7   */
        3 BLOCK_NUM                    BIT(12), /*  2-3   */
        3 ID_NUM                       BIT(20), /*  3-5   */
        3 RESERVED                     BIT(16), /*  6-7   */
      2 XID_NUMBERS                    CHAR(1); /*   8    */


DCL 1 XID_1       BASED(ADDR(XID.XID_NUMBERS)), /* Byte(s)*/
      2 RESERVED                        BIT(2), /*   8    */
      2 SENDER_LINK_STATION_ROLE        BIT(1),
      2 RESERVED                        BIT(1),
      2 XMIT_RCV_CAPABILITY             BIT(4),
      2 RESERVED                        BIT(2), /*   9    */
      2 SEGMENT_ASSEM_CAPABILITY        BIT(2),
      2 RESERVED                        BIT(4),
      2 MAX_I_FIELD_LENGTH    FIXED BINARY(15), /* 10-11  */
      2 RESERVED                        BIT(4), /*   12   */
      2 SDLC_CMD_RSP_PROFILE            BIT(4),
      2 RESERVED                        BIT(2), /*   13   */
      2 SIM_RIM_SUPPORT                 BIT(1),
      2 RESERVED                       BIT(22), /* 13-16  */
      2 MAX_NUM_I_FRAMES                BIT(7),
      2 RESERVED                        BIT(8), /*   17   */
      2 SDLC_ADDRESS_LENGTH    FIXED BINARY(8), /*   18   */
      2 SDLC_SECONDARY_STATION
            CHAR(REFER(SDLC_ADDRESS_LENGTH)), /*   19   */
      2 NUMBER_OF_DIAL_DIGITS     FIXED BIN(8), /*   20   */
      2 DIAL_DIGITS
          CHAR(REFER(NUMBER_OF_DIAL_DIGITS)); /*  21-n  */


DCL 1 XID_2       BASED(ADDR(XID.XID_NUMBERS)), /* Byte(s)*/
      2 TG_STATUS                       BIT(1), /*   8    */
      2 MULTI_LINK                      BIT(1),
      2 SEG_ASSEM_CAP                   BIT(2),
      2 RESERVED                        BIT(4),
```

```
        2 FID_0_SUPPORTED                  BIT(1), /*   9     */
        2 FID_1_SUPPORTED                  BIT(1),
        2 RESERVED                         BIT(2),
        2 FID_4_SUPPORTED                  BIT(1),
        2 RESERVED                        BIT(11), /*  9-10   */
        2 MAX_PIU_LENGTH                  BIT(16), /* 11-12   */
        2 TGN                              BIT(8), /*   13    */
        2 SA                              BIT(32), /* 14-17   */
        2 RESERVED                         BIT(1), /*   18    */
        2 ERROR_STATUS                     BIT(4),
        2 RESERVED                         BIT(3),
        2 CONTACT_OR_LOAD_STAT             BIT(8), /*   19    */
        2 IPL_LOAD_MODULE_NAME            CHAR(8), /* 20-27   */
        2 RESERVED                        BIT(16), /* 28-29   */
        2 DLC_TYPE                         BIT(8), /*   30    */
        2 DLC_UNIQUE                      CHAR(1); /*   31    */

DCL 1 XID_2_SDLC
                BASED(ADDR(XID_2.DLC_UNIQUE)), /* Byte(s)*/
        2 RESERVED                         BIT(2), /*   31    */
        2 STA_ROLE_SEC                     BIT(1),
        2 STA_ROLE_PRI                     BIT(1),
        2 RESERVED                         BIT(2),
        2 STA_XMIT_RCV_CAP                 BIT(2),
        2 MAX_RECEIVABLE_I_FIELD          BIT(16), /* 32-33   */
        2 RESERVED                         BIT(4), /*   34    */
        2 CMD_RSP_PROFILE                  BIT(4),
        2 RESERVED                         BIT(2), /*   35    */
        2 SDLC_INIT_MODE,
          3 SDLC_INIT_SEND                 BIT(1),
          3 SDLC_INIT_RCV                  BIT(1),
        2 RESERVED                        BIT(21), /* 35-38   */
        2 MAXIN                            BIT(7), /*   38    */
        2 RESERVED                        BIT(40); /* 39-43   */

DCL 1 XID_2_CHL  BASED(ADDR(XID_2.DLC_UNIQUE)), /* Byte(s)*/
        2 INIT_BUFFS                       BIT(8), /*   31    */
        2 READ_CCWS                       BIT(16), /* 32-33   */
        2 BYTES_PER_READ                  BIT(16), /* 34-35   */
        2 BYTES_OF_PAD                     BIT(8), /*   36    */
        2 STATUS_MODIFIER                  BIT(1), /*   37    */
        2 RESERVED                         BIT(1),
        2 ACTIVE_TG_ACTION                 BIT(1),
        2 RESERVED                         BIT(5),
        2 ATTN_DELAY                      BIT(16), /* 38-39   */
        2 ATTN_TIMEOUT                    BIT(16); /* 40-41   */
```

0 bits 0-3, format of XID I-field:
X'0' fixed format: only bytes 0-5 are included

X'1' variable format (for PU_T1|2 to PU_T4|5 node exchanges): bytes 0-p are included
X'2' variable format (for PU_T4|5 to PU_T4|5 node exchanges): bytes 0-p are included

bits 4-7, type of the XID-sending node:
X'1' PU_T1
X'2' PU_T2
X'3' reserved
X'4' subarea node (PU_T4 or PU_T5)

1 Length, in binary, of variable-format XID I-field; reserved for fixed-format XID I-field

2-5|7 Node Identification

2-5 bits 0-11, Block number: an IBM product specific number; see the individual product specifications for the specific values used

bits 12-31, ID number: a binary value that, together with the block number, identifies a specific station uniquely within a customer network installation; the ID number can be assigned in various ways, depending on the product; see the individual product specifications for details

End of Format 0

6-p Format 1 Continuation

6-7 Reserved

8 Link Station and Connection Protocol Flags

8 bits 0-1, reserved

bit 2, link-station role of XID sender:
0 sender is a secondary link station
1 sender is a primary link station

bit 3, reserved

bits 4-7, link-station transmit-receive capability:
X'0' two-way alternating
X'1' two-way simultaneous

9 Characteristics of the node of the XID sender:

bits 0-1, reserved

bits 2-3, segment assembly capability of the path control element of the node:
00 the Mapping field is ignored and PIUs are forwarded unchanged
01 segments are assembled on a link-station basis
10 segments are assembled on a session basis
11 only whole BIUs are allowed

bits 4-7, reserved

10-11 Maximum I-field length that the XID sender can receive:

bit 0, format flag:

0 bits 1-15 contain the maximum I-field length (only value defined)

bits 1-15, maximum I-field length, in binary

12 bits 0-3, reserved

bits 4-7, SDLC command/response profile:

X'0' SNA link profile (only value defined)

Note: This profile refers to the mandatory command/response support on a SDLC link, as follows:

- For an SDLC link, having a point-to-point or multipoint configuration, the support required is:

Commands	Responses
I-frames	I-frames
RR	RR
RNR	RNR
Test	Test
XID	XID
SNRM	UA
Disconnect	DM
-	RD (Note 1)
-	Frame Reject
Reject (Note 2)	Reject (Note 2)

Note 1: The RD response is sent by the secondary station if and only if the SPU in its node receives a DISCONTACT request from its SSCP or PUCP.

Note 2: Reject is required only if both sender and receiver have two-way simultaneous transmit-receive capability.

- For an SDLC link having a loop configuration, the support required is:

Commands	Responses
I-frames	I-frames
RR	RR
RNR	RNR
Test	Test
XID	XID
SNRM	UA
Disconnect	DM

UP	-
-	Frame Reject
Configure	Configure
-	Beacon
-	RD (Note)

Note: The RD response is sent by the secondary station if and only if the SPU in its node receives a DISCONTACT request from its SSCP or PUCP.

13 bits 0-1, reserved
bit 2, SDLC initialization mode options:
0 SIM and RIM not supported
1 SIM and RIM supported
bits 3-7, reserved

14-15 Reserved

16 bit 0, reserved
bits 1-7, maximum number of I-frames that can be received by the XID sender before an acknowledgment is sent, with an implied modulus for the send and receive sequence counts--less than 8 implies a modulus of 8, 8 or greater implies a modulus of 128

17 Reserved

18-m SDLC Address Assignment Field

18 Length in bytes (or octets) of the SDLC address to be assigned (bytes 19-m)

19-m Secondary station address to be assigned

m+1-p Dial Digits of XID Sender

m+1 Number of dial digits

m+2-p Dial digits: any byte value of the form X'Fn' (0≤n≤F) is valid

• End of Format 1

8-p Format 2 Continuation

8 bit 0, TG status:
0 TG inactive
1 TG active
bit 1, multiple-link TG support:
0 multiple-link TG not supported
1 multiple-link TG supported
bits 2-3, segment assembly capability of the path control element of the node:
00 segments are ignored and passed through
01 segments are assembled on a link station basis
10 segments are assembled on a session basis
11 segments are not allowed
bits 4-7, reserved

9 FID types supported:
bit 0, 0 FID 0 not supported

1 FID 0 supported
bit 1, 0 FID 1 not supported
1 FID 1 supported
Note: Neither bit 0 nor bit 1 is set to 1 when XID Format 2 is exchanged, but can be set by PU.SVC_MGR when the contents of XID Format 2 is carried in the CONTACTED RU.
bits 2-3, reserved
bit 4, 0 FID 4 not supported
1 FID 4 supported
bits 5-7, reserved

10 Reserved

11-12 Length, in binary, of maximum PIU that the XID sender can receive

13 Transmission group number (TGN)

14-17 Subarea address of the XID sender (right-justified with leading 0's)

18 bit 0, reserved
bits 1-4, error status (set in reply to a previously received XID):
X'8' exchanged parameters in the XIDs are not compatible
X'9' incompatible parameters in the XID received for addition of the link station to currently active multiple-link TG (e.g., maximum PIU length)
X'A' TG is not defined (i.e., no routing found)
X'C' multiple-link TG support (byte 8, bit 1) or DLC type (byte 30) specified in the XIDs is incompatible with a link in the associated active TG
bits 5-7, reserved

19 CONTACT or load status of XID sender:
X'00' CONTACT has been received by an XID command sender
X'07' XID response sender is already loaded

20-27 IPL load module name: an 8-character EBCDIC symbolic name of the IPL load module of the XID sender
Note: X'40...40' = no information conveyed

28-29 Reserved

30 DLC type:
X'01' SDLC
X'02' System/370 channel--communication controller is the secondary

31-p DLC-Dependent Parameters

- For SDLC

31 bits 0-1, reserved
bits 2-3, link-station role of XID sender:
bit 2, 0 XID sender cannot be secondary
1 XID sender can be secondary

bit 3, 0 XID sender cannot be primary
1 XID sender can be primary
Note: A combination of 00 in bits 2-3 is reserved.
bits 4-5, reserved
bits 6-7, link station transmit-receive capability:
00 two-way alternating
01 two-way simultaneous

32-33 Maximum I-field length, in binary, that the XID sender can receive

34 bits 0-3, reserved
bits 4-7, SDLC command/response profile:
X'0' SNA link profile (only value defined)
Note: See the Notes described in Format 1, byte 12, for this profile.

35 bits 0-1, reserved
bits 2-3, SDLC initialization mode options:
bit 2, 0 XID sender cannot send SIM nor receive RIM (or RQI)
1 XID sender can send SIM and receive RIM (or RQI)
bit 3, 0 XID sender cannot receive SIM nor send RIM (or RQI)
1 XID sender can receive SIM and send RIM (or RQI)
bits 4-7, reserved

36-37 Reserved

38 bit 0, reserved
bits 1-7, maximum number of I-frames that can be received by the XID sender before an acknowledgment is sent, with an implied modulus for the send and receive sequence counts--less than 8 implies a modulus of 8, 8 or greater implies a modulus of 128

39-43(=p) Reserved

31-p For System/370 Channel DLC

31 Number of initial buffers suggested by the primary link station for the secondary link station to use for data transfer from primary to secondary (primary sets and secondary echoes)
Note: X'00' = no suggestion made. If byte 31 = X'00' in the XID received, secondary uses the value defined by optional implementation and installation specific parameters and sends it to the primary

32-33 Number of Read channel command words that primary issues to secondary in a channel program (primary sets and secondary echoes)
Note: If secondary does not agree with the received value, secondary sends the value defined by implementation- and installation-specific

parameters; byte 18, bit 1, is set to 1.

34-35 Number of data bytes allocated per Read channel command at primary (primary sets and secondary echoes)
Note: If secondary does not agree with the received value, secondary sends the value defined by implementation- and installation-specific parameters; byte 18, bit 1, is set to 1.

36 Number of pad (X'00') characters secondary transmits to primary immediately preceding each PIU to be sent (primary sets and secondary echoes)
Note: If secondary does not agree with the received value, secondary sends the value defined by implementation- and installation-specific parameters; byte 18, bit 1, is set to 1.

37 bit 0, reserved for primary; for secondary:
- 0 secondary does not use the status modifier option for data transfer to primary
- 1 secondary uses the status modifier option for data transfer to primary

bit 1, reserved
bit 2, reserved for secondary; for primary:
- 0 if the TG specified in this XID is active, the secondary is to send an XID response with error status X'C' in byte 18
- 1 if the TG specified in this XID is active and associated with another System/370 channel, INOP is to be sent for the previously activated System/370 channel and the requested System/370 channel is to be activated

bits 3-7, reserved

38-39 Reserved for primary; for secondary: the maximum interval (in tenths of a second) that the secondary delays between the time it has a PIU for the primary and the time it presents an Attention signal to the primary

40-41(=p) Reserved for primary; for secondary: the maximum interval (in tenths of a second) that the secondary awaits a response to an Attention signal that has been sent to the primary before initiating inoperative link processing

FUNCTION MANAGEMENT (FM) PROFILES

This section describes the function management (FM) profiles and their use by the various sessions defined in SNA. Profile numbers not shown are reserved.

Note: If the FM Usage field 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 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 FM headers.
- No brackets.
- No alternate code.
- Normal-flow send/receive mode is HDX-CONT.
- Secondary half-session wins contention.
- Primary half-session is responsible for recovery.

FM PROFILE 2

Profile 2 specifies the following session rules:

- Secondary LU half-session uses delayed request mode.
- Secondary LU half-session uses immediate response mode.
- Only single-RU chains allowed.
- Secondary LU half-session requests indicate no-response.
- No compression.
- No DFC RUs.
- No FM headers.
- Secondary LU half-session is first speaker if brackets are used.
- Bracket termination rule 2 is used if brackets are used.
- Primary LU half-session will send EB.
- Secondary LU half-session will not send EB.
- Normal-flow send/receive mode is FDX.
- Primary LU half-session is responsible for recovery.

The FM Usage fields defining the options for Profile 2 are:

- Primary request control mode selection
- Primary chain response protocol (no-response may not be used)
- Brackets usage and reset state
- Alternate code

FM PROFILE 3

Profile 3 specifies the following session rules:

Primary LU half-session and secondary LU half-session use immediate response mode.
Primary LU half-session and secondary LU half-session support the following DFC functions:

CANCEL
SIGNAL
LUSTAT (allowed secondary-to-primary only)
CHASE
SHUTD
SHUTC
RSHUTD
BID and RTR (allowed only if brackets are used)

The FM usage fields defining the options for Profile 3 are:

Chaining use (primary and secondary)
Request control mode selection (primary and secondary)
Chain response protocol (primary and secondary)
Compression indicator (primary and secondary)
Send EB indicator (primary and secondary)
FM header usage
Brackets usage and reset state
Bracket termination rule
Alternate Code Set Allowed indicator
Normal-flow send/receive mode
Recovery responsibility
Contention winner/loser
Half-duplex flip-flop reset states

FM PROFILE 4

Profile 4 specifies the following session rules:

Primary LU half-session and secondary LU half-session use immediate response mode.

Primary LU half-session and secondary LU half-session support the following DFC functions:

- CANCEL
- SIGNAL
- LUSTAT
- QEC
- QC
- RELQ
- SHUTD
- SHUTC
- RSHUTD
- CHASE
- BID and RTR (allowed only if brackets are used)

The FM Usage fields defining the options for Profile 4 are:

- Chaining use (primary and secondary)
- Request control mode selection (primary and secondary)
- Chain response protocol (primary and secondary)
- Compression indicator (primary and secondary)
- Send EB indicator (primary and secondary)
- FM header usage
- Brackets usage and reset state
- Bracket termination rule
- Alternate Code Set Allowed indicator
- Normal-flow send/receive mode
- Recovery responsibility
- Contention winner/loser
- Half-duplex flip-flop reset states

FM PROFILE 5

Profile 5 specifies the following session rules:

- Only single-RU chains allowed.
- Primary half-session uses delayed request mode.
- Secondary half-session uses delayed request mode and delayed response mode.
- Primary half-session chains indicate definite response.
- Secondary half-session chains indicate no-response or definite response.
- No compression.
- No DFC RUs.
- No FM headers.
- No brackets.
- No alternate code.
- Normal-flow send/receive mode is FDX.

FM PROFILE 6

Profile 6 specifies the following session rules:

- Only single-RU chains allowed.
- Primary and secondary half-sessions use delayed request mode and delayed response mode.
- Primary and secondary half-session chains may indicate definite response, exception response, or no response.
- Primary half-session sends no DFC RUs.
- Secondary half-session may send LUSTAT.
- No FM headers.
- No compression.
- No brackets.
- No alternate code.
- Normal-flow send/receive mode is FDX.

FM PROFILE 7

Profile 7 specifies the following session rules:

Primary LU half-session and secondary LU half-session use immediate response mode.
Primary LU half-session and secondary LU half-session support the following DFC functions:

CANCEL
SIGNAL
LUSTAT
RSHUTD

The FM Usage fields defining the options for Profile 7 are:

Chaining use (primary and secondary)
Request control mode selection (primary and secondary)
Chain response protocol (primary and secondary)
Compression indicator (primary and secondary)
Send EB indicator (primary and secondary)
FM header usage
Brackets usage and reset state
Bracket termination rule
Alternate Code Set Allowed indicator
Normal-flow send/receive mode
Recovery responsibility
Contention winner/loser
Half-duplex flip-flop reset rules

FM PROFILE 17

Profile 17 specifies the following session rules:

Only single-RU chains allowed.
Primary and secondary half-sessions use delayed request mode and delayed response mode.
Primary and secondary half-session chains indicate definite response.
No DFC RUs.
No FM headers.
No compression.
No brackets.
No alternate code.
Normal-flow send/receive mode is FDX.

FM PROFILE 18

Profile 18 specifies the following session rules:

Primary LU half-session and secondary LU half-session use immediate response mode.
Primary LU half-session and secondary LU half-session

support the following DFC functions:

- CANCEL
- SIGNAL
- LUSTAT
- BIS and SBI (allowed only if brackets are used)
- CHASE
- BID and RTR (allowed only if brackets are used)

The FM Usage fields defining the options for Profile 18 are:

- Chaining use (primary and secondary)
- Request control mode selection (primary and secondary)
- Chain response protocol (primary and secondary)
- Compression indicator (primary and secondary)
- Send EB indicator (primary and secondary)
- FM header usage
- Brackets usage and reset state
- Bracket termination rule
- Alternate Code Set Allowed indicator
- Normal-flow send/receive mode
- Recovery responsibility
- Contention winner/loser
- Half-duplex flip-flop reset states

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, SSCP)	(SSCP, PU)	(SSCP, LU)	(LU,LU)
0	no	yes	yes	no
2	no	no	no	yes
3	no	no	no	yes
4	no	no	no	yes
5	no	yes	no	no
6	no	no	yes	no
7	no	no	no	yes
17	yes	no	no	no
18	no	no	no	yes

TRANSMISSION SERVICES (TS) PROFILES

This section describes the transmission services (TS) profiles and their use for the various sessions defined in SNA. Profile numbers not shown are reserved.

Note: If the TS Usage field 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 specifies the following session rules:

- No pacing.
- Identifiers rather than sequence numbers are used on the normal flows (whenever the TH format used includes a sequence number field).
- 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).

This profile does not require the use of the TS Usage field.

TS PROFILE 2

Profile 2 specifies the following session rules:

- Primary-to-secondary and secondary-to-primary normal flows are paced.
- Sequence numbers are used on the normal flows (whenever the TH format used includes a sequence number field).
- CLEAR is supported.
- SDT, RQR, STSN, and CRV are not supported.

The TS Usage subfields defining the options for this profile are:

- Pacing counts
- Maximum RU sizes on the normal flows

TS PROFILE 3

Profile 3 specifies the following session rules:

- Primary-to-secondary and secondary-to-primary normal flows are paced.
- Sequence numbers are used on the normal flows (whenever the TH format used includes a sequence number field).
- CLEAR and SDT are supported.
- RQR and STSN are not supported.
- CRV is supported when session-level cryptography is selected (via a BIND parameter).

The TS Usage subfields defining the options for this profile are:

- Pacing counts
- Maximum RU sizes on the normal flows

TS PROFILE 4

Profile 4 specifies the following session rules:

- Primary-to-secondary and secondary-to-primary normal flows are paced.
- Sequence numbers are used on the normal flows (whenever the TH format used includes a sequence number field).
- SDT, CLEAR, RQR, and STSN are supported.
- CRV is supported when session-level cryptography is selected (via a BIND parameter).

The TS Usage subfields defining the options for this profile are:

- Pacing counts
- Maximum RU sizes on the normal flows

TS PROFILE 5

Profile 5 specifies the following session rules:

- No pacing.
- Sequence numbers are used on normal flows.
- SDT is supported.
- CLEAR, RQR, STSN, and CRV are not supported.
- No maximum RU sizes for the normal flows are specified.

This profile does not require the use of the TS Usage field.

TS PROFILE 7

Profile 7 specifies the following session rules:

- Primary-to-secondary and secondary-to-primary normal flows are paced.
- Sequence numbers are used on the normal flows (whenever the TH format used includes a sequence number field).
- 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 defining the options for this profile are:

- Pacing counts
- Maximum RU sizes on the normal flows

TS PROFILE 17

Profile 17 specifies the following session rules:

- Primary-to-secondary and secondary-to-primary normal flows are paced.
- Identifiers rather than sequence numbers are used on the normal flows.
- SDT is supported.
- STSN and CRV are not supported.
- No maximum RU sizes for the normal flow are specified.

The TS Usage subfields defining the options for this profile are:

- Pacing counts

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, SSCP)	(SSCP, PU)	(SSCP, LU)	(LU,LU)
1	no	yes	yes	no
2	no	no	no	yes
3	no	no	no	yes
4	no	no	no	yes
5	no	yes	no	no
7	no	no	no	yes
17	yes	no	no	no

CROSS-DOMAIN RESOURCE MANAGER (CDRM) PROFILES

The CDRM profile is specified in a control vector carried in ACTCDRM and RSP(ACTCDRM) to define the cross-domain capabilities of an SSCP. CDRM Profile 0 is described here. All other profile numbers are reserved.

CDRM PROFILE 0

Profile 0, along with the CDRM usage fields in the control vector, specifies functional capabilities of the SSCP.

The options specified in the CDRM usage fields for Profile 0 are:

Network name pair session key (X'06') supported
Network address pair session key (X'07') supported
PCID session key (X'05') supported
URC support by the SSCP (and all PLUs within its domain) in cross-domain session initiation (i.e., (1) BINDs issued from all PLUs in this domain carry URC if the INIT specified a URC, and an SLU in the other domain issued the INIT; and (2) the BIND image in CDCINITs issued from the SSCP in this domain carry URC if the INIT specified a URC, and an SLU in this domain issued the INIT)

PHYSICAL UNIT (PU) TYPES

The following PU types are defined (all others are reserved):

PU TYPE 1 (PU_T1)

For all PIUs sent to and received from a PU_T1 node, the transmission header (TH) format is FID3.

PU TYPE 2 (PU_T2)

For all PIUs sent to and received from a PU_T2 node, the transmission header (TH) format is FID2.

PU TYPE 4 (PU_T4)

A PU_T4 node has intermediate and/or boundary function.

The TH format is either:

- FID0 or FID1 for all PIUs transmitted between the PU_T4 and adjacent PU_T4|5 node, if either or both nodes do not support ER and VR protocols.
- FID2 for all PIUs transmitted between the PU_T4 and an adjacent PU_T2 node.
- FID3 for all PIUs transmitted between the PU_T4 and an adjacent PU_T1 node.
- FID4 or FIDF for all PIUs transmitted between the PU_T4 and an adjacent PU_T4|5 node, if both nodes support ER and VR protocols.

PU TYPE 5 (PU_T5)

A PU_T5 is at a node that has intermediate and/or boundary function and also contains an SSCP.

The TH format is either:

- FID0 or FID1 for all PIUs transmitted between the PU_T5 and an adjacent PU_T4|5 node, if either or both nodes do not support ER and VR protocols.
- FID2 for all PIUs transmitted between the PU_T5 and an adjacent PU_T2 node.
- FID3 for all PIUs transmitted between the PU_T5 and an adjacent PU_T1 node.
- FID4 or FIDF for all PIUs transmitted between the PU_T5 and an adjacent PU_T4|5 node, if both nodes support ER and VR protocols.

The sense data included with an EXCEPTION REQUEST (EXR), a negative response, or a send 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 implementation- or end-user-defined data (hereafter referred to as user-defined data). For certain sense codes, user-defined data cannot be included in the sense data (it is never carried in send-check sense data); in its place is sense code specific information, whose format is defined along with the sense code definition, below.

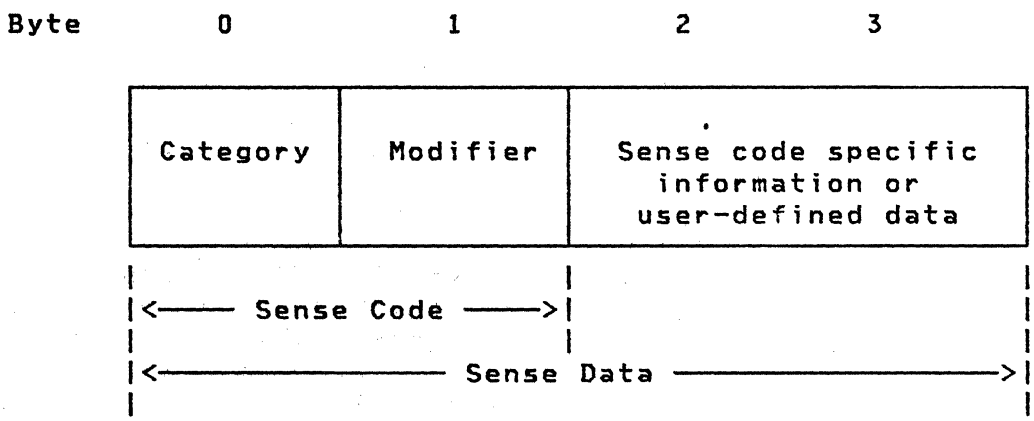


Figure G-1. Sense Data Format

Together, the category and modifier bytes hold the sense code (SNC) defined for the exception condition that has occurred.

The following categories are defined; all others are reserved:

Value	Category
X'80'	Path Error
X'40'	Request Header (RH) Usage Error
X'20'	State Error
X'10'	Request Error
X'08'	Request Reject
X'00'	User Sense Data Only

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'.

The sense codes for the other categories are discussed below. For these categories, a modifier value of X'00' can be used (as an implementation option) when no definition of the exception condition beyond the major category is to be identified.

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. In this case, the NAU services manager receiving the -RSP(Path Error) may deactivate the affected half-session.

Modifier (in hexadecimal):

01 Intermediate Node Failure: Machine or program check in a node providing intermediate function. A response may or may not be possible.

02 Link Failure: Data link failure.

03 NAU Inoperative: The NAU is unable to process requests or responses, e.g., the NAU has been disrupted by an abnormal termination.

04 Unrecognized Destination Address: A node in the path has no routing information for the destination specified by the TH.

05 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.

06 Invalid FID: Invalid FID for the receiving node. (Note 1)

07 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)

08 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.

09 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.

0A Too-Long PIU: Transmission was truncated by a receiving node because the PIU exceeded a maximum length or sufficient buffering was not available.

0B Incomplete TH: Transmission received was shorter than a TH. (Note 1)

0C DCF Error: Data Count field inconsistent with transmission length.

0D Lost Contact: Contact with the link station for which the transmission was intended has been lost, but the link has not failed. If the difference between link failure and loss of contact is not detectable, link failure (X'8002') is sent.

0E Unrecognized Origin: The origin address specified in the TH was not recognized.

0F Invalid Address Combination: The (DAF',OAF') (FID2) combination or the LSID (FID3) specified an invalid type of session, e.g., a PU-LU combination.

10 Segmented RU Length Error: An RU was found to exceed a maximum length, or required buffer allocation that might cause future buffer depletion.

11 ER Inoperative or Undefined: A PIU was received from a subarea node that does not support ER and VR protocols, and the explicit route to the destination is inoperative or undefined.

12 Subarea PU Not Active or Invalid Virtual Route: A session activation request for a peripheral PU or LU cannot be satisfied because there is no active SSCP-PU session for the subarea node providing boundary function support, or the virtual route for the specified SSCP-PU_T1|2 or SSCP-LU session is not the same as that used for the SSCP-PU session of the PU_T1|2's or LU's subarea PU.

13 COS Not Available: A session activation request cannot be satisfied because none of the virtual routes requested for the session is available. This condition may arise because each of the specified virtual routes cannot be activated for one of the following reasons:

- The specified virtual route cannot be mapped to an explicit route to the destination subarea, or the explicit route it is mapped to is not defined.
- The underlying explicit route is not operative.
- The underlying explicit route is operative but cannot be activated.
- The underlying explicit route is active but the virtual route cannot be activated.
- The session must be assigned to a virtual route with an underlying reverse explicit route number of 0, but the virtual route does not meet this criterion.

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.

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.

Modifier (in hexadecimal):

01 Invalid SC or NC RH: The RH of a session control (SC) or network control (NC) request was invalid. For example, an SC RH with pacing request indicator set to 1 is invalid.

02 Reserved.

03 BB Not Allowed: The Begin Bracket indicator (BBI) was specified incorrectly, e.g., BBI=BB with BCI=¬BC.

04 EB Not Allowed: The End Bracket indicator (EBI) was specified incorrectly, e.g., 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.

05 Incomplete RH: Transmission shorter than full TH-RH.

06 Exception Response Not Allowed: Exception response was requested when not permitted.

07 Definite Response Not Allowed: Definite response was requested when not permitted.

08 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.

09 CD Not Allowed: The Change Direction indicator (CDI) was specified incorrectly, e.g., CDI=CD with ECI=¬EC, or CDI=CD with EBI=EB.

0A No-Response Not Allowed: No-response was specified on a request when not permitted. (Used only on EXR.)

0B 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.

0C Brackets Not Supported: The bracket indicators (BBI and EBI) were specified incorrectly, e.g., a bracket indicator was set (BBI=BB or EBI=EB), but brackets are not used for the session.

0D CD Not Supported: The Change-Direction indicator was set, but is not supported.

0E Reserved.

0F 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.

10 Alternate Code Not Supported: The Code Selection indicator (CSI) was set when not supported for the session.

11 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.

12 Incorrect Specification of Request Code: The request code on a response does not match the request code on its corresponding request.

13 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).

14 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 CANCEL request was not specified with DR1I=DR1, DR2I=¬DR2, and ERI=¬ER.

15 Incorrect Use of QRI: The Queued Response indicator (QRI) was specified incorrectly, e.g., QRI=QR on an expedited-flow request.

16 Incorrect Use of EDI: The Enciphered Data indicator (EDI) was specified incorrectly, e.g., EDI=ED on a DFC request.

17 Incorrect Use of PDI: The Padded Data indicator (PDI) was specified incorrectly, e.g., PDI=PD on a DFC request.

STATE ERROR (CATEGORY CODE = X'20')

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.

Modifier (in hexadecimal):

01 Sequence Number: Sequence number received on normal-flow request was not 1 greater than the last.

02 Chaining: Error in the sequence of the chain indicator settings (BCI, ECI), such as first, middle, first.

03 Bracket: Error resulting from failure of sender to enforce bracket rules for session. (This error does not apply to contention or race conditions.)

04 Direction: Error resulting from a normal-flow request received while the half-duplex flip-flop state was not-receive, (*S,-R). (Contrast this sense code with X'081B', which signals a race condition.)

05 Data Traffic Reset: An FMD or normal-flow DFC request received by a half-session whose session activation state was active, but whose data traffic state was not active

06 Data Traffic Quiesced: An FMD or DFC request received from a half-session that has sent QUIESCE COMPLETE or SHUTDOWN COMPLETE and has not responded to RELEASE QUIESCE.

07 Data Traffic Not Reset: A session control request (e.g., STSN), allowed only while the data traffic state is reset, was received while the data traffic state was not reset.

08 No Begin Bracket: A BID or an FMD request specifying BBI=BB was received after the receiver had previously sent a positive response to BRACKET INITIATION STOPPED.

09 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. There is no user data associated with this sense code.

0A Immediate Request Mode Error: The immediate request mode protocol has been violated by the request.

0B Queued Response Error: The Queued Response protocol has been violated by a request, i.e., QRI=¬Qk when an outstanding request had QRI=QR.

0C ERP Sync Event Error: The ERP sync event protocol has been violated.

0D Response Owed Before Sending Request: An attempt has been made in half-duplex (flip-flop or contention) send/receive mode to send a normal-flow request when a response to a previously received request has not yet been sent.

REQUEST ERROR (CATEGORY CODE = X'10')

This category indicates that the RU was delivered to the intended half-session component, but could not be interpreted or processed. This condition represents a mismatch of half-session capabilities.

Modifier (in hexadecimal):

01 RU Data Error: Data in the request RU is not acceptable to the receiving FMDS component; for example, a character code is not in the set supported, a formatted data field is not acceptable to presentation services, or a required name in the request has been omitted.

02 RU Length Error: The request RU was too long or too short.

03 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.

Bytes 2 and 3 following the sense code are not used for user-defined data; they contain sense-code specific information. Settings allowed are:

0000 Function requested is not supported.

6022 The resource identified by the destination program name (DPN) is not supported.

6003 The resource identified by the primary resource name (PRN) is not supported.

(Note: This code can also be used instead of sense code X'0826'.)

04 Reserved.

05 Parameter Error: A parameter modifying a control function is invalid, or outside the range allowed by the receiver.

06 Reserved.

07 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 with byte 0 was not set to a defined value, or byte 1 was not set to an NS category supported by the receiver.

08 Invalid FM Header: The FM header was not understood or translatable by the receiver, or an FM header was expected but not present.

Bytes 2 and 3 following the sense code are not used for user-defined data; they contain sense-code specific information as defined in SNA LU-LU Session Types.

09 Format Group Not Selected: No format group was selected before issuing a Present Absolute or Present Relative Format structured field to a display.

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.

Modifier (in hexadecimal):

01 Resource Not Available: The LU, PU, or link specified in an RU is not available.

02 Intervention Required: Forms or cards are required at an output device, or a device is temporarily in local mode, or other conditions require intervention.

03 Missing Password: The required password was not supplied.

04 Invalid Password: Password was not valid.

05 Session Limit Exceeded: The requested session cannot be activated, as one of the NAUs is at its session limit. Applies to ACTCDRM, INIT, BIND, and CINIT requests.

06 Resource Unknown: The request contained a name or address not identifying a PU, LU, link, or link station known to the receiver.

07 Resource Not Available--LUSTAT Forthcoming: A subsidiary device will be unavailable for an indeterminate period of time. LUSTAT will be sent when the device becomes available.

08 Invalid Contents ID: The contents ID contained on the ACTCDRM request was found to be invalid.

09 Mode Inconsistency: The requested function cannot be performed in the present state of the receiver.

0A 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.)

0B Bracket Race Error: Loss of contention within the bracket protocol. Arises when bracket initiation/termination by both NAUs is allowed.

0C Procedure Not Supported: A procedure (Test, Trace, IPL, REQMS type) specified in an RU is not supported by the receiver.

0D NAU Contention: A request to activate a session was received while the receiving half-session was awaiting a response to a previously sent activation request for the same session; e.g., the SSCP receives an ACTCDRM from the other SSCP before it receives the response for an ACTCDRM that it sent to the other SSCP and the SSCP ID in the received ACTCDRM was less than or equal to the SSCP ID in the ACTCDRM previously sent.

0E NAU Not Authorized: The requesting NAU does not have access to the requested resource.

0F End User Not Authorized: The requesting end user does not have access to the requested resource.

10 Missing Requester ID: The required requester ID was missing.

11 Break: Asks the receiver of this sense code to terminate the present chain with CANCEL or with an FMD request carrying EC. The half-session sending the Break sense code enters chain-purge state when Break is sent.

12 Insufficient Resource: Receiver cannot act on the request because of a temporary lack of resources.

13 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.

14 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.

15 Function Active: A request to activate a network element or procedure was received, but the element or procedure was already active.

16 Function Inactive: A request to deactivate a network element or procedure was received, but the element or procedure was not active.

17 Link Inactive: A request requires the use of a link, but the link is not active.

18 Link Procedure in Process: CONTACT, DISCONTACT, IPL, or other link procedure in progress when a conflicting request was received.

19 RTR Not Required: Receiver of READY TO RECEIVE has nothing to send.

1A Request Sequence Error: Invalid sequence of requests.

1B Receiver in Transmit Mode: A race condition: normal-flow request received while the half-duplex contention state was not-receive, (*S,¬R), or while resources (such as buffers) necessary for handling normal-flow data were unavailable. (Contrast this sense code with X'2004', which signals a protocol violation.)

1C Request Not Executable: The requested function cannot be executed, because of a permanent error condition in the receiver.

1D Invalid Station/SSCP ID: The Station ID or SSCP ID in the request was found to be invalid.

1E Session Reference Error: The request contained reference to a half-session that was neither active nor in the process of being activated (generally applies to network services requests).

1F Reserved.

20 Control Vector Error: Invalid data for the control vector specified by the target network address and key.

21 Invalid Session Parameters: Session parameters were not valid or not supported by the half-session whose activation was requested.

22 Link Procedure Failure: A link-level procedure has failed due to link equipment failure, loss of contact with a link station, or an invalid response to a link command. (This is not a path error, since the request being rejected was delivered to its destination.)

23 Unknown Control Vector: The control vector specified by a network address and key is not known to the receiver.

24 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.

25 Component Not Available: The LU component (a device indicated by an FM header) is not available.

26 FM Function Not Supported: A function requested in an FMD RU is not supported by the receiver.

27 Intermittent Error--Retry Requested: An error at the receiver caused an RU to be lost. The error is not permanent, and retry of the RU (or chain) is requested.

28 Reply Not Allowed: A request requires a normal-flow reply, but the outbound data flow for this half-session is quiesced or shut down, and there is no delayed reply capability.

29 Change Direction Required: A request requires a normal-flow reply, but the half-duplex flip-flop state is not-send, (¬S,*R), CD was not set on the request, and there is no delayed reply capability.

2A Presentation Space Alteration: Presentation space altered by the end user while the half-duplex state was not-send, (¬S,*R): request executed.

2B Presentation Space Integrity Lost: Presentation space integrity lost (e.g., cleared or changed) because of a transient condition--for example, because of a transient hardware error or an end user action such as allowing presentation services to be used by the SSCP. (Note: The end-user action described under X'082A' and X'084A' is excluded here.)

2C 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.

2D LU Busy: The LU resources needed to process the request are being used; for example, the LU resources needed to process the request received from the SSCP are being used for the LU-LU session.

2E Intervention Required at LU Subsidiary Device: A condition requiring intervention, such as out of paper, or power-off, or cover interlock open, exists at a subsidiary device.

2F Request Not Executable because of LU Subsidiary Device: The requested function cannot be executed, due to a permanent error condition in one or more of the receiver's subsidiary devices.

30 Reserved

31 LU Component Disconnected: An LU component is not available because of power off or some other disconnecting condition.

32 Invalid Count Field: A count field contained in the request indicates a value too long or too short to be interpreted by the receiver, or the count field is inconsistent with the length of the remaining fields. Bytes 2 and 3 following the sense code are not used for user-defined data; they contain a binary count that indexes (zero-origin) the first byte of the invalid count field.

33 Invalid Parameter (with Pointer and Complemented Byte): one or more parameters contained in fixed- or variable-length fields of the request are invalid or not supported by the NAU that received the request. Bytes 2 and 3 following the sense code are not used for user-defined data. Byte 2 contains a binary value that indexes (zero-origin) the first byte that contained an invalid parameter. Byte 3 contains a transform of the first byte that contained an invalid parameter: the bits that constitute the one or more invalid parameters are complemented, and all other bits are copied.

34 RPO Not Initiated: A power-off procedure for the specified node was not initiated because one or more other SSCPs have contacted the node, or because a CONTACT, DUMP, IPL, or DISCONTACT procedure is in progress for that node.

35 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 following the sense code are not used for user-defined data; they contain a two-byte binary count that indexes (zero-origin) the first byte of the fixed- or variable-length field having invalid contents.

36 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.

37 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.

38 Reserved

39 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.

3A 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.

3B Invalid PCID: An invalid PCID was received, e.g., one containing an invalid network address of the SSCP of the initiating LU (ILU) or terminating LU (TLU), has been received in CDINIT, INIT-OTHER-CD, CDTERM, or TERM-OTHER-CD; or a PCID that does not identify a previously queued request has been received in CDINIT (Dequeue) or INIT-OTHER-CD (Dequeue); or, a PCID that cannot be associated with the PCID of any previously processed CDINIT has been received on CDCINIT.

3C Domain Takedown Contention: While waiting for a response to a CDTAKED, a CDTAKED request is received by the SSCP containing the SSCP-SSCP primary half-session. Contention is resolved by giving preference to the CDTAKED sent by the primary half-session.

3D Dequeue Retry Unsuccessful--Removed from Queue: The SSCP cannot successfully honor a CDINIT(Dequeue) request (which specifies "leave on queue if dequeue-retry is unsuccessful") to dequeue and process a previously queued CDINIT request (e.g., because the LU in its domain is still not available for the specified session), and removes the queued CDINIT request from its queue.

3E Reserved

3F Terminate Contention: While waiting for a response to a CDTERM, a CDTERM is received by the SSCP of the SLU. Contention is resolved by giving preference to the CDTERM sent by the SSCP of the SLU.

40 Procedure Invalid for Resource: The named procedure is not supported in the receiver for this type of resource (e.g., (1) SETCV specifies boundary function support for a type 1 node but the capability is not supported by the receiving node, or (2) the PU receiving an EXECTEST or TESTMODE is not the primary PU for the target link.)

41 Duplicate Network Address: In a cross-domain LU-LU session initiation request, the SSCP of the DLU determines that the OLU network address specified in the CDINIT request is a duplicate of an LU network address assigned to a different LU name.

42 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 TLU and the SSCP of the OLU do not have an active session with each other, and therefore TERM-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.

43 Required FMDS Synchronization Not Supplied: For example, a secondary LU (LU-LU session type 2 or 3) received a request with Write Control Code = Start Print, along with RQE and ¬CD.

44 Initiation Dequeue Contention: While waiting for a response to a CDINIT(Dequeue), a CDINIT(Dequeue) is received by the SSCP of the SLU. Contention is resolved by giving preference to the CDINIT(Dequeue) sent by the SSCP of the SLU.

45 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.)

46 ERP Message Forthcoming: The received request was rejected for a reason to be specified in a forthcoming request.

47 Restart Mismatch: Sent in response to STSN or SDT or BIND to indicate that the secondary half-session is trying to execute a resynchronizing restart but has received insufficient or incorrect information.

48 Cryptography Function Inoperative: The receiver of a request was not able to decipher the request because of a malfunction in its cryptography facility.

49 Reserved

4A Presentation Space Alteration: The presentation space was altered by the end user while the half-duplex state was not-send, (¬S,*R); request not executed.

4B 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 following the sense code are not used for user-defined data; they contain sense-code specific information. Settings allowed are:

0000 Requested resources are not available.

6022 The resource identified by the destination program name (DPN) is not supported.

6003 The resource identified by the primary resource name (PRN) is not supported.

4C Permanent Insufficient Resource: Receiver cannot act on the request because resources required to honor the request are permanently unavailable.

4D 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.

4E Invalid Session Parameters--PRI: A positive response to an activation request (e.g., BIND) was received and was changed to a negative response due to invalid session parameters carried in the response. The services manager receiving the response will send a deactivation request for the corresponding session.

4F-
50 Reserved

51 Session Busy: Another session that is needed to complete the function being requested on this session (e.g., to forward an NS RU embedded in a FORWARD request) is temporarily unavailable.

52 Session with Larger Activation Request Sequence Identifier Already Active: A session has already been activated for the subject destination-origin pair by a session activation request that carried a larger activation request identifier than the current request; the current request (ACTPU or ACTCDRM) is refused.

53 TERMINATE(Cleanup) Required: The SSCP cannot process the termination request, as it requires cross-domain SSCP-SSCP services that are not available. (The corresponding SSCP-SSCP session is not active.) TERMINATE(Cleanup) is required.

54-
55 Reserved

56 SSCP-SSCP Session Lost: Carried in the Sense Data field in a NOTIFY or NSPE sent to an ILU or SSCP(ILU) to indicate that the activation of the LU-LU session either cannot be completed or is uncertain because the SSCP-SSCP session between the two domains has been lost. (This sense code appears only in NOTIFY or NSPE, not in a negative response. Another sense code, X'0842', is used on a negative response to signal the condition when the condition is known at the time the response, e.g., to INIT, is prepared.)

57 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.

58 Reserved

59 REQECHO Data Length Error: The specified length of data to be echoed (in REQECHO) violates the maximum RU size limit for the target LU.

5A-
5F Reserved

60 Function Not Supported--Continue Session: The function requested is not supported; the function may have been specified by a request code or some other field, control character, or graphic character in an RU. Bytes 2-3 following the sense code are not used for user defined data; they contain a two-byte binary count that indexes (zero-origin) the first byte in which an error was detected. This sense code is used to request that the session continue, thereby ignoring the error.

61 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 following the sense code contain X'0000' if the COS name was generated by the SSCP or X'0001' if specified by the ILU.

62 Medium Presentation Space Recovery: An error has occurred on the current presentation space. Recovery consists of restarting at the top of the current presentation space. The sequence number returned is of the RU in effect at the top of the current presentation space. Bytes 2 and 3 following the sense code contain the byte offset from the beginning of the RU to the first byte of the RU that is displayed at the top of the current presentation space.

63 Referenced Local Character Set Identifier (LCID) Not Found: A referenced character set does not exist.

64 Function Abort: A loop will occur upon reexecution; the request sender should not send the same data.

65 Function Abort: Sender is responsible to detect the loop.

66 Function Abort: Receiver is responsible to detect the loop.

67 Sync Event Response: Indicates a negative response to a sync event.

68 No Panels Loaded: Referenced format not found because no panels are loaded for the display.

69 Panel Not Loaded: The referenced panel is not loaded for the display.

70 Reserved

71 Read Partition State Error: A Read Partition structured field was received while the display was in the retry state.

72 Orderly Deactivation Refused: An NC_DACTVR(Orderly) request has been received, but sessions are assigned to the VR and it will not be deactivated.

73 Virtual Route Not Defined: There is no ERN designated to support this VRN.

74 ER Not in a Valid State: The ER supporting the requested VR is not in a state allowing VR activation.

75 Incorrect or Undefined Explicit Route Requested: The reverse ERNs specified in the NC_ACTVR do not contain the ERN defined to be used for the VR requested, or the ERN designated to be used for the VR is not defined.

76 Nonreversible Explicit Route Requested: The ERN used by the NC_ACTVR does not use the same sequence of transmission groups (in reverse order) as the ERN that should be used for the RSP(NC_ACTVR).

77 Reserved

78 Insufficient Storage: The storage resource required for a data format is not available.

79 Storage Medium Error: A permanent error has occurred involving a storage medium.

7A Format Processing Error: A processing error occurred during data formatting.

This appendix defines the Format and Protocol Language (FAPL), a formal descriptive language used throughout the book, and the conventions used in finite-state machine state-transition graphs. FAPL is a simple extension of the syntax and semantics of PL/I.

The appendix also provides background information and definitions related to finite-state machines. It is assumed that the reader has a basic knowledge of set theory and Boolean logic.

The following set theory and logic symbols are used throughout this appendix.

Symbol	Meaning
&	and
¬	not
-->	into
=>	implies
*	any value allowed (a "don't care" condition)
:	maps
x	Cartesian product (for sets)

FINITE-STATE MACHINES

A finite-state machine (FSM) is a simple mechanism based on the need to remember a limited amount of information. An FSM can remember its current state--and nothing else. The meaning of each state depends on the purpose of the FSM, but generally states are used to remember what has occurred previously. Given its current state and some input, the FSM changes state and produces some output. The FSM may "move" to the current state as well as to any other state and the output produced may be null.

Given the current state and an input, the FSM switches to the next state and produces the appropriate output using the next-state function and output function, respectively. The next state that the FSM enters is defined by the next-state function, which selects the next state based only on the current state and the input to the FSM. Likewise, the output function determines the actual output using only the current state and the input.

Both the input and the output may be complex; the input may consist of several different types of information and a single state transition may produce several outputs.

There are two mechanisms used in this book to describe FSMs: the state-transition matrix and the state-transition graph. The first method is defined within the "FAP Language" section and the second is in the "State-Transition Graphs" section.

The remainder of this section defines FSMs formally.

DISCRETE TIME

Each system described in this book operates in discrete time; i.e., associated with each system is a set of discrete sample times

$$T = \{t1, t2, t3, \ldots\}$$

at which the system variables or their transitions are defined.

PULSED AND STATIC VARIABLES

Two types of system variables--pulsed and static--are defined. A variable is pulsed if it is defined only at the sample times in

$T = \{t1, t2, \ldots\}$.

A variable is static if it is constant between sample times and can change values only immediately following each sample time. This can be expressed more formally as follows: Let V(t) be a static variable, then

$$ti < t \leq t(i+1) \Rightarrow V(t) = V(t(i+1)).$$

The next section shows that the basic input and output variables of a finite-state machine are pulsed, and the state variable of a finite-state machine is static.

BASIC FINITE-STATE MACHINE DEFINITION

A finite-state machine (FSM) is a system operating in discrete time and consisting of five well-defined entities:

$$FSM = \langle S, X, Z, FNS, FOUT \rangle$$

where

S = a finite set of states
X = a finite set of inputs
Z = a finite set of outputs
FNS = the next-state function; $FNS:(S \times X) \rightarrow S$
FOUT = the output function; $FOUT:(S \times X) \rightarrow Z$.

The system operates at time ti as shown in Figure N-1.

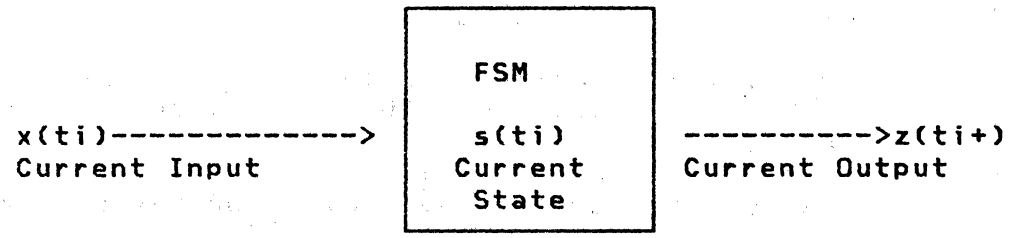


Figure N-1. Finite-State Machine

A set of discrete sampling times T={t1, t2,...} is defined. For any ti in T, the system input and state (respectively, x(ti) and s(ti)) are well defined. The output of the system and the state transition are governed by the following two equations:

$$z(ti+) = FOUT(s(ti),x(ti)) = Current\ Output$$

$$s(t(i+1)) = s(t)\ for\ all\ ti<t\leq t(i+1) = FNS\ (s(ti),x(ti)) = Next\ State$$

The FSM output occurs infinitesimally later than the causal input; hence, the notation z(ti+) for current output. The next-state variable is static and changes (if at all) immediately following each input arrival.

EXTENSIONS OF THE BASIC DEFINITIONS

Null Output

It is convenient to include the null output (no output) in an FSM output set so that FSM state transitions that produce no output can be defined.

Multiple-Stream Outputs and Routing

The basic output of an FSM is a pulsed variable for which the values all belong to an output set Z and are emitted on a single output stream. This output concept can be extended in two ways:

- If an FSM has Z = Z1 x Z2 x ... x Zn (Cartesian product of n output component sets), then the FSM is allowed to separate and route outputs by component to the n output streams as shown in Figure N-2.

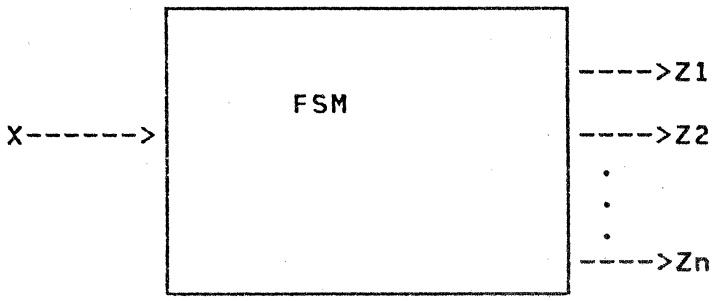


Figure N-2. Multiple-Stream Outputs

- Any function of the state set (a static variable) may constitute a static FSM output stream. FOUT can then be viewed as having two components:

FOUT.PULSED: SxX --> Z.PULSED

FOUT.STATIC: S --> Z.STATIC

State Attributes

Specific values of static output variables are referred to as FSM attributes. FSM attributes are used to group a set of states that share common characteristics. For example, in a given FSM, each state may have associated with it the attribute S or ¬S indicating whether, when in this state, the associated half-session can send a request. The association of attributes with states simplifies testing if an FSM is in any of several states.

Multiple-Stream Inputs and Routing

The basic input of an FSM is a pulsed variable, in which the pulsed values belong to an input set X and arrive on a single input stream. This input concept can be extended in two ways:

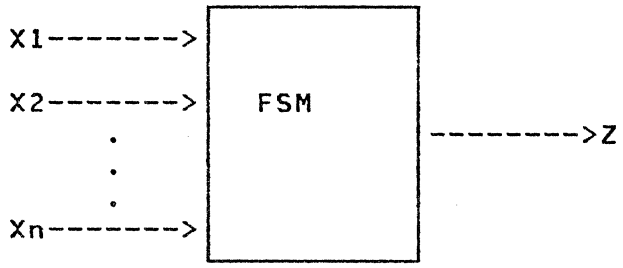


Figure N-3. Multiple-Stream Inputs

- In Figure N-3 let X1, X2,..., Xn be a collection of input sets; each Xi corresponds to the inputs capable of arriving on the i-th input stream. X = X1 x X2 x ... x Xn is then the single input set corresponding to all possible input combinations taken over the set of input streams. The input information includes input values as well as the correspondence between value and stream.

  In the FSM systems defined in this book, the multiple input streams are always taken to operate independently (asynchronously); pulsed inputs in the multiple input-stream set can never arrive simultaneously.

- Static variables can be incorporated into the input of multistream-input FSMs as follows:

  -- All the static input streams may combine, under the Cartesian product, with the static state variable to form a new static variable S' that can be used as an argument of FOUT.STATIC (the static

component of FOUT); i.e., if X1,..., Xk are static input streams, then S' = S x X1 x X2 x ... x Xk is a static variable and FOUT.STATIC: S'-->Z.STATIC.

-- If one or more Xi's are pulsed variables, then X = X1 x X2 x ... x Xn is also a pulsed variable. The set of sample times associated with the arrival of elements in X corresponds to the union of sample time sets associated with the individual pulsed inputs. If the pulsed inputs arrive asynchronously, then the only observed input combinations in X will be those for which static components are non-null and precisely one pulsed variable is non-null.

State of an FSM

The meaning of an FSM state is the effect that the state has on the input/output behavior of the FSM; the meaning of a state is conveyed completely by FNS and FOUT.

If N is the name of an FSM and E denotes a subset of Z.STATIC, N's static output set, then it is possible to test whether the current static output of FSM N is in subset E.

A commonly used special case of this facility occurs when the static output function is the current FSM state; i.e., FOUT.STATIC(cs) = cs. If E denotes a singleton subset (a single state), then testing whether the current static output of FSM N is in subset E is equivalent to testing whether N is in state E.

Note that any of these tests produces a Boolean value; at any sample time it is either true or false. As such, it can be used as an argument of a Boolean function.

The Reset Convention

All the FSMs defined in this manual conform to the following convention.

Consider an FSM = <S,X,Z,FNS,FOUT>:

- Precisely one of the states in S is identified as the RESET state.
- X contains (reset) as a valid pulsed input.
- For any state s in S, FNS(s,reset) = RESET.
- FOUT.PULSED (s,reset) = null if the (reset) input is not explicitly shown on the FSM. If the (reset) input is explicitly shown, FOUT.PULSED (s,reset) may be null or non-null.

FORMAT AND PROTOCOL LANGUAGE (FAPL)

The Format and Protocol Language (FAPL) provides extensions to PL/I in the form of additional statements, a matrix form FSM representation, and some syntax relaxation to facilitate SNA definition in an executable form.

This document assumes prior familiarity with PL/I. The IBM OS PL/I Checkout and Optimizing Compilers: Language Reference Manual, GC33-0009, provides a complete definition of PL/I.

The executable form of the architecture definition is not intended to be a storage or execution time efficient implementation of the architecture; priority is given to readability.

PL/I SUBSET USED IN FAPL

Only a small subset of PL/I statements is used in FAPL:

assignment	IF...THEN...ELSE
CALL	PROCEDURE...END
DECLARE	RETURN
DO...END	SELECT...END

The only PL/I data types permitted in FAPL are:

FIXED BINARY	fixed length CHARACTER
POINTER	fixed length BIT
structures	one-dimension arrays

SYNTAX NOTATION

The syntax notation used in this appendix to describe the syntax of the FAPL statements is defined as follows:

```
[       ]
|       |   Brackets indicate an optional parameter.
|       |   If not specified, the underlined value
|       |   becomes the default value.
L       J
```

```
[       ]
|       |   Braces indicate a parameter that must be
<       >   specified.  The possible values are
|       |   listed within the braces.
L       J
```

Choices are indicated by vertical stacking within the brackets or braces; unlike the PL/I syntax notation (described in the PL/I Reference Manual), the vertical stroke (|) is not used in the FAPL syntax notation. (However, it is an allowable operator within the FAPL language itself.)

Upper case symbols indicate words that are used as shown. These are the keywords of the language and define the operation to be performed by the statement or built-in function. Lower case symbols indicate a field that is replaced with the correct value in a statement. These are called the parameters of the statements and define the objects on which the operation is performed.

Brackets and braces are omitted in the actual usage of a statement, but parentheses are retained where they are shown.

EXTENSIONS TO PL/I

Extended Comparisons

Syntax:

```
IF v [¬]= ( w1 [| w2] ... ) THEN ...
```

where v and w are variable names or constants (general expressions are not permitted).

Semantics:

FAPL allows comparisons between one variable or constant and several other variables or constants within one comparison expression. This is translated to the PL/I equivalent:

```
IF v = w1 |
   v = w2 |
   ...       THEN ...
```

Example:

```
IF A = ( B | C | D ) THEN ...
```

Representation of Bit Strings

Syntax:

```
X'h...'
B'b...'
```

where h is a hexadecimal digit (0-9,A-F) and b is a binary digit (0,1).

Semantics:

String constant identifiers (i.e., "X" and "B") are placed on the left of the constant as indicated above, rather than on the right as in PL/I. Bit-string constants that are 4n-bits long (n = 1,2,...) can be represented in FAPL by a series of hexadecimal digits, where each hexadecimal digit represents four bits. The hexadecimal and equivalent binary representations of a bit string are used interchangeably in FAPL.

Example:

```
X'086C' is equivalent to B'0000100001101100'.
```

Reserved Bits in Data Structures

Syntax:

```
RESERVED
```

Semantics:

FAPL allows the use of "RESERVED" in place of a variable name to name a component of a structure that is set to 0 and is not referenced.

Example:

```
DCL 1 A,
      2 B        BIT(4),
      2 RESERVED BIT(2),
      2 C        BIT(2);
```

CONSTANT Attribute

Syntax:

```
CONSTANT(i)
```

where i is a literal. The syntax for specifying i is the same as for the PL/I INITIAL attribute with all extensions as described in "Representation of Bit Strings."

Semantics:

The CONSTANT attribute defines a variable's value when that value is not changed throughout execution. A variable defined as CONSTANT is never assigned a value during execution.

Examples:

```
DCL 1 CONST,
      2 A  BIT(2)     CONSTANT(B'01'),
      2 B  FIXED BIN  CONSTANT(1);
```

GENERIC Attribute

Syntax:

```
DCL #g GENERIC [ENTRY(p[,p]...)] [RETURNS(d)]

                                    VALUES(v[,v]...);
```

where g is a variable name. The pound sign (#) is always the initial character of a GENERIC name; p and d are valid type specifications, and v is a variable name beginning with the string g or is the keyword NO_OP.

Semantics:

The GENERIC attribute defines a variable as a generic name for a procedure name or FSM name. Generic variables allow generic references to FSMs and procedures. A generic name reference implies that the specific name reference is determined at execution time by using the name that has most recently been set into the generic variable. A GENERIC variable may be used wherever one of its values would be acceptable. The VALUES attribute is used to specify the valid names that can be set into the generic variable. The special name NO_OP may also be used to indicate that if this GENERIC procedure or FSM is invoked, no routine will be called.

When assigning a value to a generic variable, the value is written as a procedure name, i.e., it is not enclosed in quotation marks.

All names that may be assigned to a generic FSM, begin with the generic name as the first part of their name. For a generic procedure this is not a requirement; the names in the VALUES clause determine those procedure names that may be assigned. The "#" and "#FSM_" strings are used to prefix procedure-name generic variables and FSM-name generic variables, respectively.

The ENTRY and RETURNS attributes are used with the GENERIC attribute if the procedures named in the VALUES attribute use entry parameters or return values, respectively.

Examples:

```
DCL #FSM_A GENERIC VALUES(FSM_AA,FSM_AB,FSM_AC);

DCL #ADD GENERIC ENTRY(PTR) RETURNS(FIXED BIN)
                 VALUES(ADDNAME,ADDNUMBER,INSERTID);
```

REFER Option

Syntax:

```
REFER (v)
```

where v is a variable name.

Semantics:

The REFER option is used to indicate a field that contains the upper bound of an array or the length of a character string; v is the name of the element of the same based structure or entity that contains the count.

Examples:

```
DCL 1 NUMLISTS BASED (RU_PTR),
      2 LISTLEN BIT(8),
      2 LIST_OF_DATA(1:REFER(LISTLEN)) BIT(64);

DCL 1 VSTRING BASED (STING_PTR),
      2 STRINGLEN FIXED BIN(15),
      2 STRING CHAR(REFER(STRINGLEN));
```

Arrays with Unspecified Length

Syntax:

```
                ┌   ┐
                │ a ; │
n i (lb:*) <    │     >
                │ , │
                └   ┘
```

where n is a level number, i is an identifier, lb is the lower bound of the array (either 0 or 1), and a is an attribute list.

Semantics:

This notation is used when the length of the array is known by context, but does not correspond to any specific field. It can be used as the last component of a based structure or an entity.

Example:

```
DCL 1 RESPONSE BASED,
      2 FIXED_DATA CHAR(4),
      2 FORMAT_SPECIFIC_DATA(0:*) CHAR(10);
```

Character Strings with Unspecified Length

Syntax:

```
n i CHAR(*);
```

where n is a level number and i is an identifier.

Semantics:

This notation is used when the length of the character string is known by context, but does not correspond to any specific field. It can be used as the last field in a based structure or entity.

Example:

```
DCL 1 REQUEST BASED,
      2 FIXED_DATA CHAR(8),
      2 FORMAT_SPECIFIC_DATA CHAR(*);
```

Substring Notation

Syntax:

```
v([x1,]x2:x3)
```

where v is a variable name declared with either the BIT or the CHARACTER attribute and x1, x2, and x3 are integer expressions.

Semantics:

A substring within a character-string variable or a bit-string variable is addressed using this notation. The string variable or array of string variables from which the substring is taken is the variable v. The integer expressions represent the following: x1 is present if the variable is an array and it indicates the array element being addressed, x2 indicates the starting position of the substring in the variable, and x3 indicates the ending position of the substring in the variable.

Zero origin is used to number the bits or characters within a string variable in FAPL.

Examples:

```
A(3,2:4)
A(1:5)
A(3:3)
```

SELECT Statement

Syntax:

```
          ┌         ┐
          |INORDER  |
SELECT    <         > [(x)];
          |ANYORDER |
          └         ┘
```

where x is an expression, possible values of which are given in the WHEN clauses. The FAPL SELECT statement uses the PL/I syntax for SELECT except for insertion of the keyword, INORDER or ANYORDER, immediately following the SELECT keyword.

Semantics:

The PL/I SELECT statement is extended to include the keywords, INORDER or ANYORDER, to convey information to an implementer. INORDER indicates that the WHEN clauses are not mutually exclusive, and are to be implemented in the order shown. ANYORDER indicates that the WHEN clauses are mutually exclusive, and may be implemented in any order that optimizes execution time.

In either case, the OTHERWISE clause is executed only if no WHEN clause is true.

Examples:

```
SELECT INORDER;
  WHEN (FLOW = NORMAL)
    CALL NORMAL_MU;
  WHEN (TYPE = REQUEST)
    CALL EXPEDITED_RQ;
  OTHERWISE
    CALL EXPEDITED_RSP;
END;

SELECT ANYORDER (CATEGORY);
  WHEN (SC)
    SEND MU TO SC_PROCESS;
  WHEN (DFC)
    INSERT MU LAST IN Q_DFC;
  WHEN (FMD)
    DO;
      CALL FMD_PROCESS;
      DISCARD MU;
    END;
  END;
```

RESTRICTIONS TO PL/I DATA TYPES

Binary Numbers

All FIXED BINARY variables represent unsigned integers and the precision associated with the variable indicates the number of bits occupied.

Attributes

The ENTRY attribute is allowed only in declarations using GENERIC.

The INITIAL attribute is never used in FAPL.

Arrays

FAPL arrays are limited to one dimension and always have a 0 or 1 lower bound. The lower bound is always stated explicitly.

FAPL NAMES

Name Lengths

The PL/I name limit of 31 characters is observed. In some cases, a name has a smaller character length limit because of implied names (e.g., entity names, which imply a pointer name of the form, entity_name_PTR, have a name-size limit of 27 characters).

Qualified Names

This book follows a naming convention using qualifiers separated by periods to denote more specific factor components of composite protocol machines. In block diagrams, component submachines are shown as blocks within a larger block that represents the composite machine. DFC.RCV and DFC.SEND, for example, are inner blocks within the DFC block.

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 also applies to phrases indentifying state names and FAPL variables.

These FAPL conventions for use of the period and underscore complement the PL/I use, i.e., in PL/I a period denotes a qualifier for a name defined within a structure; FAPL allows both functional (procedural) and data structure decomposition to be shown.

RESERVED KEYWORDS

The following keywords are reserved in FAPL, and are not used as variable or procedure names:

Attribute Names: CONSTANT, GENERIC, VALUES

Statement Names: CONTROL_BLOCK_DEFINITION, CREATE, DESTROY, ENTITY, FIND, INSERT, LOCK, NEWLIST, PURGE, REMOVE, SCAN, SCANEND, SEND, UNLOCK

Function Names: DISPATCHED_BY, EMPTY, FIRST_ENTRY, INPUT, LAST_ENTRY, NEXT_ENTRY, PREV_ENTRY, SEND_OR_RECEIVE_CHECK

Other Names: DESTINATION, FSM_DEFINITION, FSM_INPUT_DEFINITION, PROCNAME, RESERVED

Any variable beginning with the prefix "FSM_" is assumed to be an FSM name; no other variable type begins with these four characters.

LIST PROCESSING

FAPL Facilities

A FAPL list consists of list entries circularly chained, using forward and backward pointers. Statements exist to add and remove entries from lists without concern for the chaining logic.

A list is created with the NEWLIST statement. Storage is allocated for a list header and a pointer to this list header is set into the list name pointer. The list name is declared as a pointer variable within the appropriate control block. List references are always by a pointer to the list header. Any pointer variable may be set from the list name pointer and be used subsequently to refer to the list.

List entries consist of entities defined with the ENTITY statement and created with the CREATE statement. Each created entity is located by a procedure-defined pointer

variable or by the implied pointer, "entity_name_PTR." Only one type of entity is used on any particular list and a particular copy of an entity resides on only one list at any one time. Reference to an entity is by a pointer to it.

Lists are managed using a FIFO, priority, or user-programmed discipline. When using the first-in-first-out mode, the INSERT statement adds an entity to the end of the list. In priority mode (indicated with the BY_ASCENDING and BY_DESCENDING keywords), the INSERT statement inserts an entity based on the numeric value of a specified field within the entity to be added. The FIRST option of the REMOVE statement removes an entity from the top of the list. An example of the order of entities within a list when using the BY_ASCENDING and BY_DESCENDING options is shown in Figure N-4. The priority field values within the entities are shown.

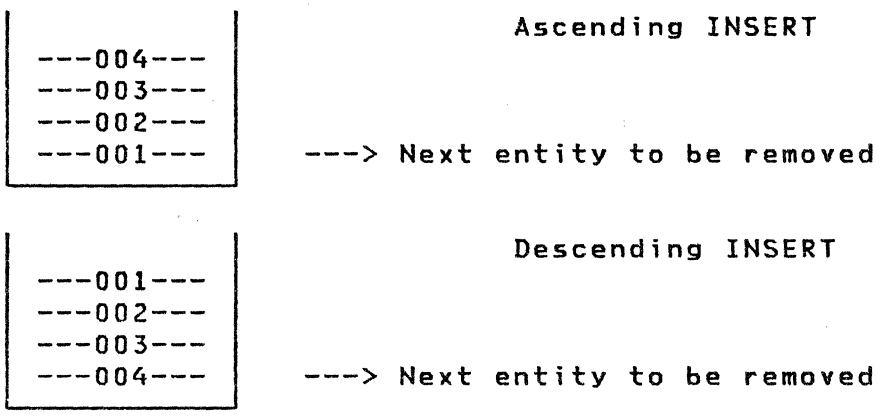


Figure N-4. Priority Ordering in Lists

Several statement types are included in FAPL to facilitate list processing. They are:

NEWLIST	Creates and initializes a new list
CREATE	Creates an entity
DESTROY	Destroys a list
DISCARD	Discards an entity and frees the storage allocated for it
INSERT	Adds an entity to a list
REMOVE	Removes an entity from a list
FIND	Finds a particular entity in a list and sets a pointer to it
SCAN	Searches a list and executes a group of statements as each list entry is addressed
PURGE	Removes all entities from a list and discards them

See the "Statements" section for a full definition of these statements. List-handling statement usage examples are shown in Figure N-5.

```
DCL TABLE PTR;                  /* DEFINES PTR VARIABLES FOR */
DCL ITEM_PTR PTR;               /* LIST NAME AND ENTITY      */
       .                        /* DEFAULT POINTER           */
       .
       .
ENTITY(ITEM),                   /* DEFINES AN ENTITY DATA    */
  2 R  BIT(4),                  /* STRUCTURE                 */
  2 S  BIT(4),
  2 T  CHAR(2);
       .
       .
       .
NEWLIST TABLE ENTRY_NAME(ITEM);
       .                        /* CREATES A NEW LIST        */
       .
       .
CREATE ITEM;                    /* CREATES A COPY OF ITEM    */
ITEM.T='AB';                    /* WITH DEFAULT POINTER      */
       .                        /* ITEM_PTR SET TO POINT     */
       .                        /* TO IT.  FIELD T IS SET TO */
       .                        /* AN INITIAL VALUE          */

INSERT ITEM IN TABLE;           /* ADDS ITEM TO TABLE        */
       .
       .
       .
FIND ITEM IN TABLE WHERE(ITEM.T='AB');
       .                        /* LOCATES A SPECIFIC ITEM   */
       .                        /* IN THE TABLE              */
       .
SCAN TABLE PTR(ITEM_PTR);       /* PROCESSES EACH ITEM IN    */
  IF ITEM.T = 'AB' THEN         /* THE TABLE AND TAKES THOSE */
    REMOVE ITEM FROM TABLE DISCARD; /* WITH A GIVEN          */
SCANEND;                        /* CRITERION OFF THE LIST    */
                                /* AND DISCARDS TEHM         */
```

Figure N-5. FAPL List-Handling Examples

Queues

Queues are implemented as lists in FAPL; queues are a special case of lists that are known to a scheduler. (See Appendix C, "The Execution Model," for more detail about queues and schedulers.) A scheduler selects queues to be serviced in an order based on implementation-dependent algorithms.

A NEWLIST statement, with the QUEUE option, is used to establish a new queue. (See the NEWLIST statement definition in the "Statements" section.)

CONTROL_BLOCK_DEFINITION Statement

Syntax:

```
CONTROL_BLOCK_DEFINITION:
  v [v] ...
END CONTROL_BLOCK_DEFINITION;
```

where v is an entity name or the name NCB.

Semantics:

The names listed in this statement are the only valid control blocks. Only control blocks are used as contexts for FSMs and in the USING clause of SEND statements.

Only one CONTROL_BLOCK_DEFINITION statement appears in a FAPL program.

Example:

```
CONTROL_BLOCK_DEFINITION:
  LSCB TGCB ERCB VRCB SCB
  NRCB DRCB TCCB
  NCB
END CONTROL_BLOCK_DEFINITION;
```

CREATE Statement

Syntax:

```
CREATE [p->]e;
```

where p is a pointer and e is an entity name. If p is omitted, the default pointer for e, e_PTR, is implied.

Semantics:

An attempt is made to obtain storage for a new copy of e. If storage is available, p, or the default pointer, is set to point to the new copy. The storage obtained for e is set to binary 0's. If storage is unavailable, p, or the default pointer, is set to NULL.

Examples:

```
CREATE A;
CREATE P->A;
```

In the first statement, A_PTR is set to the new entity or NULL; in the second statement, pointer P is set to the appropriate value and A_PTR is unchanged.

DESTROY Statement

Syntax:

```
DESTROY l;
```

where l is a list name or a pointer set to point to a list.

Semantics:

Any entries on l are removed and discarded. Storage used by the header for l is returned to the system; l is set to null.

Example:

```
DESTROY A;
```

DISCARD Statement

Syntax:

```
DISCARD [p->]e;
```

where p is a pointer and e is an entity name. If p is omitted, the default pointer for e, e_PTR, is implied.

Semantics:

The storage for the copy of e pointed to by p, or the default pointer, is returned to the system. If p is used, p is set to NULL and the default pointer is unaffected. If the default pointer is used, it is set to NULL.

Example:

```
DISCARD A;
```

ENTITY Statement

Syntax:

```
ENTITY(e),
  2 i [a]
  [,n i [a]] ... ;
```

where e is an entity name, n is a level number, i is an identifier, and a is an attribute list. The syntax for the ENTITY statement is similar to a PL/I DECLARE statement for a structure. The level number of the first element in the data structure is 2. The rest of the data structure definition follows the rules for a PL/I DECLARE statement.

Semantics:

An ENTITY statement defines the format of a data structure that is to be manipulated using the FAPL list-handling statements. The defined data structure is equivalent to a PL/I BASED structure with level 1 identifier, e, that is based on a default pointer named e_PTR.

Storage for a new copy of e is obtained using the CREATE statement and is freed using the DISCARD statement. The INSERT and REMOVE statements are used for manipulating entities on a list. The SCAN and FIND statements also reference entities.

Example:

```
ENTITY(A),
  2 B,
    3 C BIT(4),
    3 D BIT(4),
  2 E BIT(8);
```

FIND Statement

Syntax:

```
FIND [p->]e IN l WHERE(b);
```

where e is an entity name, p is a pointer, l is a list name or a pointer set to point to a list, and b is a Boolean expression. If p is omitted, the default pointer for e, e_PTR, is implied. If p is specified, fields in e referred to in b are qualified explicitly by p->.

Semantics:

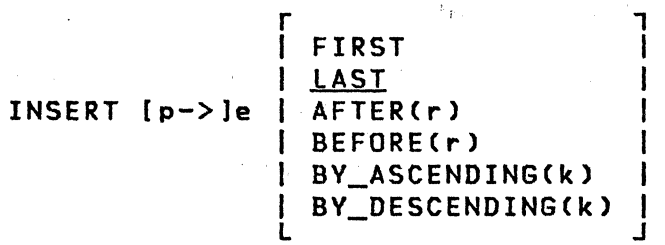
The list l is scanned from the first entry to the last entry. If an entity is found where b is true, then p or the default pointer is set to point to the entity, and the scan is stopped. If the list is empty, or if no entity is found with b true, then p, or the default pointer, is set to NULL. The default pointer is unaffected if p is used.

Examples:

```
FIND A IN B WHERE(A.C=D);
FIND P->A IN B WHERE(P->A.C=D);
```

INSERT Statement

Syntax:



where p is a pointer, e is an entity name, l is a list name or a pointer set to point to a list, r is a pointer set to point to an entity in l, and k is a field in e or a concatenation of fields in e that provides a priority key in the form of a bit string. If p is omitted, the default pointer for e, e_PTR, is implied.

Semantics:

Entity e is inserted in l. If no positional keyword is specified or if LAST is specified, e is inserted as the last entity in l. If FIRST is specified, e is inserted first in l. BEFORE indicates that e is to be inserted before the entity in l pointed to by r. AFTER indicates that e is to be inserted after the entity in l pointed to by r.

One entity is "before" another if it would be removed earlier in a series of REMOVE FIRST operations. For example, the entity returned by NEXT_ENTRY(p) is after p. The entity returned by FIRST_ENTRY is before all other entities on the list and the entity returned by LAST_ENTRY is after all other entities on the list.

If BY_ASCENDING is specified, e is inserted in l at the point where all entries before that point have k-values less than or equal to the k-value of the entity being added, and all entries after that point have k-values greater than the k-value of the entity being added. If BY_DESCENDING is specified, e is inserted in l at the point where all entries before that point have k-values greater than or equal to the k-value of the entity being added, and all entries after that point have k-values less than the k-value of the entity being added.

Examples:

```
INSERT A IN B;
INSERT P->A FIRST IN B;
INSERT A AFTER(C) IN B;
INSERT A BEFORE(C) IN B;
INSERT A BY_DESCENDING(A.K) IN B;
```

LOCK/UNLOCK Statement

Syntax:

```
LOCK l;

  s;[s;] ...

UNLOCK;
```

where l is a list name or a pointer set to point to a list and s is a statement.

Semantics:

The list l is unavailable to any other process for read or write access from the time the LOCK is executed until the UNLOCK is executed. Locking a list also locks all entities on the list. If l is already locked by another process when the LOCK is attempted, the LOCK statement does not complete execution until the list becomes available.

LOCKs are never nested. No statement between the LOCK and UNLOCK prevents the flow of control from passing through the UNLOCK statement (e.g., RETURN does not occur.)

Example:

```
LOCK A;
  IF ¬EMPTY(A) THEN
    REMOVE FIRST(B) FROM A;
UNLOCK;
```

NEWLIST Statement

Syntax:

```
                                    ┌                    ┐
                                    │ FIFO               │
NEWLIST [p->]l ENTRY_NAME(e)        │ BY_ASCENDING(k)    │ [QUEUE];
                                    │ BY_DESCENDING(k)   │
                                    └                    ┘
```

where p is a pointer, l is a list name, e is an entity name, and k is a field in e, or a concatenation of fields in e, that provide a priority key in the form of a bit string.

Semantics:

The name for the new list is l. Storage is obtained for a header for l, and a pointer to the header is set into a pointer variable named l. The only valid entity type that resides in l is e. The ordering specification (FIFO, BY_ASCENDING, or BY_DESCENDING) is an information field defining the normal processing mode used on the list. An information field is equivalent to a comment; it is an aid to the reader and has no execution-time effect. QUEUE specifies that l is known to the scheduler as a scheduled data queue. See the section, "Queues," for information on the distinction between lists and queues.

Examples:

```
NEWLIST A ENTRY_NAME(B);
NEWLIST A ENTRY_NAME(B) QUEUE;
NEWLIST A ENTRY_NAME(B) BY_ASCENDING(B.KEY);
```

PURGE Statement

Syntax:

```
PURGE l;
```

where l is a list name, or a pointer set to point to a list.

Semantics:

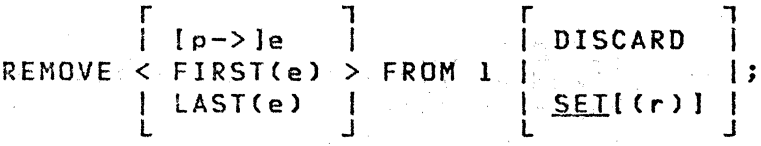
All entities in l are removed and discarded. If l is empty, no action occurs.

Example:

```
PURGE A;
```

REMOVE Statement

Syntax:



where p and r are pointers, e is an entity name, and l is a list name, or a pointer set to point to a list. If p is omitted, the default pointer for e, e_PTR, is implied. If r is omitted, it defaults to p if p is specified, and to e_PTR, otherwise.

Semantics:

An entity is removed from l. The keyword options specify which entity in l is to be removed. If the first option is used, the entity pointed to by p, or by the default pointer, is removed. FIRST(e) causes the first entry in l to be removed; LAST(e) causes the last entry to be removed. The DISCARD keyword causes the same function as the DISCARD statement to be performed after e is removed from l, i.e., the entity is discarded and p or the default pointer is set to NULL if the p->e or e option, respectively, is used. If DISCARD is used with the FIRST or LAST option, the entity is discarded and no pointers are affected. The SET keyword causes a pointer to be set to the entity that is removed.

Examples:

```
REMOVE A  FROM B;
REMOVE P->A FROM B;
REMOVE FIRST(A) FROM B DISCARD;
REMOVE LAST(A) FROM B SET(LAST_PTR);
```

SCAN Group

Syntax:

```
                  ┌                 ┐
                  |FROM FIRST       |
  SCAN l PTR(p)   |FROM ENTITY(r)   |  [WHILE(b1)] [UNTIL(b2)];
                  |FROM AFTER(r)    |
                  └                 ┘

    [s;[s;] ...]

  SCANEND;
```

where l is a list name, or a pointer set to point to a list; p and r are pointers; b1 and b2 are Boolean expressions, and s is a statement.

Both the WHILE and UNTIL clauses may exist in the same statement.

All references within b1, b2, and the SCAN group to elements of the entity being scanned are qualified by p->, unless p is the default pointer for the entity type in l.

If r is used and does not point to an entity on list l, the result of the SCAN is undefined.

Semantics:

Pointer p is set to point to each entity in l in succession, and the scan body statement(s) are executed for each entity pointed to. The beginning point of the scan is specified with the FROM clause. If the FROM clause is not present, or FROM FIRST is specified, p is set to point to the first entity in l, and then to each succeeding entity. The FROM ENTITY(r) option specifies that p is to be set initally to r, and then to each succeeding entry in l. The FROM AFTER(r) option specifies that p is to be set to the entity in l after the entity pointed to by r, and then set to each succeeding entry in l.

The WHILE and UNTIL options provide for terminating the SCAN before the end of l is reached. The WHILE option specifies that b1 is to be evaluated before the scan body is executed for the current p. If b1 is true, the scan is ended. The UNTIL option specifies that b2 is to be evaluated after the scan body is executed for the current p. If b2 is true, the scan is ended.

The SCAN statement is equivalent to the following logic:

```
if list is empty then
   exit;
set pointer p to beginning point;
do while condition b1 is true;
   execute scan body;
   if until-condition b2 is true or
        this is the last entity on the list then
      leave;
   else
      set pointer p to the next entity;
end of do while;
```

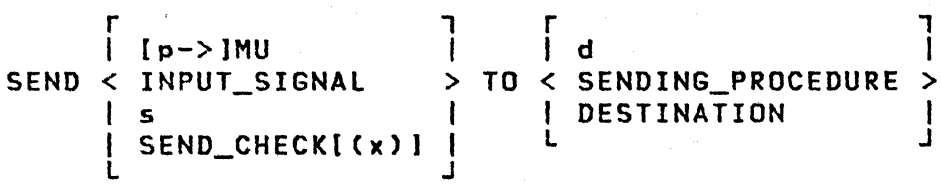
SCANs are nested up to a depth of three.

Examples:

```
SCAN A PTR(A_ENTRY_PTR);
  IF A_ENTRY.C=X THEN
    DISCARD A_ENTRY;
SCANEND;

SCAN A PTR(B) UNTIL(B->A_ENTRY.C=X);
SCANEND;
```

SEND Statement

Syntax:




```
[USING(parm[,parm]...)];
```

where d is a destination procedure name, p is a pointer, s is a character-string signal, and x is a sense code. Each parm has one of the following forms:

```
cbp = q
PARM_PTR = q
ORIGIN = proc
```

where cbp is a control block pointer name other than NCB_PTR, q is a pointer, and proc is a procedure name.

If p is omitted, the default pointer MU_PTR is implied. The signal, s, has up to thirty-one characters enclosed in quotes. The sense code, x, is expressed as a hexadecimal constant or as a variable, declared BIT(32), containing the sense code. A control-block pointer is the default pointer of any entity that is defined as a control block in the CONTROL_BLOCK_DEFINITION statement.

Semantics:

The SEND statement causes a message unit or character-string signal to be sent to another procedure by creating a dispatching queue entry (DQE) and inserting it on the dispatching queue. (See Appendix C, "The Execution Model," for a description of the dispatcher and the format of DQEs.) When a message unit is sent to another procedure, the sending procedure loses access to it, i.e., p or MU_PTR is set to NULL. When a signal is sent to another procedure, the receiving procedure does not receive the message unit.

The SEND statement is executed by performing the following steps in the indicated order:

1. A DQE is created.

2. The destination procedure name is moved into the DQE:

   - If SENDING_PROCEDURE is specified as the destination, the destination is the procedure that sent an input to start the subthread of the currently executing procedure (via SEND).

   - If DESTINATION is specified as the destination, the procedure name is taken from the variable named DESTINATION, which has been set to the destination procedure name elsewhere.

3. If no ORIGIN parameter appears in the USING clause, the name of the procedure that was invoked by the dispatcher to start the current subthread is moved into the DQE as the procedure that is initiating the SEND. If an ORIGIN parameter appears in the USING clause, the procedure name specified is moved into the DQE as the procedure that is initiating the SEND.

4. If an input signal is included in the SEND statement, the signal is moved into the DQE.

5. If the SEND_CHECK option is used, the sense code indicated is copied into the MUCB field, SEND_CHECK_SENSE, and the SEND_CHECK bit is set. (The MUCB is described in Appendix C.) If the sense code is omitted, only the SEND_CHECK bit is set; this option is used when SEND_CHECK_SENSE is set prior to the SEND statement. The SEND_CHECK option is used to inform the end user that an error has occurred.

6. If a message unit is being sent, the pointer to the current message unit is moved into the DQE. If a signal is being sent, this field of the DQE is set to NULL. The current pointers to all control blocks (as defined by the CONTROL_BLOCK_DEFINITION statement) except NCB, are moved into the DQE. This is done to provide the same execution environment for the sent-to procedure as exists in the sending procedure.

   An alternative control block pointer may be specified with the USING option for any control block that is normally carried by the SEND. If this option is used, the specified pointer is passed to the receiving procedure and the control block pointer remains unchanged in the current subthread.

   The USING option also allows specifying a PARM_PTR in order to pass a pointer to a parameter list to another procedure.

7. The DQE is placed last on the dispatching queue.

The recipient of the SEND and all procedures that it calls can check the name of the sender by using the DISPATCHED_BY built-in function. These procedures can also check any signal that was sent by using the INPUT function. No procedure in the subthread changes either of these values. The procedures have access to the MU, the parameter, and all control blocks that are passed by referring to the appropriate pointer; these values can be changed by any procedure in the subthread.

Examples:

```
SEND MU TO DESTINATION;
SEND 'RESET' TO D;
SEND SEND_CHECK(X'080D') TO SENDING_PROCEDURE;
SEND INPUT_SIGNAL TO D USING(SCB_PTR=P);
```

DISPATCHED_BY Function

Syntax:

```
DISPATCHED_BY(n[*])
```

where n is a procedure name or a procedure name prefix. A procedure name prefix is identified with an asterisk (*) following n.

Semantics:

This Boolean function returns a true value if the currently executing procedure received control because of a SEND issued in an execution subthread that was begun with procedure n (i.e., a SEND to procedure n initiated the subthread that sent to the currently executing subthread). If n specifies a procedure name prefix, indicated by an asterisk(*), a true value is returned if n is the beginning of the procedure name initiating the sending subthread.

Examples:

```
IF DISPATCHED_BY(A) THEN . . .
IF DISPATCHED_BY(A*) THEN . . .
```

EMPTY Function

Syntax:

```
EMPTY(l)
```

where l is a list name, or a pointer set to point to a list.

Semantics:

This Boolean function returns a true value if and only if the named list is empty.

Example:

```
IF EMPTY(Q_A) THEN . . .
```

FIRST_ENTRY Function

Syntax:

```
FIRST_ENTRY(l)
```

where l is a list name, or a pointer set to point to a list.

Semantics:

This function returns a pointer to the first entry in l. If the list is empty, the result is undefined.

Example:

```
IF ¬EMPTY(A) THEN
  B = FIRST_ENTRY(A)->A_ENTRY.C;
```

INPUT Function

Syntax:

```
INPUT(x)
```

where x may be one of the following:

```
RQ
RSP
SIGNAL
'character_string_signal'
```

Semantics:

This Boolean function returns a true value if x was sent (via SEND) to the procedure initiating the currently executing subthread. If x is SIGNAL, a true value is returned if any signal was sent. If x is a specific character_string_signal, a true value is returned if that specific signal was sent.

Two SNA specific input types may be specified as x (RQ or RSP) and for these a true value is returned if the RRI (Request-Response indicator) matches the specific input, and no signal was sent.

This function is the only way a signal can be tested. It is valid only within a procedure, not an FSM.

Examples:

```
IF INPUT(SIGNAL) THEN . . .
IF INPUT('A') THEN . . .
```

LAST_ENTRY Function

Syntax:

```
LAST_ENTRY(l)
```

where l is a list name, or a pointer set to point to a list.

Semantics:

This function returns a pointer to the last entry in the referenced list. If the list is empty, the result is undefined.

Example:

```
IF ¬EMPTY(A) THEN
  B = LAST_ENTRY(A)->A_ENTRY.C;
```

NEXT_ENTRY Function

Syntax:

```
NEXT_ENTRY(p)
```

where p is a pointer to an entity in a list.

Semantics:

The NEXT_ENTRY function returns a pointer to the entity in a list after the entity pointed to by p. If the referenced entity is the last entry in the list, the result is undefined.

Example:

```
/* P POINTS AT AN ENTITY IN A */
IF P¬=LAST_ENTRY(A) THEN
  B = NEXT_ENTRY(P)->A_ENTRY.C;
```

PREV_ENTRY Function

Syntax:

```
PREV_ENTRY(p)
```

where p is a pointer to an entity in a list.

Semantics:

The PREV_ENTRY function returns a pointer to the entity in a list before the entity pointed to by p. If the referenced entity is the first entry in the list, the result is undefined.

Example:

```
/* P POINTS AT AN ENTITY IN A */
IF P¬=FIRST_ENTRY(A) THEN
  B = PREV_ENTRY(P)->A_ENTRY.C;
```

SEND_OR_RECEIVE_CHECK Function

Syntax:

```
SEND_OR_RECEIVE_CHECK([p->]f[('c')])
```

where p is a pointer, f is a reference to an FSM, and c is a character-string signal.

Semantics:

This Boolean function returns a true value if the current input conditions and state of the specified FSM would cause the access of a SEND_OR_RECEIVE_CHECK indicator. If it returns a true value, the output code associated with the FSM entry is executed.

If c is present, the input condition FSMINPUT=c is true.

If p is present, it specifies a pointer to a control block of the type specified in the FSM_DEFINITION CONTEXT with which the FSM is associated; if it is not present, the default pointer for the CONTEXT control block is assumed.

More specifically, an action code at a matrix intersection is accessed based on the current state of the FSM and of input conditions. If the next-state indicator in the accessed action code is a SEND_OR_RECEIVE_CHECK indicator (>), the function returns a true value. If an output code is associated with the SEND_OR_RECEIVE_CHECK indicator, it is executed. If the next-state indicator is a CANNOT_OCCUR indicator (/), a state number, or a no-state-change indicator (-), the function returns a false value, and no output code is executed.

For more information on finite-state machines in FAPL, see the section, "Finite-State Machine (FSM) Representation in FAPL" (page N-41).

Example:

```
IF SEND_OR_RECIEVE_CHECK(#FSM_HDX) THEN
  DISCARD MU;
```

```
FSM_fsmname: FSM_DEFINITION CONTEXT(context_name);
 [ declarations ]
```

STATE ATTRIBUTES--->	[sa[,sa]]		[sa[,sa]]
[STATE NAMES--------->]	snam		snam
	.		.
	.		.
	snam	. . .	snam
[STATE NUMBERS------>]	snum		snum
INPUTS			
ic [,ic] . . .	ac[,ac[,ac]]		ac[,ac[,ac]]
ic [,ic] . . .	ac[,ac[,ac]]		ac[,ac[,ac]]
ic [,ic] . . .	ac[,ac[,ac]]		ac[,ac[,ac]]
.			
.			
.			
ic [,ic] . . .	ac[,ac[,ac]]		ac[,ac[,ac]]

MULTIPLE_ACTION_CODE	DEFINING CONDITION
1	Boolean_expression
2	Boolean_expression
3	Boolean_expression

OUTPUT CODE	FUNCTION
oc-1	FAPL statement(s)
	.
	.
	.
oc-n	FAPL statement(s)

```
END FSM_fsmname;
```

Notation definition:

```
ac  = action code name          snam  = state name component
ic  = input condition           snum  = state number
sa  = state attribute
```

An action code (ac) has the syntax ns[(oc)], where:

```
ns  = next state indicator
oc  = output code
```

Allowed next state indicators and associated action code formats are:

```
n[(oc)]  - normal state transition to state "n"
-[(oc)]  - same state transition
>[(oc)]  - error condition, no state change
/[NOTEn] - CANNOT_OCCUR condition, no state change
```

See the text for more details on the FSM definition syntax.

Figure N-6. State-transition Matrix Form FSM Definition Syntax

FINITE-STATE MACHINE (FSM) REPRESENTATION IN FAPL

FSM Names

FSM names always include the "FSM_" prefix followed by the specific FSM name.

```
FSM_fsmname
```

State-Transition Matrices

FAPL uses a state-transition matrix to represent FSMs.

The syntax of the state-transition matrix FSM definition is shown in Figure N-6. The column headings give the FSM state names (and attributes), 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, mnemonics are used in the matrices. A description of each of the elements within an FSM definition follows.

FSM Definition: The FSM_DEFINITION statement and a paired END statement delimit an FSM definition. Both statements contain the FSM name. The FSM_DEFINITION statement also specifies the context within which the FSM exists. The context is specified as a control block name (as defined by the CONTROL_BLOCK_DEFINITION statement) that will contain the current state of the FSM.

State Names: The state names associated with each state of the FSM are specified at the top of each column in the matrix. The state name consists of a single component, or multiple components arranged vertically. If the name contains more than one component, underscores are implied between each component.

State Attributes: Optionally, one or two state attributes are specified above the state name for each state of an FSM. Their use simplifies testing that an FSM is in one of several possible states (e.g., a state that allows receiving a message). See "Testing FSM State Attributes" for details on testing state attributes.

State Number: State numbers are specified in each column heading under the state name and are referred to within the matrix as next-state indicators.

Input Conditions: The inputs referred to within the matrix are defined outside the matrix. The input conditions are mnemonic names corresponding to Boolean expressions that are defined within an "FSM_INPUT_DEFINITION" section. See "FSM_INPUT_DEFINITION Statement" (Page N-43) for the syntax and the input definition at the end of Chapter 5 for an example input definition.

Input conditions preceded by an asterisk (*) denote an "any value allowed" (or "don't care") input; these are shown for descriptive completeness, but their presence has no effect. The not sign (¬) preceding an input condition indicates that a false condition is required. For each possible input condition, the FSM_INPUT_DEFINITION section includes one or more logical (Boolean) tests. Each mnemonic in the FSM_INPUT_DEFINITION is followed by a Boolean expression and is delimited by a semicolon (;). This Boolean expression is used at the time the FSM is called to determine if the input condition is true or false.

Input conditions of the form

s(t1[,t2]...)

(where s and t are alphanumeric strings) are handled specially. This input condition is equivalent to

s(t1)[,s(t2)]...

and each of the individual input conditions are in the FSM_INPUT_DEFINITION section. Further, if any of the t's is preceded by a not sign (¬), the ¬ is moved before the "s" associated with that "t." That is,

s(t1,¬t2)

becomes

s(t1),¬s(t2).

For example, the input conditions:

CT(BB,CD)

and

CT(BB,¬CD)

would have associated with them the two FSM_INPUT_DEFINITION lines:

CT(BB)

and

CT(CD)

and the two lines are equivalent to

CT(BB),CT(CD)

and

CT(BB),¬CT(CD),

respectively.

When input conditions for one logical input line overflow onto a second physical line of the matrix, a comma follows the last input condition on the first line to indicate that the next line is a continuation.

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.

FSM Input Definition

Syntax:

```
FSM_INPUT_DEFINITION:
  ic  b;
  [ic  b;] ...
END FSM_INPUT_DEFINITION;
```

where ic is an input condition and b is a Boolean expression. An input condition name is a character string of alphanumeric characters and the following special characters: '|&()_±. The input condition name may be up to 31 characters long.

Semantics:

The FSM_INPUT_DEFINITION statement defines all valid input conditions for FSMs.

Example:

See the input definition at the end of Chapter 5.

Action Codes: The action code (ac) specified at each matrix intersection has the syntax:

ns[(oc)]

The next-state indicator (ns) may have one of four type values. A number indicates the next state that the FSM assumes if the matrix intersection is addressed. A dash (-) indicates that no state change is to occur. A greater-than sign (>) indicates an error condition. (See the SEND_OR_RECEIVE_CHECK function description for details on how an error condition of this type is tested.) A slash (/)

indicates that this matrix intersection cannot be selected because of previous checks in a procedure or other FSM in this node. An optional note indicates where the check occurs.

The table in Figure N-7 summarizes differences among the next-state indicators.

ns	Can an output code be associated with it?	Return value of SEND_OR_RECEIVE_CHECK	Action in CALL
n	yes	false	change to indicated state
-	yes	false	no state change
>	yes	true	no state change
/	no	false	error condition

Figure N-7. Next-State Indicators

Output Code: An output code (oc) optionally occurs in a matrix-intersection action code and indicates the output function to be executed. The output code is one to six characters in length, and is defined in the OUTPUT_CODE section following the matrix in the FSM definition box. The output-code definition is a set of FAPL statements that are executed if the output code is addressed in the matrix.

Multiple Action Codes: In certain cases, it is possible to reduce the number of rows in a matrix by merging rows whose state transitions and output actions differ only for one input in their input sets (or for some other variable, not necessarily passed explicitly as input to the FSM, that can be tested). In these cases, the rows are merged by specifying multiple action codes separated by commas within each applicable matrix intersection. A MULTIPLE_ACTION_CODE definition section in the FSM then determines which of the multiple action codes applies for a specific invocation of the FSM.

FSM Initialization

When a control block is created, the STATES array is initialized to 1. This means that all FSMs have an initial state of 1. If a different reset state is wanted, specific initialization is required.

FSM Input Signals

Input character-string signals to an FSM are checked for in the FSM_INPUT_DEFINITION section by the test

FSMINPUT = signal_name.

When an input character-string signal is sent to an FSM, input condition rows in the matrix that do not test for an input signal are implied not to be met. An input condition that tests an FSM input signal is enclosed by single quotes (').

Calling FSMs

Syntax:

```
CALL FSM_n[('i')];
```

where n is an FSM name and i is a character-string signal. Only literal strings can be passed to an FSM.

Semantics:

The CALL FSM statement executes the procedure generated from an FSM definition. Execution of this procedure causes selection of an FSM action code based on the current state of the FSM and the input line that is true. The FSM procedure scans 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 SEND_OR_RECEIVE_CHECK (>), the CALL acts as if a no-state-change indicator (-) was encountered. If the next-state indicator is a CANNOT_OCCUR indicator (/), the FSM signals an execution-time error.

If an input condition in the FSM is defined in the FSM_INPUT_DEFINITION as FSMINPUT='character_string', the input condition is true if i is equal to the specified character string. If i is included in the call, any input line, without an input condition that tests FSMINPUT, is false.

Examples:

```
CALL FSM_A;
CALL FSM_A('RESET');
```

Testing FSM States

Syntax:

```
[p->]FSM_n = sn
```

where p is a pointer, n is an FSM name, and sn is a state name for the FSM.

Semantics:

This expression is used in an IF or WHEN statement to test the current state of an FSM. The test determines if the FSM is in the specific state named in the comparison expression.

If p is present, it specifies a pointer to a control block of the type specified in the FSM_DEFINITION CONTEXT with which the FSM is associated; if it is not present, the default pointer for the CONTEXT control block is assumed.

Examples:

```
IF FSM_A = STATE_X THEN ...

WHEN (FSM_A = STATE_Y) ...
```

Testing FSM State Attributes

Syntax:

```
                                ┌ ┐      ┌ ┐
[p->]FSM_n = ( [p->]FSM_n.sa[,  |¬|sa[,   |¬|sa] )
               └ ┘              |*|      |*|
                                └ ┘      └ ┘
```

where p is a pointer, n is an FSM name, and sa is a state attribute name.

Semantics:

The state-attribute comparison expression is used in an IF or WHEN comparison expression to test the attributes of the current state of the FSM. The test determines if the specified attributes are associated with the current state of the FSM. An asterisk (*) preceding an attribute indicates "any value allowed" (a "don't care" condition).

If p is present, it specifies a pointer to a control block of the type specified in the FSM_DEFINITION CONTEXT with which the FSM is associated; if it is not present, the default pointer for the CONTEXT control block is assumed.

Examples:

```
IF FSM_A = (*S,R) THEN ...

WHEN (FSM_A = (S,¬R)) ...
```

If "S" and "R" are attributes associated with various states of FSM_A that indicate if sending (S) or receiving (R) are allowed in a particular state, the IF statement determines if the FSM is in a state that allows receiving, and the WHEN statement determines if the FSM is in a state that allows sending, but not receiving.

Detecting Potential FSM Check Conditions

An FSM may be invoked for the express purpose of checking if a CALL to the FSM causes a SEND_OR_RECEIVE_CHECK indicator to be accessed. This is done by testing an FSM with the built-in function SEND_OR_RECEIVE_CHECK. (See the "Functions" section.)

This form of FSM reference causes the procedure generated from an FSM definition to be executed similarly to a CALL of an FSM. An action code at a matrix intersection is accessed based on the current state of the FSM and input conditions. If the next-state indicator in the accessed action code is a SEND_OR_RECEIVE_CHECK indicator (>), the function returns a true value. If an output code is associated with the SEND_OR_RECEIVE_CHECK indicator, it is executed. If the next-state indicator is other than a SEND_OR_RECEIVE_CHECK indicator, the function returns a false value and no output code is executed.

This facility for detecting an impending SEND_OR_RECEIVE_CHECK avoids the need to "back out" of state changes of previously called FSMs when a check occurs in an FSM. That is, the procedure can verify that all FSMs will accept the current input before advancing any to the next state based on this input.

STATE-TRANSITION GRAPHS

Finite-state machines can be described by state-transition matrices, as is done in FAPL, or by state-transition graphs. This section describes the conventions used in state-transition graphs in this book.

The terminology used is that of the "Finite-State Machines" section. Most terms are defined again as they are used; FNS refers to the next-state function and FOUT refers to the output function.

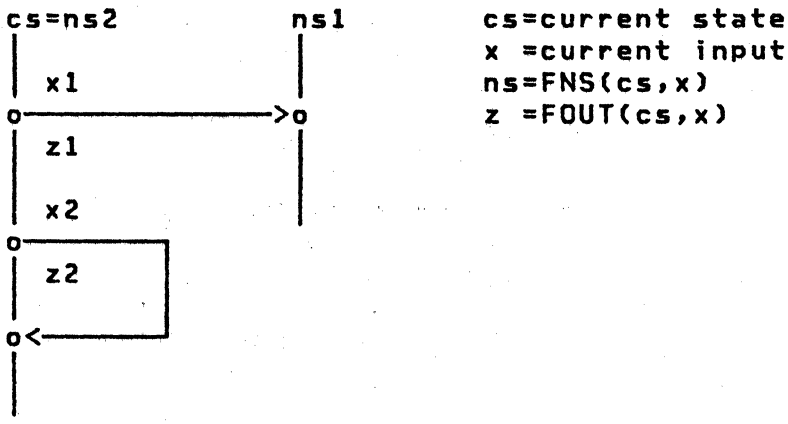


Figure N-8. Basic State-Transition Graph

Figure N-8 shows a simple state-transition graph. States are indicated by vertical lines (state lines), suitably named at the top (and/or the bottom). Each transition between states (i.e., ns=FNS(cs,x)) is represented by a horizontal arrow having these properties:

- The tail of the arrow starts at a circle on the cs state line.
- The head of the arrow ends at a circle on the ns state line.
- The input x associated with the transition appears at the tail, directly above the arrow.
- The output z=FOUT(cs,x) associated with the pending transition appears at the tail directly below the arrow.
- If ns=FNS(cs,x)=cs, the transition arrow is represented as a loop.

When multiple-stream outputs are used, the output stream destination information can be added to each state name and transition as shown in Figure N-9.

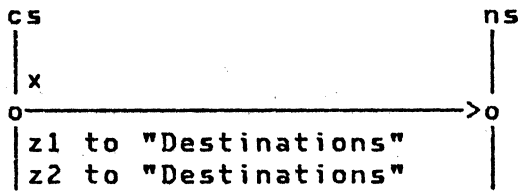


Figure N-9. Multiple-Stream Outputs in a State-Transition Graph

For FSMs with multiple input streams arriving as output from different source FSMs, it can be helpful to identify the source of every input on the transition graph as shown if Figure N-10.

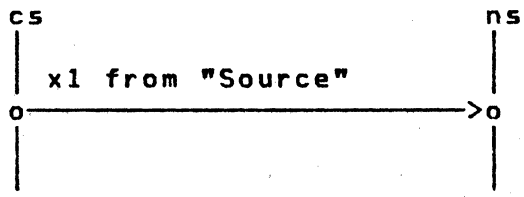


Figure N-10. Multiple-Stream Input in a State-Transition Graph

It sometimes occurs that for a particular input, both FNS and FOUT are independent of the current state. Such a situation is referred to as state-independent transitions and can be formally described as follows:

Let FSM = <S, X, Z, FNS, FOUT> have the following property:

For some Sk = {s1,s2,...,sk}, a subset of S, and some x in X

- FNS(s1,x) = FNS(s2,x) =...= FNS(sk,x) = sj
- FOUT(s1,x) = FOUT(s2,x) =...=FOUT(sk,x) = zj

State-independent transitions can be represented in a state-transition graph as shown in Figure N-11.

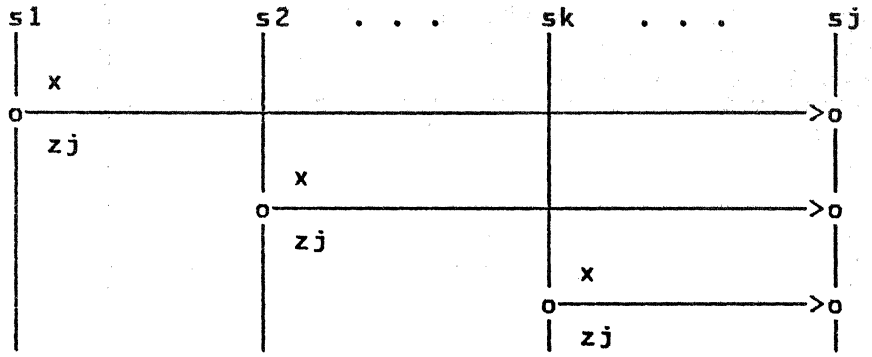


Figure N-11. State Independent Transitions

In order to simplify the state transition graph of an FSM, the set of k single-input transition arrows (each with a distinct source state in Sk) can be represented by a single, closed tail, broad arrow as shown in Figure N-12.

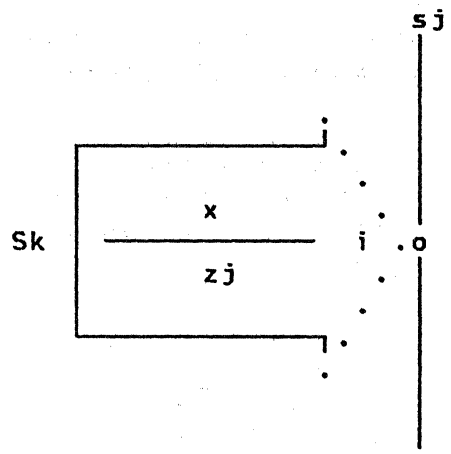


Figure N-12. Broad Arrow

The label on the closed tail of a broad arrow is an expression explicitly denoting Sk.

The integer, i, in the head of the arrow distinguishes it from other closed broad arrows in the same graph; in addition, the integer is listed next to an isolated circle on the state line for each state in Sk. Figure N-13 illustrates these notational conventions.

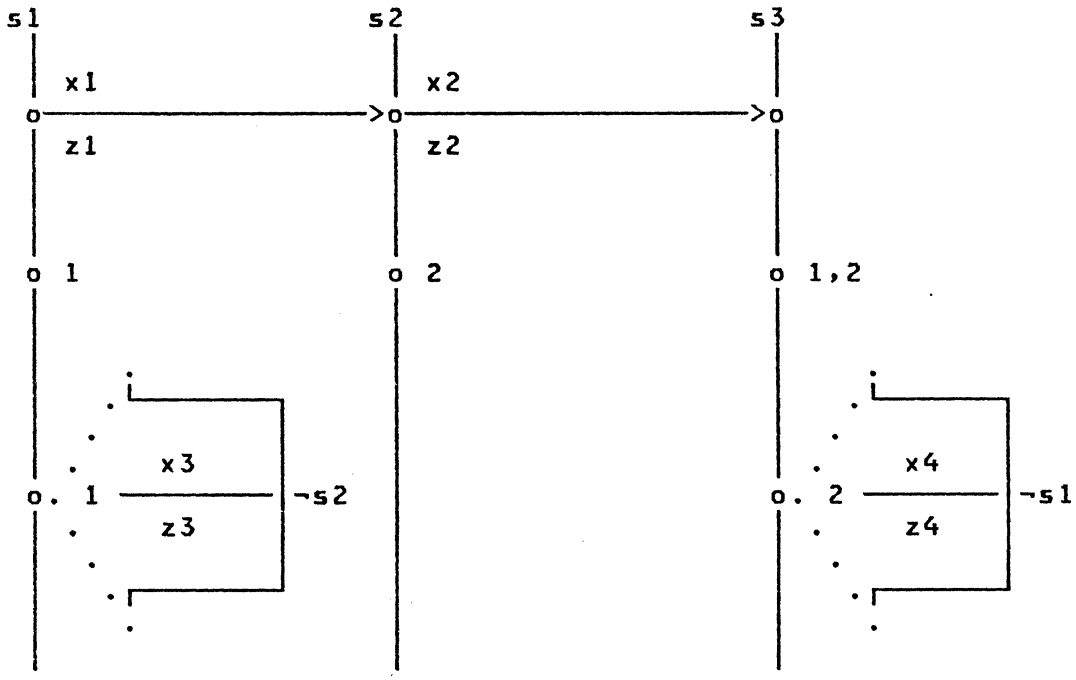


Figure N-13. Example of a State-Transition Graph with Multiple Broad Arrows

If Sk = S, the tail of the arrow is left open; because all states are source states, tail labels, arrowhead integers, and isolated circles are not used. An open broad arrow is shown in Figure N-14.

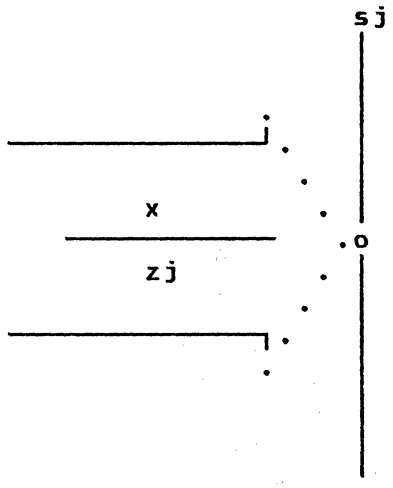


Figure N-14. Open Broad Arrow

This page intentionally left blank

Special Characters

A

D

E

F

G

H

I

L

Q

R

S

T

U

APPENDIX T. TERMINOLOGY: ACRONYMS AND ABBREVIATIONS

A	address (SDLC)
ABCONN	ABANDON CONNECTION
ABCONNOUT	ABANDON CONNECT OUT
ACTCDRM	ACTIVATE CROSS-DOMAIN RESOURCE MANAGER
ACTCONNIN	ACTIVATE CONNECT IN
ACTE	activate ERP
ACTLINK	ACTIVATE LINK
ACTLU	ACTIVATE LOGICAL UNIT
ACTPU	ACTIVATE PHYSICAL UNIT
ACTTRACE	ACTIVATE TRACE
ADDLINKSTA	ADD LINK STATION
ADDR	address
ADJ	adjacent
ALS	adjacent link station
ANA	ASSIGN NETWORK ADDRESSES
ASCII	American Standard Code for Information Interchange
BB	Begin Bracket
BBI	Begin Bracket indicator
BBIU	Begin-BIU
BBIUI	Begin-BIU indicator
BC	Begin Chain
BCI	Begin Chain indicator
BF	boundary function
BFSCB	boundary function session control block
BIN	binary
BIND	BIND SESSION
BINDF	BIND FAILURE
BIS	BRACKET INITIATION STOPPED
BIU	basic information unit
BLU	basic link unit
BSM	bracket state manager
BTU	basic transmission unit
BTUCB	basic transmission unit control block
C	Control (SDLC)
CCITT	International Consultative Committee for Telegraph and Telephone
CD	Change Direction, cross-domain
CDCINIT	CROSS-DOMAIN CONTROL INITIATE
CDCSESS	cross-domain control session requests
CDI	Change Direction indicator
CDINIT	CROSS-DOMAIN INITIATE
CDRM	cross-domain resource manager
CDSESSEND	CROSS-DOMAIN SESSION ENDED
CDSESSSF	CROSS-DOMAIN SESSION SETUP FAILURE
CDSESSST	CROSS-DOMAIN SESSION STARTED
CDSESSTF	CROSS-DOMAIN SESSION TAKEDOWN FAILURE

CDTAKED	CROSS-DOMAIN TAKEDOWN
CDTAKEDC	CROSS-DOMAIN TAKEDOWN COMPLETE
CDTERM	CROSS-DOMAIN TERMINATE
CINIT	CONTROL INITIATE
CLEANUP	CLEAN UP SESSION
CNM	communication network management
CNMA	communication network management application
CNMS	communication network management services
CONNOUT	CONNECT OUT
COS	class of service
CP	control point
CPCB	control point control block
CPMGR	connection point manager
CRV	CRYPTOGRAPHY VERIFICATION
CS	configuration services
CSC	common session control
CSI	Code Selection indicator
CT	correlation table
CTERM	CONTROL TERMINATE
DACTCDRM	DEACTIVATE CROSS-DOMAIN RESOURCE MANAGER
DACTCONNIN	DEACTIVATE CONNECT IN
DACTLINK	DEACTIVATE LINK
DACTLU	DEACTIVATE LOGICAL UNIT
DACTPU	DEACTIVATE PHYSICAL UNIT
DACTTRACE	DEACTIVATE TRACE
DAF	Destination Address field
DAF'	DAF prime
DCE	Data Circuit-terminating Equipment for a CCITT X.21 connection
DCF	Data Count field
DCL	DECLARE (PL/I)
DEC_WS	Decrement Window Size
DEF	Destination Element field
DELETENR	DELETE NETWORK RESOURCE
DES	Data Encryption Standard
DFC	data flow control
Disc	Disconnect (SDLC)
DISPSTOR	DISPLAY STORAGE
DLC	data link control
DLU	destination LU
DQ	dequeue
DRCB	domain resource control block
DR1	Definite Response 1
DR1I	Definite Response 1 indicator
DR2	Definite Response 2
DR2I	Definite Response 2 indicator
DSRLST	DIRECT SEARCH LIST
DT	data traffic
DTE	Data Terminal Equipment for a CCITT X.21 connection
DUMPINIT	DUMP INITIAL

EA	element address
EB	End Bracket
EBCDIC	extended binary coded decimal interchange code
EBI	End Bracket indicator
EBIU	End-BIU
EBIUI	End-BIU indicator
EC	End Chain
ECI	End Chain indicator
ED	Enciphered Data
EDI	Enciphered Data indicator
EFI	Expedited Flow indicator
EIA	Electronic Industries Association
ER	Exception Response requested
ER_INOP	EXPLICIT ROUTE INOPERATIVE
ERC	explicit route control
ERCB	explicit route control block
ERI	Exception Response indicator
ERN	explicit route number
ERP	error recovery procedure(s)
ESLOW	ENTERING SLOWDOWN
EXCP	exception
EXECTEST	EXECUTE TEST
EXP,Exp	expedited
EXR	EXCEPTION REQUEST
EXRD	EXR indicating definite-response requested
EXRE	EXR indicating exception-response requested
EXRN	EXR indicating no-response requested
EXSLOW	EXITING SLOWDOWN

F	flag (SDLC)
FAPL	Format and Protocol Language
FCS	frame check sequence (SDLC)
FDX	full-duplex
FF	flip-flop
FI	Format Indicator
FID	Format Identification (field)
FIFO	first-in, first-out
FM	function management
FMD	function management data
FMDS	function management data services
FMH	FM header
FMP	FM profile
FNA	FREE NETWORK ADDRESSES
FSM	finite-state machine
FSP	first speaker

HDX	half-duplex
HDX-CONT	HDX contention
HDX-FF	HDX flip-flop
HSCB	half-session control block
HSID	half-session identification

I	initiate
ID	identifier, identification
IERN	Initial Explicit Route Number field
ILU	initiating logical unit (LU sending INIT)
INIT	INITIATE, initial, initialize
INITPROC	INITIATE PROCEDURE
INOP	INOPERATIVE
IPL	initial program load
IPLINIT	IPL INITIAL
IPR	ISOLATED PACING RESPONSE
LCP	LOST CONTROL POINT
LD	Lost Data
LDI	Lost Data indicator
LDREQD	NS LOAD REQUIRED
LIFO	last-in, first-out
LSA	LOST SUBAREA
LSCB	link station control block
LSID	Local Session Identification
LU	logical unit
LUSTAT	LOGICAL UNIT STATUS
MGR	manager
MN&MA	management and maintenance services
MPF	Mapping field
MU	message unit
MUCB	message unit control block
N_PRTY	network priority
NA	not available
NA,na	network address
NAU	network addressable unit
NC	network control
NCB	node control block
NEG	negative
NG	no good
nn	network name
NORM,Norm	normal
NRCB	node resource control block
NS	network services
NS(c)	network services, configuration services
NSH	NS header
NS(ma)	network services, maintenance services
NS(me)	network services, measurement services
NS(mn)	network services, management services
NSPE	NETWORK SERVICES PROCEDURE ERROR
NS(s)	network services, session services
NSLSA	NETWORK SERVICES LOST SUBAREA
NTWK	network

O	Orderly
OAF	Origin Address field
OAF'	OAF prime
OLU	origin LU
OSAF	Origin Subarea field
P	primary
PAC	Pacing Request, Pacing Response (value of PI in RH)
PATHCB	path control block
PC	path control
PCCB	path control control block
PCID	procedure correlation identification
PD	Padded Data
PDI	Padded Data indicator
PI	Pacing indicator
PIU	path information unit
PLU	primary logical unit
PL/I	Programming Language/I
POS	positive
PPU	primary physical unit (PU at the node supporting DLC.PRI)
PRI	primary
PRID	procedure related identifier
PROCSTAT	PROCEDURE STATUS
PS	presentation services
PTR	pointer
PU	physical unit
PUCP	physical unit control point
PU_Tp	physical unit type "p" (p=1,2,4,5)
Q	queue, queued, quiesced
QC	QUIESCE COMPLETE
QEC	QUIESCE AT END OF CHAIN
QR	Queued Response
QRI	Queued Response indicator
R	receive, receiving
RC	return code
RCV	receive
RECFMS	RECORD FORMATTED MAINTENANCE STATISTICS
RECMD	RECORD MEASUREMENT DATA
RECMS	RECORD MAINTENANCE STATISTICS
RECSTOR	RECORD STORAGE
RECTD	RECORD TEST DATA
RECTR	RECORD TEST RESULTS
RECTRD	RECORD TRACE DATA
RELQ	RELEASE QUIESCE
REQACTLU	REQUEST ACTIVATE LOGICAL UNIT
REQCONT	REQUEST CONTACT
REQDISCONT	REQUEST DISCONTACT

REQECHO	REQUEST ECHO TEST
REQFNA	REQUEST FREE NETWORK ADDRESS
REQMS	REQUEST MAINTENANCE STATISTICS
REQTEST	REQUEST TEST
RERN	reverse explicit route number
RES	resource(s)
REX	route extension
RH	request/response header
RNAA	REQUEST NETWORK ADDRESS ASSIGNMENT
RPO	REMOTE POWER OFF
RQ	request
RQD	RQ indicating definite-response requested
RQE	RQ indicating exception-response requested
RQN	RQ indicating no-response requested
RQR	REQUEST RECOVERY
RRI	Request/Response indicator
RSHUTD	REQUEST SHUTDOWN
RSP	response
RTI	Response Type indicator
RTR	READY TO RECEIVE
RU	request/response unit
S	secondary, sending
SA	subarea address
SBI	STOP BRACKET INITIATION
SC	session control
SCB	session control block
SCS	SNA character string
SD	Sense Data Included
SDI	Sense Data Included indicator
SDLC	Synchronous Data Link Control
SDT	START DATA TRAFFIC
SEC	secondary
SESS	session
SESSEND	SESSION ENDED
SESSST	SESSION STARTED
SETCV	SET CONTROL VECTOR
SHUTC	SHUTDOWN COMPLETE
SHUTD	SHUTDOWN
SID	session identification
SIG	SIGNAL
SLU	secondary logical unit
SNA	Systems Network Architecture
SNAI	SNA indicator
SNC	sense code
SNF	Sequence Number field
SNRM	Set Normal Response Mode (SDLC)
SNS	session network services
SON	session outage notification
SPU	secondary physical unit (PU at the node supporting DLC.SEC)
SQN	sequence number
SS	session services

SSCP	system services control point
STA	station
STARTMEAS	START MEASUREMENT
STOPMEAS	STOP MEASUREMENT
STSN	SET AND TEST SEQUENCE NUMBERS
SVC	services
TC	transmission control
TCCB	transmission control control block
TERM	TERMINATE, termination
TG	transmission group
TGC	transmission group control
TGCB	transmission group control block
TGN	transmission group number
TH	transmission header
TLU	terminating logical unit (LU sending TERM)
Tp	type p
TPF	Transmission Priority field
TS	transmission services
TSP	TS profile
UNBIND	UNBIND SESSION
UNBINDF	UNBIND FAILURE
UPM	undefined protocol machine
URC	user request correlation
VR	virtual route
VR_CWI	Virtual Route Change Window indicator
VR_CWRI	Virtual Route Change Window Reply indicator
VR_INOP	VIRTUAL ROUTE INOPERATIVE
VR_RWI	Virtual Route Reset Window indicator
VRC	virtual route control
VRCB	virtual route control block
VRID	virtual route identifier
VRN	virtual route number
VRPRQ	Virtual Route Pacing Request indicator
VRPRS	Virtual Route Pacing Response indicator
XID	Exchange Station Identification (SDLC)

Systems Network Architecture Format and
Protocol Reference Manual:
Architectural Logic
Order No. SC30-3112-2

READER'S COMMENT FORM

This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

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 would like a reply, complete the following (Please Print):

Your Name _____ Date _____

Company Name _____

Department _____

Street Address _____

City _____ State _____ Zip Code _____

Thank you for your cooperation. No postage stamp is necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Cut or Fold Along This Line

Order No. SC30-3112-2

ADDITIONAL COMMENTS:

Please Do Not Staple

Fold and tape ..... Fold and tape

First Class
Permit 40
Armonk
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Dept. E02
P.O. Box 12195
Research Triangle Park
North Carolina 27709

Fold and tape ..... Fold and tape

IBM®

Cut or Fold Along Line

SNA Format and Protocol Reference Manual: Architectural Logic (File S370-30) Printed in U.S.A. Order No. SC30-3112-2

SNA Format and Protocol Reference Manual: Architectural Logic Printed in U.S.A. Order No. SC30-3112

SC30-3112-2